



*The*

# LINUX ENTERPRISE CLUSTER

BUILD A **HIGHLY AVAILABLE** CLUSTER WITH  
COMMODITY HARDWARE AND FREE SOFTWARE



Karl Kopper



NO STARCH  
PRESS



## **THE LINUX ENTERPRISE CLUSTER**



# THE LINUX ENTERPRISE CLUSTER

**Build a Highly Available Cluster  
with Commodity Hardware  
and Free Software**

by Karl Kopper



San Francisco

**THE LINUX ENTERPRISE CLUSTER.** Copyright © 2005 by Karl Kopper.

All rights reserved. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the copyright owner and the publisher.



Printed on recycled paper in the United States of America

1 2 3 4 5 6 7 8 9 10 – 07 06 05 04

No Starch Press and the No Starch Press logo are registered trademarks of No Starch Press, Inc. Other product and company names mentioned herein may be the trademarks of their respective owners. Rather than use a trademark symbol with every occurrence of a trademarked name, we are using the names only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

Publisher: William Pollock

Managing Editor: Karol Jurado

Production Manager: Susan Berge

Cover and Interior Design: Octopod Studios

Developmental Editor: William Pollock

Technical Reviewers: Brian Elliott Finley, Peter Holzleitner, Simon “Horms” Hormann, Chuck Lever, Joseph Mack, Matt Massie, Martin Pool, Alan Robertson, John Mark Walker, and Brian Ward

Copyeditors: Andy Carroll and Bonnie Granat

Illustrator for part openers: Nate Grundmann

Compositor: Riley Hoffman

Proofreader: Stephanie Provines

Indexer: Ted Laux

For information on book distributors or translations, please contact No Starch Press, Inc. directly:

No Starch Press, Inc.

555 De Haro Street, Suite 250, San Francisco, CA 94107

phone: 415.863.9900; fax: 415.863.9950; [info@nostarch.com](mailto:info@nostarch.com); <http://www.nostarch.com>

The information in this book is distributed on an “As Is” basis, without warranty. While every precaution has been taken in the preparation of this work, neither the author nor No Starch Press, Inc. shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in it.

*Library of Congress Cataloging-in-Publication Data*

Kopper, Karl.

The Linux Enterprise Cluster : build a highly available cluster with commodity hardware and free software / Karl Kopper.

p. cm.

Includes index.

ISBN 1-59327-036-4

1. Linux. 2. Parallel processing (Electronic computers) 3. Electronic data processing--Distributed processing. 4. Cluster analysis. I. Title.

QA76.58K67 2005

005.26'8--dc22

2003023352

This book is dedicated to all of the free software developers who wrote the software that made it possible for me to build a free, highly available cluster so I no longer had to work nights and weekends to keep the business system up and running.

## **TECHNICAL REVIEWERS**

Alan Robertson (Heartbeat)

Simon “Horms” Horman (Heartbeat + LVS)

Joseph Mack (LVS)

Matt Massie (Ganglia)

Charles Lever (NFS)

Martin Pool (SSH + rsync)

Brian Elliot Finley (SystemImager)

Peter Holzleitner (Mon + SNMP)

John Mark Walker (RPM, FTP, Booting, Linux Administration)

Brian Ward (Linux Kernel)

## BRIEF CONTENTS

<b>Introduction</b>		<b>Part II: High Availability</b>
	1	
<b>Primer</b>		<b>Chapter 4</b>
	9	Synchronizing Servers with rsync and SSH
		69
<b>Part I: Cluster Resources</b>		
<b>Chapter 1</b>		<b>Chapter 5</b>
Starting Services		Cloning Systems with SystemImager
15		95
<b>Chapter 2</b>		<b>Chapter 6</b>
Handling Packets		Heartbeat Introduction and Theory
33		111
<b>Chapter 3</b>		<b>Chapter 7</b>
Compiling the Kernel		A Sample Heartbeat Configuration
53		131

<b>Chapter 8</b>	<b>Chapter 14</b>
Heartbeat Resources and Maintenance	The Load Balancer
147	239
<b>Chapter 9</b>	<b>Chapter 15</b>
Stonith and ipfail	The High-Availability Cluster
163	255
<b>Part III: Cluster Theory and Practice</b>	<b>Chapter 16</b>
	The Network File System
	279
<b>Chapter 10</b>	<b>Part IV: Maintenance and Monitoring</b>
How to Build a Linux Enterprise Cluster	
185	
<b>Chapter 11</b>	<b>Chapter 17</b>
The Linux Virtual Server: Introduction and Theory	The Simple Network Management Protocol and Mon
193	301
<b>Chapter 12</b>	<b>Chapter 18</b>
The LVS-NAT Cluster	Ganglia
209	327
<b>Chapter 13</b>	<b>Chapter 19</b>
The LVS-DR Cluster	Case Studies in Cluster Administration
231	341
<b>Chapter 20</b>	<b>Chapter 20</b>
	The Linux Cluster Environment
	359

<b>Appendix A</b> Downloading Software from the Internet (from a Text Terminal) 369	<b>Appendix D</b> Strategies for Dependency Failures 387
<b>Appendix B</b> Troubleshooting with the tcpdump Utility 375	<b>Appendix E</b> Other Potential Cluster Filesystems and Lock Arbitration Methods 395
<b>Appendix C</b> Adding Network Interface Cards to Your System 379	<b>Appendix F</b> LVS Clusters and the Apache Configuration File 397
	<b>Index</b> 407



# CONTENTS IN DETAIL

<b>INTRODUCTION</b>	<b>1</b>
Properties of a Linux Enterprise Cluster .....	2
Architecture of the Linux Enterprise Cluster .....	4
The Load Balancer .....	4
The Shared Storage Device .....	5
The Print Server .....	5
The Cluster Node Manager .....	5
No Single Point of Failure .....	6
In Conclusion .....	7
<b>PRIMER</b>	<b>9</b>
High Availability Terminology .....	10
Linux Enterprise Cluster Terminology .....	10
<b>PART I: CLUSTER RESOURCES</b>	
<b>1</b>	
<b>STARTING SERVICES</b>	<b>15</b>
How Do Cluster Services Get Started? .....	15
Starting Services with init .....	16
The /etc/inittab File .....	16
Respawning Services with init .....	18
Managing the init Script Symbolic Links with chkconfig .....	18
Managing the init Script Symbolic Links with ntsysv .....	20
Removing Services You Do Not Need .....	21
Using the Red Hat init Scripts on Cluster Nodes .....	21
In Conclusion .....	31
<b>2</b>	
<b>HANDLING PACKETS</b>	<b>33</b>
Netfilter .....	34
A Brief History of Netfilter .....	35
Setting the Default Chain Policy .....	38
Using iptables and ipchains .....	39
Clear Existing Rules .....	39
Set the Default INPUT Chain Policy .....	39
FTP .....	40
Passive FTP .....	41
DNS .....	41

Telnet .....	42
SSH .....	42
Email .....	42
HTTP .....	43
ICMP .....	43
Review and Save Your Rules .....	43
Routing Packets with the Linux Kernel .....	44
Matching Packets for a Particular Destination Network .....	45
Matching Packets for a Particular Destination Host .....	45
Matching Packets for Any Destination or Host .....	46
To View Your Routing Rules .....	47
Making Your Routing Rules Permanent .....	47
iptables and Routing Resource Script .....	47
The ip Command .....	50
In Conclusion .....	51

### **3 COMPILING THE KERNEL 53**

What You Will Need .....	54
Step 1: Get the Source Code .....	54
Using the Stock Kernel .....	54
Using the Kernel Supplied with Your Distribution .....	55
Decide Which Kernel Version to Use .....	55
Step 2: Set the Options You Want .....	56
Installing a New Kernel .....	56
Upgrading or Patching the Kernel .....	56
Upgrading a Kernel from a Distribution Vendor .....	57
Set Your Kernel Options .....	57
Step 3: Compile the Code .....	59
Step 4: Install the Object Code and Configuration File .....	60
Install the System.map .....	61
Save the Kernel Configuration File .....	62
Step 5: Configure Your Boot Loader .....	62
In Conclusion .....	65

## **PART II: HIGH AVAILABILITY**

### **4 SYNCHRONIZING SERVERS WITH RSYNC AND SSH 69**

rsync .....	70
Open SSH 2 and rsync .....	71
SSH: A Secure Data Transport .....	71
SSH Encryption Keys .....	72
Establishing the Two-Way Trust Relationship: Part 1 .....	73
Establishing the Two-Way Trust Relationship: Part 2 .....	76
Two-Node SSH Client-Server Recipe .....	77

Create a User Account That Will Own the Data Files .....	78
Configure the Open SSH2 Host Key on the SSH Server .....	79
Create a User Encryption Key for This New Account on the SSH Client .....	79
Copy the Public Half of the User Encryption Key from the SSH Client to the SSH Server .....	81
Test the Connection from the SSH Client to the SSH Server .....	82
Using the Secure Data Transport .....	86
Improved Security .....	86
rsync over SSH .....	87
Copying a Single File with rsync .....	89
rsync over Slow WAN Connections .....	89
Scheduled rsync Snapshots .....	90
ipchains/iptables Firewall Rules for rsync and SSH .....	92
In Conclusion .....	93

## 5 CLONING SYSTEMS WITH SYSTEMIMAGER 95

SystemImager .....	95
Cloning the Golden Client with SystemImager .....	96
SystemImager Recipe .....	96
Install the SystemImager Server Software on the SystemImager Server .....	97
Using the Installer Program from SystemImager .....	98
Install the SystemImager Client Software on the Golden Client .....	98
Create a System Image of the Golden Client on the SystemImager Server .....	99
Make the Primary Data Server into a DHCP Server .....	101
Create a Boot Floppy for the Golden Client .....	103
Start rsync as a Daemon on the Primary Data Server .....	106
Install the Golden Client System Image on the New Clone .....	106
Post-installation Notes .....	107
Performing Maintenance: Updating Clients .....	108
SystemInstaller .....	108
System Configurator .....	109
In Conclusion .....	109

## 6 HEARTBEAT INTRODUCTION AND THEORY 111

The Physical Paths of the Heartbeats .....	112
Serial Cable Connection .....	113
Ethernet Cable Connection .....	113
Partitioned Clusters and STONITH .....	113
Heartbeat Control Messages .....	114
Heartbeats .....	114
Cluster Transition Messages .....	114
Retransmission Requests .....	115
Ethernet Heartbeat Control Messages .....	115
Security and Heartbeat Control Messages .....	115
How Client Computers Access Resources .....	115
Failover Using IP Address Takeover (IPAT) .....	116

Secondary IP Addresses and IP Aliases .....	116
Ethernet NIC Device Names .....	119
Secondary IP Address Names .....	119
IP Aliases .....	120
Offering Services .....	121
Gratuitous ARP (GARP) Broadcasts .....	122
Resource Scripts .....	125
Status of the Resource .....	125
Resource Ownership .....	126
Using init Scripts as Heartbeat Resource Scripts .....	128
Heartbeat Configuration Files .....	130
In Conclusion .....	130

## 7 A SAMPLE HEARTBEAT CONFIGURATION 131

Recipe .....	132
Preparations .....	132
Step 1: Install Heartbeat .....	132
Step 2: Configure /etc/ha.d/ha.cf .....	134
Step 3: Configure /etc/ha.d/haresources .....	136
Configure the haresources File .....	137
Step 4: Configure /etc/ha.d/authkeys .....	139
Step 5: Install Heartbeat on the Backup Server .....	139
Step 6: Set the System Time .....	140
Step 7: Launch Heartbeat .....	140
Launch Heartbeat on the Primary Server .....	140
Launch Heartbeat on the Backup Server .....	142
Examining the Log Files on the Primary Server .....	143
Stopping and Starting Heartbeat .....	144
Monitoring Resources .....	145
In Conclusion .....	145

## 8 HEARTBEAT RESOURCES AND MAINTENANCE 147

The Haresources File Syntax .....	147
Haresources File Syntax: Primary-Server Name .....	148
Haresources File Syntax: IP Alias .....	148
Heartbeat's Automated Network Interface Card Selection Process .....	149
Specifying a Network Interface Card .....	151
Customizing IP Address Takeover with the iptakeover Script .....	151
The Haresources File Syntax: Resources .....	152
Load Sharing with Heartbeat .....	154
Load Sharing with Heartbeat: Round-Robin DNS .....	155
Problems with Round Robin DNS Load Balancing .....	156
Wide-Area Load Balancing .....	157
Operator Alerts: Audible Alarm .....	157
Operator Alerts: Email Alerts .....	158
Heartbeat Maintenance .....	158

Changing Heartbeat Configuration Files .....	158
Server Maintenance and the Heartbeat <code>auto_fallback</code> Option .....	159
Forcing the Primary Server into Standby Mode .....	160
Tuning Heartbeat's Deadtime Value .....	161
Informational Messages in Heartbeat's Log .....	161
Failover and Respawn (Automatically Restarting Failed Resources) .....	161
License Manager Failover .....	162
In Conclusion .....	162

## **9 STONITH AND IPFAIL 163**

Stonith .....	164
An Unconventional Approach: Using a Single Stonith Device .....	165
Sample Heartbeat with Stonith Configuration .....	166
Stonith Sequence of Events .....	167
Stonith Devices .....	168
Viewing the Current List of Supported Stonith Devices .....	169
The Stonith Meatware "Device" .....	169
Using the Stonith Meatware Device with Heartbeat .....	170
Using a "Real" Stonith Device .....	173
Avoiding Multiple Stonith Events .....	174
Network Failures .....	175
ipfail .....	175
Watchdog and Softdog .....	176
Enable Watchdog in the Kernel .....	177
Kernel Panic—Hang or Reboot? .....	178
Configure Heartbeat to Support Watchdog .....	178
Testing Your Heartbeat Configuration .....	179
In Conclusion .....	181

## **PART III: CLUSTER THEORY AND PRACTICE**

<b>10 HOW TO BUILD A LINUX ENTERPRISE CLUSTER 185</b>	
Steps for Building a Linux Enterprise Cluster .....	186
NAS Server .....	186
Kernel Netfilter and Kernel Packet Routing .....	187
Cloning a Linux Machine .....	187
Cluster Naming Scheme .....	187
Applying System Configuration Changes to All Nodes .....	187
Building an LVS-NAT Cluster .....	187
Building an LVS-DR Cluster .....	188
Installing Software to Remove Failed Cluster Nodes .....	188
Installing Software to Monitor the Cluster Nodes .....	189
Monitoring the Performance of Cluster Nodes .....	189
Updating Software on Cluster Nodes and Servers .....	189

Centralizing User Account Administration .....	189
Installing a Printing System .....	190
Installing a Highly Available Batch Job-Scheduling System .....	190
Purchasing the Cluster Nodes .....	190
In Conclusion .....	192

## 11 THE LINUX VIRTUAL SERVER: INTRODUCTION AND THEORY 193

LVS IP Address Name Conventions .....	194
The Virtual IP (VIP) .....	194
The Real IP (RIP) .....	195
The Director's IP (DIP) .....	195
The Client Computer's IP (CIP) .....	195
IP Addresses in an LVS Cluster .....	195
Types of LVS Clusters .....	196
Network Address Translation (LVS-NAT) .....	196
Direct Routing (LVS-DR) .....	198
IP Tunneling (LVS-TUN) .....	200
LVS Scheduling Methods .....	202
Fixed (or Non-dynamic) Scheduling Methods .....	202
Dynamic Scheduling Methods .....	203
In Conclusion .....	208

## 12 THE LVS-NAT CLUSTER 209

How Client Computers Access LVS-NAT Cluster Resources .....	209
Virtual IP Addresses on LVS-NAT Real Servers .....	214
Building an LVS-NAT Web Cluster .....	216
Recipe for LVS-NAT .....	217
Step 1: Install the Operating System .....	217
Step 2: Configure and Start Apache on the Real Server .....	218
Step 3: Set the Default Route on the Real Server .....	219
Step 4: Install the LVS Software on the Director .....	220
Step 5: Configure LVS on the Director .....	221
Step 6: Test the Cluster Configuration .....	224
LocalNode: Using the Director as a Real Server .....	227
In Conclusion .....	229

## 13 THE LVS-DR CLUSTER 231

How Client Computers Access LVS-DR Cluster Services .....	232
ARP Broadcasts and the LVS-DR Cluster .....	236
Client Computers and ARP Broadcasts .....	236
In Conclusion .....	238

## 14 THE LOAD BALANCER

239

LVS and Netfilter .....	239
The Director's Connection Tracking Table .....	242
Hash Table Structure .....	242
Controlling the Hash Buckets .....	242
Viewing the Connection Tracking Table .....	243
Timeout Values for Connection Tracking Records .....	243
Return Packets and the Netfilter Hooks .....	244
LVS Without Persistence .....	246
LVS Persistence .....	247
Persistent Connection Template .....	247
Types of Persistent Connections .....	248
Persistent Client Connection (PCC) .....	249
Persistent Port Connection (PPC) .....	249
Port Affinity .....	250
Netfilter Marked Packets .....	250
In Conclusion .....	253

## 15 THE HIGH-AVAILABILITY CLUSTER

255

Redundant LVS Directors .....	255
High-Availability Cluster Design Goals .....	256
The High-Availability LVS-DR Cluster .....	257
Introduction to Idirectord .....	258
How Idirectord Monitors Cluster Nodes (LVS Real Servers) .....	258
LVS, Heartbeat, and Idirectord Recipe .....	261
Hide the Loopback Interface .....	261
Install the Heartbeat on a Primary and a Backup Director .....	263
Install Idirectord and Its Required Software Components .....	263
Install Idirectord .....	266
Test Your Idirectord Installation .....	266
Create the Idirectord Configuration File .....	267
Create the Health Check Web Page .....	272
Start Idirectord Manually and Test Your Configuration .....	273
Add Idirectord to the Heartbeat Configuration .....	274
Stateful Failover of the IPVS Table .....	275
Modifications to Allow Failover to a Real Server Inside the Cluster .....	277
In Conclusion .....	278

## 16 THE NETWORK FILE SYSTEM

279

Lock Arbitration .....	280
The Lock Arbitrator .....	280
The Existing Kernel Lock Arbitration Methods .....	281
The Network Lock Manager (NLM) .....	283
NLM and Kernel Lock Arbitration .....	284
NLM and Kernel BSD flock .....	284

NLM and Kernel System V lockf .....	285
NLM and Kernel Posix fcntl .....	285
NFS and File Lock (dotlock) Arbitration .....	286
Finding the Locks Held by the Linux Kernel .....	286
Performance Issues with NFS—Bottlenecks and Perceptions .....	288
Single Transactions and User Perception of NFS Performance .....	289
Multiple Transactions and User Perception of NFS Performance .....	290
Managing Lock and GETATTR Operations in a Cluster Environment .....	291
Managing Attribute Caching .....	291
Managing Interactive User Applications and Batch Jobs in a Cluster Environment .....	292
Run Batch Jobs Outside the Cluster .....	292
Use Multiple NAS Servers .....	292
Measuring NFS Latency .....	293
Measuring Total I/O Operations .....	294
Achieving the Best NAS Performance Possible .....	295
NFS Client Configuration Options .....	296
Putting It All Together .....	296
Developing NFS .....	297
Additional Starting Points for Information on Linux and NFS .....	297
In Conclusion .....	298

## PART IV: MAINTENANCE AND MONITORING

<b>17</b>		
<b>THE SIMPLE NETWORK MANAGEMENT PROTOCOL</b>		
<b>AND MON</b>		<b>301</b>
Mon .....	301	
Mon Alerts .....	302	
Mon Monitoring Scripts .....	302	
Where to Run Mon .....	302	
Basic Mon Recipe .....	303	
Step 1: Compile and Install the fping Package .....	304	
Step 2: Install the SNMP Package .....	304	
Step 3: Install the Required CPAN Modules for Mon .....	305	
Step 4: Install the Mon Software .....	306	
Step 5: Create the /etc/mon/mon.cf Configuration File .....	307	
Step 6: Test by Running the fping.monitor and mail.alert Scripts Manually .....	308	
Step 7: Create the Mon Log Directory and Mon Log File .....	309	
Step 8: Start the Mon Program in Debugging Mode and Test .....	309	
Mon and SNMP "Proof of Concept" Recipe .....	310	
Step 1: Install Net-SNMP Client Software on Each Cluster Node .....	310	
Step 2: Create the snmp.conf Configuration File .....	311	
Step 3: Start the SNMP Agent .....	311	
Step 4: View the SNMP MIB Locally .....	312	
Step 5: Install the SNMP Monitoring Agents .....	313	
Examine the Output of netsnmp-freespace.monitor .....	314	
Install Mon on All Cluster Nodes .....	314	

Mon and SNMP “Real-World” Recipe .....	315
Step 1: Create the snmpd.conf on All Cluster Nodes .....	315
Step 2: Install netsnmp-proc.monitor and the New mon.cf File on the Cluster Node Manager .....	317
Step 3: Install the Mon init Script .....	318
Step 4: Run SNMP and Mon .....	319
Email Alerts from Mon .....	319
Creating Your Own SNMP Script .....	320
A Sample Custom SNMP Script .....	320
Monitoring Your SNMP Script with Mon .....	321
Things to Monitor with SNMP Monitoring Scripts .....	322
Forcing a Stonith Event with Mon .....	323
Forcing a Heartbeat Failover with Mon .....	324
In Conclusion .....	325

## 18 GANGLIA 327

Introduction to Ganglia .....	328
gmond .....	328
gmetad .....	328
Installing Ganglia’s Prerequisite Packages .....	329
Install PHP .....	329
Install RRDtool .....	329
Install Apache on the Cluster Node Manager .....	330
Installing Ganglia on the Cluster Node Manager .....	330
Installing Ganglia on the Cluster Nodes .....	330
Configuring gmetad and gmond on the Cluster Node Manager .....	331
Modify /etc/gmetad.conf .....	331
Modify /etc/gmond.conf .....	331
Start gmond and gmetad .....	332
Add the Ganglia Page to Your Apache Configuration .....	332
The Ganglia Web Package .....	333
The Title Section .....	333
The Node Snapshot Section .....	334
Examining the Cluster Node from the Ganglia Web Package .....	336
Examining the Cluster Node from the Shell Prompt .....	337
gstat .....	338
Running a Command on the Least-Loaded Cluster Node Using gstat .....	338
Creating Custom Metrics with gmetric .....	338
In Conclusion .....	340

## 19 CASE STUDIES IN CLUSTER ADMINISTRATION 341

Administering Accounts Without Active Directory .....	342
Legacy Unix Account Administration Methods: The Problem .....	342
The Best of Both Worlds .....	343
Building a Reliable Cluster Account Authentication Mechanism .....	343
Using the Local passwd and Group File on Each Cluster Node .....	343

A Simple Script .....	344
Building a Fault-Tolerant Print Spooler .....	345
Cluster Nodes and Job Ordering .....	345
LPRng: A Linux Enterprise Cluster Printing System .....	346
Cluster Nodes and Print Jobs .....	346
Building the Cluster Printing System Based on LPRng .....	347
Install LPRng on the Central Print Server .....	347
Install LPRng on the Cluster Nodes .....	348
Modify the /etc/printcap.local File on the Cluster Nodes .....	348
Modify the /etc/printcap File on the Central Print Server .....	349
Managing Print Jobs on the Central Print Server .....	350
Managing Print Jobs from the Cluster Nodes .....	351
Rebooting Nodes for Preventative Maintenance .....	352
Using ipvsadm Commands to Remove a Cluster Node .....	352
Changing the Weight of a Cluster Node to 0 .....	352
Disabling Telnet Access to One of the Cluster Nodes .....	353
Sending and Receiving Email in a Cluster Environment .....	353
Creating a Batch Job-Scheduling System with No Single Point of Failure .....	354
Run ssh-keygen .....	355
Modify the sshd_config File on Each Cluster Node .....	355
Create the (RSA) known_hosts Entries on the Cluster Node Manager .....	355
The Batch Job Scheduler .....	356
In Conclusion .....	357

## **20** **THE LINUX CLUSTER ENVIRONMENT** **359**

The Linux Enterprise Cluster .....	360
Applications Running on the Cluster .....	361
The Cluster Node Manager .....	361
The Clients .....	362
The NAS Server .....	363
High-Availability NAS Server .....	364
Highly Available Serial Devices .....	364
Serial-to-IP Communication Device .....	365
High-Availability Modems .....	365
Highly Available Database Server .....	365
High-Availability SQL Server .....	366
Putting It All Together .....	367
The Cluster Environment .....	367
In Conclusion .....	368

## **A** **DOWNLOADING SOFTWARE FROM THE INTERNET** **(FROM A TEXT TERMINAL)** **369**

Using Lynx .....	371
Using Wget .....	372
What to Do with the File You Downloaded .....	372
tar Files .....	373

rpm Files .....	373
Creating Your Own Tape Archive (tar) File .....	374

**B** **TROUBLESHOOTING WITH THE TCPDUMP UTILITY** **375**

**C** **ADDING NETWORK INTERFACE CARDS TO YOUR SYSTEM** **379**

Monolithic Versus Modular Kernels .....	380
View Your Existing Configuration .....	380
Install the Card and Reboot .....	381
Run <code>linuxconf</code> .....	382
Testing the New NIC .....	383
Changes Made by <code>linuxconf</code> .....	384
Using the New NIC .....	384
Testing and Troubleshooting .....	384

**D** **STRATEGIES FOR DEPENDENCY FAILURES** **387**

What Are Dependencies? .....	387
Dynamic Executables and Shared Objects .....	388
rpm Packages and Shared Library Dependencies .....	390
Fixing a Dependency Failure Manually .....	391
Fixing a Dependency Failure Automatically .....	392
Using Yum to Install rpm Packages .....	392
Installing a New rpm Package with Yum .....	393
In Conclusion .....	393

**E** **OTHER POTENTIAL CLUSTER FILESYSTEMS AND LOCK ARBITRATION METHODS** **395**

**F** **LVS CLUSTERS AND THE APACHE CONFIGURATION FILE** **397**

ServerName .....	398
DocumentRoot .....	398
BindAddress .....	399
Port .....	399
Listen .....	399
Apache Virtual Host Configuration on Cluster Nodes .....	400
Apache IP-Based Virtual Hosts .....	400

Name-Based Virtual Hosts .....	401
Self-Referential (Redirection) URLs .....	403
IP-Based Virtual Hosts and Self-Referential URLs .....	404
Name-Based Virtual Hosts and Self-Referential URLs .....	404
Verify Your Virtual Host Configuration .....	404

## INDEX

**407**

## ACKNOWLEDGMENTS

A courageous management team made it possible for me to deploy the cluster described in this book into production—to my knowledge the first of its kind to be used to support an American brick-and-mortar business. I would especially like to thank Kevin Michel and Bill Dettmann for the faith they placed in the people who worked for them, Jim Gallops for standing up and defending GNU software when it counted the most, Ken Paradox for patiently correcting the many mistakes I made when I was trying to do what I said I was going to do, and Francis Hamilton for testing the LPRng printing system at the peril of his own efficiency.

Technical guardian angels from each of the projects helped to correct me when I strayed into inaccuracies. I would especially like to thank Alan Robertson (Heartbeat) for helping me when I misunderstood the syntax and structure of the Heartbeat package; Simon Hormann (The Linux Virtual Server, Heartbeat, ldirectord, and rpm dependencies) for going the extra mile (then two, then three; how many extra miles was it?); Matt Massie (Ganglia) for his gentle corrections and helpful ideas; Joseph Mack (The Linux Virtual Server) for his inspiration, advice, and support; Martin Pool (Open SSH and rsync) for putting up with my wayward ways; Charles Lever (NFS) for helping me to get over the really big hurdles; Brian Elliot Finley (SystemImager) for his terrific technical review; Peter Holzleitner (Mon and SNMP) for a truly selfless act; John Mark Walker (Linux utilities and the Linux boot process) for the grunt work; Brian Ward (The Linux Kernel) for saving me from my own folly; Dr. Patrick Powell (LPRng) for squashing every bug before they could do any harm; Mark Sherman (Linux novice) for lending a hand early on and being my kernel compile guinea pig; and Richard Stallman (Free Software Foundation) for clarifying the definition of the terms *Linux*, *GNU*, and *free*.

Also, a special thanks to all of the editors at No Starch press for catching the inaccurate, the obfuscated, and the unintelligible: William Pollock (editor and publisher, No Starch Press) for his uncompromising approach to every poorly formed idea; Susan Berge (production manager) for catching the pieces, keeping things on track, putting up with my eleventh hour scrambles, and then catching the pieces again; Karol Jurado (managing editor) for her graceful coordination of the cadre of technical reviewers; Riley Hoffman (compositor) for making the whole greater than the sum of the parts, redrawing every figure in the book, putting up with my criticisms of a job well done, and then doing it again; Patricia Witkin (communications manager), Leigh Poehler (rights manager/sales analyst), and Christina Samuell (administrative assistant) for smoothing out the bumps created by one delay after another, and then working hard to get the word out; Andy Carroll (editor) for a phenomenal job of cleaning things up and catching what so many others had already missed; and Bonnie Granat (editor) for seeing things all the way through to the end, and for her patience and understanding when she had to ask me and then ask me again.

On a personal note, I would like to thank my wife for letting me take time away from our relationship over the past four and a half years to sit alone at a computer, and my friends and family for helping me to stay on the path.

This book has been given so much support and attention that if any inaccuracies—technical or otherwise—remain they can certainly be attributed to me.

**Karl Kopper**  
**Sacramento, CA**

# INTRODUCTION



The term *cluster* is generally used to describe a broad range of distributed processing systems, but those who use the term in the computer industry have yet to agree upon a definition that has any weight to it. Gregory Pfister in his 1997 book, *In Search of Clusters*,<sup>1</sup> spends more than 500 pages grappling with this problem. He proposes the following elegant definition:

A cluster is a type of parallel or distributed system that:

- Consists of a collection of interconnected whole computers.
- Is used as a single unified computing resource.

Let me further clarify what I mean when I use the term *Linux Enterprise Cluster* by describing its properties and architecture.

---

<sup>1</sup> This book (and the artwork on its cover) inspired the Microsoft Wolfdpack product name.

## IEEE TASK FORCE ON CLUSTER COMPUTING

In 1999, the prestigious 380,000-member Institute of Electrical and Electronics Engineers, Inc. (IEEE) created the Task Force on Cluster Computing (TFCC). For details, see <http://www.ieeefcc.org>. In the “Cluster Computing White Paper” (2000), Thomas Sterling, writing on behalf of the TFCC (<http://arxiv.org/ftp/cs/papers/0004/0004014.pdf>), described a specific type of cluster called a *commodity cluster*. Sterling defined a commodity cluster as “a local computing system comprising a set of independent computers and a network interconnecting them.” He then described the implementation of a commodity cluster as follows:

A cluster is local in that all of its component subsystems are supervised within a single administrative domain, usually residing in a single room and managed as a single computer system. The constituent computer nodes are commercial-off-the-shelf (COTS), are capable of full independent operation as is, and are of a type ordinarily employed individually for standalone mainstream workloads and applications. The nodes may incorporate a single microprocessor or multiple microprocessors in a symmetric multiprocessor (SMP) configuration. The interconnection network employs COTS local area network (LAN) or systems area network (SAN) technology that may be a hierarchy of or multiple separate network structures. A cluster network is dedicated to the integration of the cluster compute nodes and is separate from the cluster’s external (worldly) environment. A cluster may be employed in many modes including but not limited to: high capability or sustained performance on a single problem, high capacity or throughput on a job or process workload, high availability through redundancy of nodes, or high bandwidth through multiplicity of disks and disk access or I/O channels.

A *Linux Enterprise Cluster* is a type of commodity cluster that typically runs mission-critical applications to support a community of users. Users of a Linux Enterprise Cluster do not need to sit in front of a Linux workstation; they may connect to the cluster using a web browser, telnet client, or any client application that knows how to communicate with the services running on the cluster nodes.

## Properties of a Linux Enterprise Cluster

If we succeed in building a system that can be used as a “single unified computing resource” using “a local computing system comprising a set of independent computers and a network interconnecting them,” then the user, the programmer, the program, and even the networking equipment and the other servers on the network will not know that they are using or communicating with a cluster.

Thus, the four basic properties of a Linux Enterprise Cluster are:

### **Users do not know that they are using a cluster**

If users do know they are using a cluster, they are using distinct, distributed servers and not a single *unified* computing resource.

### **Nodes within a cluster do not know they are part of a cluster**

In other words, the operating system does not need to be modified to run on a cluster node, and the failure of one node in the cluster has no effect on the other nodes inside the cluster. (Each cluster node is *whole* or complete—it can be rebooted or removed from the cluster without affecting the other nodes.)

A Linux Enterprise Cluster is a commodity cluster because it uses little or no specialty hardware and can use the normal Linux operating system. Besides lowering the cost of the cluster, this has the added benefit that the system administrator will not have to learn an entirely new set of skills to provide basic services required for normal operation, such as account authentication, host-name resolution, and email messaging.

### **Applications running in the cluster do not know that they are running inside a cluster**

If an application—especially a mission-critical legacy application—must be modified to run inside the cluster, then the application is no longer using the cluster as a single unified computing resource.

Some applications can be written using a cluster-aware application programming interface (API),<sup>2</sup> a Message Passing Interface (MPI),<sup>3</sup> or distributed objects. They will retain some, but not all, of the benefits of using the cluster as a single unified computing resource. But multiuser programs should not have to be rewritten<sup>4</sup> to run inside a cluster if the cluster is a single unified *computing* resource.

### **Other servers on the network do not know that they are servicing a cluster node**

The nodes within the Linux Enterprise Cluster must be able to make requests of servers on the network just like any other ordinary client computer. The servers on the network (DNS, email, user authentication, and so on) should not have to be rewritten to support the requests coming from the cluster nodes.

---

<sup>2</sup> Such as the Distributed Lock Manager discussed in Chapter 16.

<sup>3</sup> MPI is a library specification that allows programmers to develop applications that can share messages even when the applications sharing the messages are running on different nodes. A carefully written application can thus exploit multiple computer nodes to improve performance.

<sup>4</sup> This is assuming that they were already written for a multiuser operating system environment where multiple instances of the same application can run at the same time.

## Architecture of the Linux Enterprise Cluster

Let's use Pfister's statement that all clusters should act like "a single unified computing resource" to describe the architecture of an enterprise cluster. One example of a unified computing resource is a single computer, as shown in Figure 1.

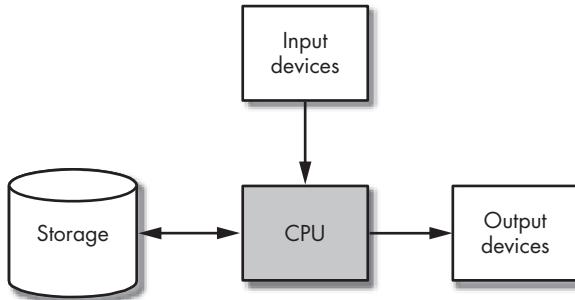


Figure 1: Simplified architecture of a single computer

If we replace the CPU in Figure 1 with "a collection of interconnected whole computers," the diagram could be redrawn as shown in Figure 2.

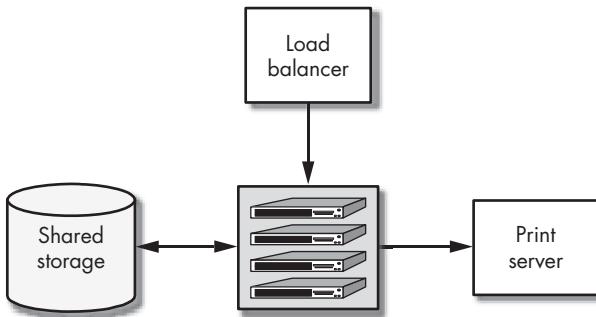


Figure 2: Simplified architecture of an enterprise cluster

The load balancer replaces the input devices. The shared storage device replaces a disk drive, and the print server is just one example of an output device. Let's briefly examine each of these before we describe how to build them.

### **The Load Balancer**

The load balancer sits between the users of the cluster and the "whole computers," which are the nodes that make up the cluster. The load balancer decides how best to distribute the incoming workload across all of the nodes.

In an enterprise cluster, user-based transactions make up the incoming workload. As users make connections to the cluster, such as HTTP web requests or telnet connections, they are assigned to cluster nodes based upon a load-balancing scheme (see Chapter 11).

## ***The Shared Storage Device***

The shared storage device acts as the single repository for the enterprise cluster's data, just as a disk drive does inside a single computer. Like a single disk drive in a computer, one of the most important features of this storage device is its ability to arbitrate access to the data so that two programs cannot modify the same piece of data at the same time. This feature is especially important in an enterprise cluster because two or more programs may be running on different cluster nodes at the same time. An enterprise cluster must therefore use a filesystem on all cluster nodes that can perform *lock arbitration*. That is, the filesystem must be able to protect the data stored on the shared storage device in order to prevent two applications from thinking they each have exclusive access to the same data at the same time (see Chapter 16).

## ***The Print Server***

The output device in this example is just one of many possible cluster output devices—a shared print server. A shared print server and other servers that offer services to all cluster nodes are located outside the cluster to arbitrate contention for things like printers, fax lines, and so on. Servers that arbitrate contention for output devices do not know they are servicing cluster nodes and can continue to use the classic client-server model for network communication. As we'll see in the next section, all of the server functions required to service the cluster nodes can be consolidated on to a single highly available server pair.

## ***The Cluster Node Manager***

Servers outside the cluster may do a variety of other things as well, such as:

- Act as license servers for applications running inside the cluster.
- Act as the central user account database.
- Provide other services such as DNS and email to all cluster nodes.
- Monitor the health of the cluster.

Again, the applications running on these servers outside the cluster should not need to be modified to support the nodes inside the cluster. In other words, the applications offering services to clients that are cluster nodes need not be cluster aware.

These functions can be combined on a high-availability server pair that we will call the *cluster node manager*. The cluster node manager holds the master user account database (and uses NIS or LDAP to distribute user accounts to all nodes<sup>5</sup>). The cluster node manager monitors the health of

---

<sup>5</sup> The Webmin and OSCAR packages both have methods of distributing user accounts to all cluster nodes that do not rely on NIS or LDAP, but these methods can still use the cluster node manager as the central user account database.

the cluster (using the Mon and Ganglia packages that will be discussed in Part 4); it is also a central print spooler (using LPRng). The cluster node manager can also offer a variety of other services to the cluster using the classic client-server model, such as the ability to send faxes.<sup>6</sup>

## No Single Point of Failure

We can further describe the architecture of the enterprise cluster by discussing a basic requirement of any mission-critical system: it must have no single point of failure.

An enterprise cluster should always have the following characteristic: *Any computer within the cluster, or any computer the cluster depends upon for normal operation, can be rebooted without rebooting the entire cluster.*

One way to be able to reboot servers the cluster depends upon without affecting the cluster is to build high-availability server pairs for all of the servers the cluster depends upon for normal operation. Our simplified cluster drawing could be redrawn as shown in Figure 3.

This figure shows two load balancers, two print servers, and two shared storage devices servicing four cluster nodes.

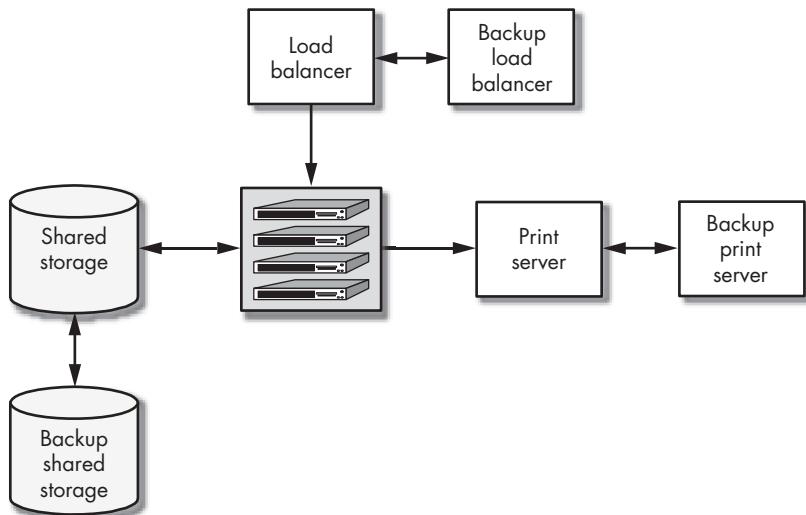


Figure 3: Enterprise cluster with no single point of failure

In Part II of this book, we will learn how to build high-availability server pairs using the Heartbeat package. Part III describes how to build a highly available load balancer and a highly available cluster node manager. (Recall from the previous discussion that the cluster node manager can be a print server for the cluster.)

---

<sup>6</sup> A brief discussion of Hyrax is included near the end of Chapter 17.

**NOTE** *Of course, one of the cluster nodes may also go down unexpectedly or may no longer be able to perform its job. If this happens, the load balancer should be smart enough to remove the failed cluster node from the cluster and alert the system administrator. The cluster nodes should be independent of each other, so that aside from needing to do more work, they will not be affected by the failure of a node.*

## In Conclusion

I've introduced a definition of the term *cluster* that I will use throughout this book as a logical model for understanding how to build one. The definition I've provided in this introduction (the four basic properties of a Linux Enterprise Cluster) is the foundation upon which the ideas in this book are built. After I've introduced the methods for implementing these ideas, I'll conclude the book with a final chapter that provides an introduction to the physical model that implements these ideas. You can skip ahead now and read this final chapter if you want to learn about this physical model, or, if you just one to get a cluster up and running you can start with the instructions in Part II. If you don't have any experience with Linux you will probably want to start with Part I, and read about the basic components of the GNU/Linux operating system that make building a Linux Enterprise Cluster possible.



## PRIMER



This book could have been called *The GNU/Linux Enterprise Cluster*. Here is what Richard Stallman, founder of the GNU Project, has to say on this topic:

Mostly, when people speak of “Linux clusters” they mean one in which the GNU/Linux system is running. They think the whole system is Linux, so they call it a “Linux cluster.” However, the right term for this would be “GNU/Linux cluster.”

The accurate meaning for “Linux cluster” would be one in which the kernel is Linux. The identity of the kernel is relevant for some technical development and system administration issues. However, what users and user programs see is not the kernel, but rather the rest of the system. The only reason that Linux clusters are similar from the user’s point of view is that they are really all GNU/Linux clusters.

A “free software cluster” (I think that term is clearer than “free cluster”) would be one on which only free software is running. A GNU/Linux cluster can be a free software cluster, but isn’t necessarily one.

The basic GNU/Linux system is free software; we launched the development of the GNU system in 1984 specifically to have a free software operating system. However, most of the distributions of GNU/Linux add some non-free software to the system. (This software usually is not open-source either.) The distributors describe these programs as a sort of bonus, and since most of the users have never encountered any other way to look at the matter, they usually go along with that view. As a result, many of the machines that run GNU/Linux (whether clusters or not) have non-free software installed as well. Thus, we fail to achieve our goal of giving computer users freedom.

Before I describe how to build a GNU/Linux Enterprise Cluster, let me define a few basic terms that I’ll use throughout the book to describe one.

## High Availability Terminology

I’ll start with the terminology used to describe high availability. I’m not going to split hairs at this point—I’ll do that later, when I explore each concept in depth. The important thing here is that I introduce a vocabulary that allows us to talk about multiple computers working together to accomplish a task.

When a program runs, it is called a *process*. A process running on a Linux system is called a *daemon*. A daemon and the effects it produces is called a *service*. A service is called a *resource* when it is combined with its operating environment (configuration files, data, network mechanism used to access it, and so forth). A *failover* occurs when a resource moves from one computer to another. A proper failover or *high-availability* configuration has no single point of failure.<sup>1</sup>

## Linux Enterprise Cluster Terminology

Now that I’ve introduced the terminology used to describe high availability, I’d like to introduce a few terms that allow us to describe a GNU/Linux Enterprise Cluster.

---

<sup>1</sup> I’m departing from the Red Hat cluster terminology here, but I am using these terms as they are defined by Linux-ha advocates (see <http://www.linux-ha.org>).

In a cluster, all computers (called *nodes*) offer the same services. The cluster *load balancer* intercepts all incoming requests for services.<sup>2</sup> It then distributes the requests across all of the cluster nodes as evenly as possible. A *high-availability cluster* is able to failover the *cluster load balancing resource* from one computer to another.

This book describes how to build a *high-availability, load-balancing cluster* using the GNU/Linux operating system and several free software packages. All of the recipes in this book describe how to build configurations that have been used by the author in production. (Some projects and recipes considered for use in this book did not make it into the final version because they proved too unreliable or too immature for use in production.)

---

<sup>2</sup> To be technically correct here, I need to point out that you can build a cluster without one machine acting as a load balancer for the cluster—see <http://www.vitaesoft.com> for one method of accomplishing this.



# PART I

## CLUSTER RESOURCES





# 1

## STARTING SERVICES



In this chapter we will examine the basic boot procedure used on a Linux server and how to disable or enable a service. This chapter includes a list of services normally found on a Linux server and describes how these services are used on the nodes inside a cluster.

### How Do Cluster Services Get Started?

In a cluster environment, a daemon or service is started:

- By the `init` daemon when the system boots or enters a runlevel.
- By Heartbeat when it first starts or needs to take ownership of a resource.
- By `xinetd` when a network request comes in.
- Using some other non-standard method (such as the Daemontools<sup>1</sup> package).

---

<sup>1</sup> The Daemontools package is described in Chapter 8.

**NOTE** For the moment, we are leaving batch jobs out of this discussion (because batch jobs are normally not considered “cluster services” that are offered to client computers). To learn more about batch jobs in a cluster environment, see Chapter 18.

Which of these methods should be used depends on the answers to the following questions:

1. Should the service run at all times or only when network requests come in?
2. Should the service failover to a backup server when the primary server crashes?
3. Should the service be restarted automatically if it stops running or ends abnormally?

If the service needs to run only as a result of an incoming network request (telnet and pop are good examples of this type of service), then it should be started by xinetd. However, the majority of services running on a Linux server, and on the nodes inside a cluster, will run all the time and should therefore be started by init when the system first boots (hence the name “init”).

If a service should continue to run even when the system it is running on crashes (the LPRng lpd printing daemon running on a print server, for example), then the Heartbeat program should start the service (described in Part II of this book).

In this chapter I’ll focus on how init works and later, in Part II, I’ll describe how to start services using Heartbeat. Experienced Linux and Unix system administrators will find the first part of this chapter to be a repeat of concepts they are already familiar with and may want to skip to the section titled “Using the Red Hat init Scripts on Cluster Nodes.”

## Starting Services with init

Once your boot loader has finished the initial steps to get your kernel up and running, the init daemon—the parent of all processes—is started. Then init launches the daemons necessary for normal system operation based on entries in its /etc/inittab configuration file.

### *The /etc/inittab File*

The heart of the /etc/inittab configuration file is the following seven lines that define seven system runlevels:

---

```
10:0:wait:/etc/rc.d/rc 0
11:1:wait:/etc/rc.d/rc 1
12:2:wait:/etc/rc.d/rc 2
13:3:wait:/etc/rc.d/rc 3
```

---

```
14:4:wait:/etc/rc.d/rc 4
15:5:wait:/etc/rc.d/rc 5
16:6:wait:/etc/rc.d/rc 6
```

---

**NOTE** *The system can be at only one runlevel at a time.*

These lines indicate that the `rc` program (the `/etc/rc.d/rc` script) should run each time the runlevel of the system changes and that `init` should pass a single character argument consisting of the runlevel number (0 through 6) to the `rc` program. The `rc` program then launches all of the scripts that start with the letter `S` in the corresponding runlevel subdirectory underneath the `/etc/rc.d` directory.

The default runlevel, the runlevel for normal server operation, is defined in the `/etc/inittab` file on the `initdefault` line:

---

```
id:3:initdefault:
```

---

**NOTE** *Use the `runlevel` command to see the current runlevel.*

The default runlevel for a Red Hat Linux server that is not running a graphical user interface is runlevel 3. This means the `init` daemon runs the `rc` program and passes it an argument of 3 when the system boots. The `rc` program then runs each script that starts with the letter `S` in the `/etc/rc.d/rc3.d`, passing them the argument `start`.

Later, when the system administrator issues the `shutdown` command, the `init` daemon runs the `rc` program again, but this time the `rc` program runs all of the scripts in the `/etc/rc.d/rc3.d` directory that start with the letter `K` (for “kill”) and passes them the argument `stop`.

On Red Hat, you can use the following command to list the contents of all of the runlevel subdirectories:

---

```
#ls -l /etc/rc.d/rc?.d | less
```

---

**NOTE** *The question mark in this command says to allow any character to match. So this command will match on each of the subdirectories for each of the runlevels: `rc0.d`, `rc1.d`, `rc2.d`, and so forth.*

The `start` and `kill` scripts stored in these subdirectories are, however, not real files. They are just symbolic links that point to the real script files stored in the `/etc/rc.d/init.d` directory. All of the `start` and `stop` `init` scripts are, therefore, stored in one directory (`/etc/rc.d/init.d`), and you control which services run at each runlevel by creating or removing a symbolic link within one of the runlevel subdirectories.

Fortunately, software tools are available to help manage these symbolic links. We will explain how to use these tools in a moment, but first, let’s examine one way to automatically restart a daemon when it dies by using the `respawn` option in the `/etc/inittab` file.

## Respawning Services with init

You can cause the operating system to automatically restart a daemon when it dies by placing the name of the executable program file in the `/etc/inittab` file and adding the `respawn` option. `init` will start the daemon as the system enters the runlevel and then watch to make sure the daemon stays running (restarting the daemon, if it dies), as long as the system remains at the same runlevel.

A sample `/etc/inittab` entry using the `respawn` option looks like this:

---

```
sn:2345:respawn:/usr/local/scripts/start_snmpd > /dev/null
```

---

This entry tells the `init` daemon to run the script `/usr/local/scripts/start_snmpd` at runlevels 2, 3, 4, and 5, and to send any output produced to the `/dev/null` “device” (the output is discarded).

Notice, however, that we have entered a script file here (`/usr/local/scripts/start_snmpd`) and not the actual `snmpd` executable. How will `init` know which running process to monitor if this script simply runs the `snmpd` daemon and then ends? `init` will assume the program has died each time the script `/usr/local/scripts/start_snmpd` finishes running, and it will dutifully run the script again.

**NOTE** *When this happens, you are likely to see an error message from init saying the sn identifier (referring to the first field in the /etc/inittab file) is respawning too fast and init has decided to give up trying for a few minutes.*

So to make sure the `respawn` option works correctly, we need to make sure the script used to start the `snmpd` daemon *replaces* itself with the `snmpd` daemon. This is easy to do with the bash programming statement `exec`. So the simple bash script `/usr/local/scripts/start_snmpd` looks like this:

---

```
#!/bin/bash
exec /usr/sbin/snmpd -s -P /var/run/snmpd -l /dev/null
```

---

The first line of this script starts the bash shell, and the second line causes whatever process identification number, or PID, is associated with this bash shell to be replaced with the `snmpd` daemon (the bash shell disappears when this happens). This makes it possible for `init` to run the script, find out what PID it is assigned, and then monitor this PID. If `init` sees the PID go away (meaning the `snmpd` daemon died), it will run the script again and monitor the new PID thanks to the `respawn` entry in the `/etc/inittab` file.<sup>2</sup>

## Managing the init Script Symbolic Links with chkconfig

The `chkconfig` program creates and removes the symbolic links used by `init`'s servant, the `/etc/rc.d/rc` script. These symbolic links are the real way to enable

---

<sup>2</sup> Note that the `snmpd` daemon also supports a `-f` argument that accomplishes the same thing as the `exec` statement in this bash script. See the `snmpd` man page for more information.

or prevent a service from automatically running at a particular runlevel. As previously noted, all of the symbolic links for all system runlevels point back to the real scripts stored in the `/etc/rc.d/init.d` directory (actually, according to the Linux Standards Base,<sup>3</sup> this directory is the `/etc/init.d` directory, but on Red Hat, these two directories point to the same files<sup>4</sup>).

If you want a service (called `myservice`) to run at runlevels 2, 3, and 4, this means you want `chkconfig` to create three symbolic links in each of the following runlevel subdirectories:

---

```
/etc/rc.d/rc2.d
/etc/rc.d/rc3.d
/etc/rc.d/rc4.d
```

---

These symbolic links should start with the letter `S` to indicate you want to start your script at each of these runlevels. `chkconfig` should then create symbolic links that start with the letter `K` (for “kill”) in each of the remaining `/etc/rc.d/rc<runlevel>.d` directories (for runlevels 0, 1, 5, and 6).

To tell `chkconfig` you want it to do this for your script, you would add the following lines to the script file in the `/etc/rc.d/init.d` directory:

---

```
# chkconfig: 234 99 90
# description: myscript runs my daemon at runlevel 2, 3, or 4
```

---

You can then simply run the `chkconfig` program, which interprets this line to mean “create the symbolic links necessary to run this script at runlevel 2, 3, and 4 with the `start` option, and create the symbolic links needed to run this script with the `kill` option at all other runlevels.”

The numbers 99 and 90 in this example are the start and stop priority of this script. When a symbolic link is created in one of the runlevel subdirectories, it is also given a priority number used by the `rc` script to determine the order it will use to launch (or kill) the scripts it finds in each of the runlevel subdirectories. A lower priority number after the `S` in the symbolic link name means the script should run before other scripts in the directory. For example, the script to bring up networking, `/etc/rc.d/rc3.d/S10network`, runs before the script to start the Heartbeat program `/etc/rc.d/rc3.d/S34heartbeat`.

The second commented line in this example contains a (required) description for this script. The `chkconfig` commands to first remove any old symbolic links that represent an old `chkconfig` entry and then add this script (it was saved in the file `/etc/rc.d/init.d/myscript`) are:

---

```
chkconfig --del myscript
chkconfig --add myscript
```

---

<sup>3</sup> See <http://www.linuxbase.org/spec> for the latest revision to the LSB.

<sup>4</sup> Because `/etc/init.d` is a symbolic link to `/etc/rc.d/init.d` on current Red Hat distributions.

**NOTE** Always use the `--del` option before using the `--add` option when using `chkconfig` to ensure you have removed any old links.

The `chkconfig` program will then create the following symbolic links to your file:

---

```
/etc/rc.d/rc0.d/K90myscript
/etc/rc.d/rc1.d/K90myscript
/etc/rc.d/rc2.d/S99myscript
/etc/rc.d/rc3.d/S99myscript
/etc/rc.d/rc4.d/S99myscript
/etc/rc.d/rc5.d/K90myscript
/etc/rc.d/rc6.d/K90myscript
```

---

Again, these symbolic links just point back to the real file located in the `/etc/init.d` directory (from this point forward, I will use the LSB directory name `/etc/init.d` instead of the `/etc/rc.d/init.d` directory originally used on Red Hat systems) throughout this book. If you need to modify the script to start or stop your service, you need only modify it in one place: the `/etc/init.d` directory.

You can see all of these symbolic links (after you run the `chkconfig` command to add them) with the following single command:

---

```
#find /etc -name '*myscript' -print
```

---

You can also see a report of all scripts with the command:

---

```
#chkconfig --list
```

---

### ***Managing the init Script Symbolic Links with ntsysv***

The `ntsysv` command relies on these same commented `chkconfig` lines in each of the scripts in the `/etc/rc.d/init.d` directory to manage the symbolic links to start and stop scripts at each runlevel.

Normally, `ntsysv` only manages the currently running level of the system (not very useful), so you will probably want to tell it to manage all of the runlevels by entering the command:

---

```
#ntsysv --level 0123456
```

---

This will bring up a list of all of the scripts in the `/etc/rc.d/init.d` directory. By adding an asterisk next to the script name in this listing (and selecting OK), you tell the `ntsysv` program to create all of the symbolic links just as if you had run the `chkconfig --add <scriptname>` command. If you remove the asterisk next to the script name, the symbolic links are removed just as if you had entered the `chkconfig --del <scriptname>` command.

**NOTE** *chkconfig* and *ntsysv* commands do not start or stop daemons (neither command actually runs the scripts). They only create or remove the symbolic links that affect the system the next time it enters or leaves a runlevel (or is rebooted). To start or stop scripts, you can enter the command *service <script-name> start* or *service <script-name> stop*, where *<script-name>* is the name of the file in the /etc/init.d directory.

## **Removing Services You Do Not Need**

For each script you do not need (see the next section for a description), you can disable or remove the services by doing the following:

Kill the currently running daemon (you can skip this step and simply reboot the system when you are finished):

---

```
#/etc/init.d/<script-name> stop
```

---

Or on Red Hat systems, the following command works just as well:

---

```
#service <script-name> stop
```

---

Remove symbolic links to the script that cause it to start or stop at the predefined *chkconfig* runlevels with the command:

---

```
#chkconfig --del <script-name>
```

---

or

---

```
#ntsysv --level 0123456
```

---

Then, inside of the *ntsysv* utility, remove the \* next to each script name to cause it not to run at boot time.

## **Using the Red Hat init Scripts on Cluster Nodes**

This section contains a description of the init scripts used on a normal Red Hat system along with a discussion of how the services started by the init script are used, or can be used, in a Linux Enterprise Cluster. Note that you may not have all of these scripts on your system due to a difference in the Red Hat release you are using or because you do not have the service installed.

### **aep1000**

Supports the Accelerated Encryption Processing card to speed Secure Socket Layer (SSL) encryption. Not used to build the cluster in this book.

### **anacron**

Used like *cron* and *at* to start commands at a particular time, even on systems that are not running continuously. On Red Hat, the /etc/anacrontab file lists the normal log rotation cron job that is executed daily. If you

disable this, you will no longer perform log rotation, and your system disk will eventually fill up. In Chapters 18 and 19, we will describe two methods of running cron jobs on all cluster nodes without running the `cron` daemon on all nodes (the `cron` daemon runs on a single node and remotely initiates or starts the `anacron` program once a day on each cluster node).

#### **apmd**

Used for advanced power management. You only need this if you have an uninterruptible power supply (UPS) system connected to your system and want it to automatically shut down in the event of a power failure before the battery in your UPS system runs out. In Chapter 9, we'll see how the Heartbeat program can also control the power supply to a device through the use of a technology called STONITH (for "shoot the other node in the head").

#### **arpwatch**

Used to keep track of IP address-to-MAC address pairings. Normally, you do not need this daemon. As a side note, the Linux Virtual Server Direct Routing clusters (as described in Chapter 13) must contend with potential Address Resolution Protocol (ARP) problems introduced through the use of the cluster load-balancing technology, but `arpwatch` does not help with this problem and is normally not used on cluster nodes.

#### **atd**

Used like `anacron` and `cron` to schedule jobs for a particular time with the `at` command. This method of scheduling jobs is used infrequently, if at all. In Chapter 18, we'll describe how to build a no-single-point-of-failure batch scheduling mechanism for the cluster using Heartbeat, the `cron` daemon, and the `clustersh` script.

#### **autofs**

Used to automatically mount NFS directories from an NFS server. This script only needs to be enabled on an NFS client, and only if you want to automate mounting and unmounting NFS drives. In the cluster configuration described in this book, you will not need to mount NFS file systems on the fly and will therefore not need to use this service (though using it gives you a powerful and flexible means of mounting several NFS mount points from each cluster node on an as-needed basis.) If possible, avoid the complexity of the `autofs` mounting scheme and use only one NFS-mounted directory on your cluster nodes.

#### **bcm5820**

Support for the Broadcom Cryponet BCM5820 chip for speeding SSL communication. Not used in this book.

**crond**

Used like `anacron` and `atd` to schedule jobs. On a Red Hat system, the `crond` daemon starts `anacron` once a day. In Chapter 18, we'll see how you can build a no-single-point-of-failure batch job scheduling system using `cron` and the open source `Ganglia` package (or the `clustersh` script provided on the CD-ROM included with this book). To control cron job execution in the cluster, you may not want to start the `crond` daemon on all nodes. Instead, you may want to only run the `crond` daemon on one node and make it a high-availability service using the techniques described in Part II of this book. (On cluster nodes described in this book, the `crond` daemon is not run at system start up by `init`. The Heartbeat program launches the `crond` daemon based on an entry in the `/etc/ha.d/haresources` file. Cron jobs can still run on all cluster nodes through remote shell capabilities provided by SSH.)

**cups**

The common Unix printing system. In this book, we will use `LPRng` instead of `cups` as the printing system for the cluster. See the description of the `lpd` script later in this chapter.

**firstboot**

Used as part of the system configuration process the first time the system boots.

**functions**

Contains library routines used by the other scripts in the `/etc/init.d` directory. Do not modify this script, or you will risk corrupting all of the `init` scripts on the system.

**gpm**

Makes it possible to use a mouse to cut and paste at the text-based console. You should not disable this service. Incidentally, cluster nodes will normally be connected to a KVM (keyboard video mouse) device that allows several servers to share one keyboard and one mouse. Due to the cost of these KVM devices, you may want to build your cluster nodes without connecting them to a KVM device (so that they boot without a keyboard and mouse connected).<sup>5</sup>

**halt**

Used to shut the system down cleanly.

**httpd**

Used to start the apache daemon(s) for a web server. You will probably want to use `httpd` even if you are not building a web cluster to help the cluster load balancer (the `ldirectord` program) decide when a node should be removed from the cluster. This is the concept of monitoring a parallel service discussed in Chapter 15.

---

<sup>5</sup> Most server hardware now supports this.

**identd**

Used to help identify who is remotely connecting to your system. The theory sounds good, but you probably will never be saved from an attack by `identd`. In fact, hackers will attack the `identd` daemon on your system. Servers inside an LVS-NAT cluster also have problems sending `identd` requests back to client computers. Disable `identd` on your cluster nodes if possible.

**ipchains or iptables**

The ability to manipulate packets with the Linux kernel is provided by the `ipchains` (for kernels 2.2 and earlier) or `iptables` (for kernels 2.4 and later). This script on a Red Hat system runs the `iptables` or `ipchains` commands you have entered previously (and saved into a file in the `/etc/sysconfig` directory). For more information, see Chapter 2.

**irda**

Used for wireless infrared communication. Not used to build the cluster described in this book.

**isdn**

Used for Integrated Services Digital Networks communication. Not used in this book.

**kdcrotate**

Used to administer Kerberos security on the system. See Chapter 19 for a discussion of methods for distributing user account information.

**keytable**

Used to load the keyboard table.

**killall**

Used to help shut down the system.

**kudzu**

Probes your system for new hardware at boot time—very useful when you change the hardware configuration on your system, but increases the amount of time required to boot a cluster node, especially when it is disconnected from the KVM device.

**lisa**

The LAN Information Server service. Normally, users on a Linux machine must know the name or address of a remote host before they can connect to it and exchange information with it. Using `lisa`, a user can perform a network discovery of other hosts on the network similar to the way Windows clients use the Network Neighborhood. This technology is not used in this book, and in fact, may confuse users if you are building the cluster described in this book (because the LVS-DR cluster nodes will be discovered by this software).

## 1pd

This is the script that starts the LPRng printing system. If you do not see the 1pd script, you need to install the LPRng printing package. The latest copy of LPRng can be found at <http://www.lprng.com>. On a standard Red Hat installation, the LPRng printing system will use the /etc/printcap.local file to create an /etc/printcap file containing the list of printers. Cluster nodes (as described in this book) should run the 1pd daemon. Cluster nodes will first spool print jobs locally to their local hard drives and then try (essentially, forever) to send the print jobs to a central print spooler that is also running the 1pd daemon. See Chapter 19 for a discussion of the LPRng printing system.

## netfs

Cluster nodes, as described in this book, will need to connect to a central file server (or NAS device) for lock arbitration. See Chapter 16 for a more detailed discussion of NFS. Cluster nodes will normally need this script to run at boot time to gain access to data files that are shared with the other cluster nodes.

## network

Required to bring up the Ethernet interfaces and connect your system to the cluster network and NAS device. The Red Hat network configuration files are stored in the /etc/sysconfig directory. In Chapter 5, we will describe the files you need to look at to ensure a proper network configuration for each cluster node after you have finished cloning. This script uses these configuration files at boot time to configure your network interface cards and the network routing table on each cluster node.<sup>6</sup>

## nfs

Cluster nodes will normally not act as NFS servers and will therefore not need to run this script.<sup>7</sup>

## nfslock

The Linux kernel will start the proper in-kernel NFS locking mechanism (rpc.lockd is a kernel thread called *klockd*) and rpc.statd to ensure proper NFS file locking. However, cluster nodes that are NFS clients can run this script at boot time without harming anything (the kernel will always run the lock daemon when it needs it, whether or not this script was run at boot time). See Chapter 16 for more information about NFS lock arbitration.

## nscd

Helps to speed name service lookups (for host names, for example) by caching this information. To build the cluster described in this book, using nscd is not required.

---

<sup>6</sup> In the LVS-DR cluster described in Chapter 13, we will modify the routing table on each node to provide a special mechanism for load balancing incoming requests for cluster resources.

<sup>7</sup> An exception to this might be very large clusters that use a few cluster nodes to mount filesystems and then re-export these filesystems to other cluster nodes. Building this type of large cluster (probably for scientific applications) is beyond the scope of this book.

**ntpd**

This script starts the Network Time Protocol daemon. When you configure the name or IP address of a Network Time Protocol server in the `/etc/ntp.conf` file, you can run this service on all cluster nodes to keep their clocks synchronized. As the clock on the system drifts out of alignment, cluster nodes begin to disagree on the time, but running the `ntpd` service will help to prevent this problem. The clock on the system must be reasonably accurate before the `ntpd` daemon can begin to *slew*, or adjust, it and keep it synchronized with the time found on the NTP server. When using a distributed filesystem such as NFS, the time stored on the NFS server and the time kept on each cluster node should be kept synchronized. (See Chapter 16 for more information.) Also note that the value of the hardware clock and the system time may disagree. To set the hardware clock on your system, use the `hwclock` command. (Problems can also occur if the hardware clock and the system time diverge too much.)

**pcmcia**

Used to recognize and configure `pcmcia` devices that are normally only used on laptops (not used in this book).

**portmap**

Used by NFS and NIS to manage RPC connections. Required for normal operation of the `nfslocking` mechanism and covered in detail in Chapter 16.

**pppoe**

For Asymmetric Digital Subscriber Line (ADSL) connections. If you are not using ADSL, disable this script.

**pxe**

Some diskless clusters (or diskless workstations) will boot using the PXE protocol to locate and run an operating system (PXE stands for Preboot eXecution Environment) based on the information provided by a PXE server. Running this service will enable your Linux machine to act as a PXE server; however, building a cluster of diskless nodes is outside the scope of this book.<sup>8</sup>

**random**

Helps your system generate better random numbers that are required for many encryption routines. The secure shell (SSH) method of encrypting data in a cluster environment is described in Chapter 4.

**rawdevices**

Used by the kernel for device management.

---

<sup>8</sup> See Chapter 10 for the reasons why. A more common application of the PXE service is to use this protocol when building Linux Terminal Server Project or LTSP Thin Clients.

**rhnsrc**

Connects to the Red Hat server to see if new software is available. How you want to upgrade software on your cluster nodes will dictate whether or not you use this script. Some system administrators shudder at the thought of automated software upgrades on production servers. Using a cluster, you have the advantage of upgrading the software on a single cluster node, testing it, and then deciding if all cluster nodes should be upgraded. If the upgrade on this single node (called a *Golden Client* in SystemImager terminology) is successful, it can be copied to all of the cluster nodes using the cloning process described in Chapter 5. (See the `updateclient` command in this chapter for how to update a node after it has been put into production.)

**rstatd**

Starts `rpc.rstatd`, which allows remote users to use the `rup` command to monitor system activity. System monitoring as described in this book will use the Ganglia package and Mon software monitoring packages that do not require this service.

**rusersd**

Lets other people see who is logged on to this machine. May be useful in a cluster environment, but will not be described in this book.

**rwall**

Lets remote users display messages on this machine with the `rwall` command. This may or may not be a useful feature in your cluster. Both this daemon and the next should only be started on systems running behind a firewall on a trusted network.

**rwshod**

Lets remote users see who is logged on.

**saslauthd**

Used to provide Simple Authentication and Security Layer (SASL) authentication (normally used with protocols like SMTP and POP). Building services that use SASL authentication is outside the scope of this book. To build the cluster described in this book, this service is not required.

**sendmail**

Will you allow remote users to send email messages directly to cluster nodes? For example, an email order-processing system requires each cluster node to receive the email order in order to balance the load of incoming email orders. If so, you will need to enable (and configure) this service on all cluster nodes. `sendmail` configuration for cluster nodes is discussed in Chapter 19.

**single**

Used to administer runlevels (by the `init` process).

**smb**

Provides a means of offering services such as file sharing and printer sharing to Windows clients using the package called Samba. Configuring a cluster to support PC clients in this manner is outside the scope of this book.

**snmpd**

Used for Simple Network Management Protocol (SNMP) administration. This daemon will be used in conjunction with the Mon monitoring package in Chapter 17 to monitor the health of the cluster nodes. You will almost certainly want to use SNMP on your cluster. If you are building a public web server, you may want to disable this daemon for security reasons (when you run `snmpd` on a server, you add an additional way for a hacker to try to break into your system). In practice, you may find that placing this script as a respawn service in the `/etc/inittab` file provides greater reliability than simply starting the daemon once at system boot time. (See the example earlier in this chapter—if you use this method, you will need to disable this init script so that the system won’t try to launch `snmpd` twice.)

**snmptrapd**

SNMP can be configured on almost all network-connected devices to send *traps* or alerts to an SNMP Trap host. The SNMP Trap host runs monitoring software that logs these traps and then provides some mechanism to alert a human being to the fact that something has gone wrong. Running this daemon will turn your Linux server into an SNMP Trap server, which may be desirable for a system sitting outside the cluster, such as the cluster node manager.<sup>9</sup> A limitation of SNMP Traps, however, is the fact that they may be trying to indicate a serious problem with a single trap, or alert, message and this message may get lost. (The SNMP Trap server may miss its one and only opportunity to hear the alert.) The approach taken in this book is to use a server sitting outside the cluster (running the Mon software package) to poll the SNMP information stored on each cluster node and raise an alert when a threshold is violated or when the node does not respond. (See Chapter 17.)

**squid**

Provides a proxy server for caching web requests (among other things). `squid` is not used in the cluster described in this book.

**sshd**

Open SSH daemon. We will use this service to synchronize the files on the servers inside the cluster and for some of the system cloning recipes in this book (though using SSH for system cloning is not required).

---

<sup>9</sup> See Part IV of this book for more information.

### **syslog**

The `syslog` daemon logs error messages from running daemons and programs based on configuration entries in the `/etc/syslog.conf` file. The log files are kept from growing indefinitely by the `anacron` daemon, which is started by an entry in the `crontab` file (by `cron`). See the `logrotate` man page. Note that you can cause all cluster nodes to send their `syslog` entries to a single host by creating configuration entries in the `/etc/syslog.conf` file using the `@hostname` syntax. (See the `syslog` man page for examples.) This method of error logging may, however, cause the entire cluster to slow down if the network becomes overloaded with error messages. In this book, we will use the default `syslog` method of sending error log entries to a locally attached disk drive to avoid this potential problem. (We'll proactively monitor for serious problems on the cluster using the `Mon` monitoring package in Part IV of this book.)<sup>10</sup>

### **tux**

Instead of using the Apache `httpd` daemon, you may choose to run the `Tux` web server. This web server attempts to introduce performance improvements over the Apache web server daemon. This book will only describe how to install and configure Apache.

### **winbindd**

Provides a means of authenticating users using the accounts found on a Windows server. This method of authentication is not used to build the cluster in this book. See Chapter 19 for a description of alternative methods available, and also see the discussion of the `ypbind` init script later in this chapter.

### **xfs**

The X font server. If you are using only the text-based terminal (which is the assumption throughout this book) to administer your server (or `telnet/ssh` sessions from a remote machine), you will not need this service on your cluster nodes, and you can reduce system boot time and disk space by not installing any X applications on your cluster nodes. See Chapter 20 for an example of how to use X applications running on Thin Clients to access services inside the cluster.

### **xinetd**<sup>11</sup>

Starts services such as FTP and telnet upon receipt of a remote connection request. Many daemons are started by `xinetd` as a result of an incoming TCP or UDP network request. Note that in a cluster configuration you may need to allow an unlimited number of connections for services started by `xinetd` (so that `xinetd` won't refuse a client computer's request

---

<sup>10</sup> See also the `mod_log_spread` project at [http://www.backhand.org/mod\\_log\\_spread](http://www.backhand.org/mod_log_spread).

<sup>11</sup> On older versions of Linux, and some versions of Unix, this is still called `inetd`.

for a service) by placing the instances = UNLIMITED line in the /etc/xinetd.conf configuration file. (You have to restart xinetd or send it a SIGHUP signal with the kill command for it to see the changes you make to its configuration files.)

### ypbind

Only used on NIS client machines. If you do not have an NIS server,<sup>12</sup> you do not need this service. One way to distribute cluster password and account information to all cluster nodes is to run an LDAP server and then send account information to all cluster nodes using the NIS system. This is made possible by a licensed commercial program from PADL software (<http://www.padl.com>) called the *NIS/LDAP gateway* (the daemon is called yplapd). Using the NIS/LDAP gateway on your LDAP server allows you to create simple cron scripts that copy all of the user accounts out of the LDAP database and install them into the local /etc/passwd file on each cluster node. You can then use the /etc/nsswitch.conf file to point user authentication programs at the local /etc/passwd file, thus avoiding the need to send passwords over the network and reducing the impact of a failure of the LDAP (or NIS) server on the cluster. See the /etc/nsswitch.conf file for examples and more information. Changes to the /etc/xinetd.conf file are recognized by the system immediately (they do not require a reboot).

**NOTE** *These entries will not rely on the /etc/shadow file but will instead contain the encrypted password in the /etc/passwd file.*

The cron job that creates the local passwd entries only needs to contain a line such as the following:

---

```
ypcat passwd > /etc/passwd
```

---

This command will overwrite all of the /etc/passwd entries on the system with the NIS (or LDAP) account entries, so you will need to be sure to create all of the normal system accounts (especially the root user account) on the LDAP server.

An even better method of applying the changes to each server is available through the use of the patch and diff commands. For example, a shell script could do the following:

---

```
ypcat passwd > /tmp/passwd
diff -e /etc/passwd /tmp/passwd > /tmp/passwd.diff
patch -be /etc/passwd /tmp/passwd.diff
```

---

---

<sup>12</sup> Usually used to distribute user accounts, user groups, host names, and similar information within a trusted network.

These commands use the `ed` editor (the `-e` option) to modify only the lines in the `passwd` file that have changed. Additionally, this script should check to make sure the NIS server is operating normally (if no entries are returned by the `ypcat` command, the local `/etc/passwd` file will be erased unless your script checks for this condition and aborts). This makes the program safe to run in the middle of the day without affecting normal system operation. (Again, note that this method does not use the `/etc/shadow` file to store password information.) In addition to the `passwd` database, similar commands should be used on the `group` and `hosts` files.

If you use this method to distribute user accounts, the LDAP server running `ypldapd` (or the NIS server) can crash, and users will still be able to log on to the cluster. If accounts are added or changed, however, all cluster nodes will not see the change until the script to update the local `/etc/passwd`, `group`, and `hosts` files runs.

When using this configuration, you will need to leave the `ypbind` daemon running on all cluster nodes. You will also need to set the domain name for the client (on Red Hat systems, this is done in the `/etc/sysconfig/network` file).

#### **ypasswd**

This is required only on an NIS master machine (it is not required on the LDAP server running the NIS/LDAP gateway software as described in the discussion of `ypbind` in the previous list item). Cluster nodes will normally always be configured with `ypasswd` disabled.

#### **ypserv**

Same as `ypasswd`.

## **In Conclusion**

This chapter has introduced you to the Linux Enterprise Cluster from the perspective of the services that will run inside it, on the cluster nodes. The normal method of starting daemons when a system boots on a Linux system using init scripts<sup>13</sup> is also used to start the daemons that will run inside the cluster. The init scripts that come with your distribution can be used on the cluster nodes to start the services you would like to offer to your users. A key feature of a Linux Enterprise Cluster is that all of the cluster nodes run the same services; all nodes inside it are the same (except for their network IP address).

In Chapter 2 I lay the foundation for understanding the Linux Enterprise Cluster from the perspective of the network packets that will be used in the cluster, and in the Linux kernel that will sit in front of the cluster.

---

<sup>13</sup> Or `rc` scripts.



# 2

## HANDLING PACKETS



When a packet from a client computer arrives on a Linux server, it must pass through the kernel before it is delivered to a daemon or service. If a service needs to failover from one server to another, the kernel's packet-handling methods must<sup>1</sup> be the same on both servers. Resources can therefore be said to include kernel packet-handling methods and rules.

This chapter explains the techniques for changing the fate of network packets inside the Linux kernel. These techniques will provide you with the foundation for building a cluster load balancer in Part II of this book. The cluster load balancer receives all packets for the cluster and forwards them to the cluster nodes thanks to the (optional) code inside the kernel, called Netfilter.

---

<sup>1</sup> Technically, they do not have to be exactly the same if the backup server is more permissive. They do, however, have to use the same packet marking rules if they rely on Netfilter's ability to mark packets. This is an advanced topic not discussed until Part III of this book.

This chapter also provides sample commands that demonstrate how you tell Netfilter to block packets before they reach the services on a Linux server; these commands can be used on the cluster load balancer and on all of the cluster nodes. These commands are normally not used when the cluster and its client computers sit behind a firewall, but they should be used on publicly accessible clusters (such as a web cluster) *even when the cluster is protected by a firewall*.

The Linux kernel can also affect the fate of a network packet by its ability to route packets. This ability is normally used on a server that has two network interface cards. Inbound packets from an untrusted network arrive on one interface card and are examined before they are allowed out the second interface card that is connected to the internal, or trusted, network.

To simplify this complex topic, we will use one network hardware configuration in all of the examples in this chapter. This hypothetical hardware configuration has two Ethernet network interface cards. The first is called `eth0` and is on an untrusted, public network. The second is called `eth1` and is connected to a private, trusted network. Later, in Part II of this book, we'll see how the private, trusted network can become a cluster.

## Netfilter

Every packet coming into the system off the network cable is passed up through the protocol stack, which converts the electronic signals hitting the network interface card into messages the services or daemons can understand. When a daemon sends a reply, the kernel creates a packet and sends it out through a network interface card. As the packets are being processed up (inbound packets) and down (outbound packets) the protocol stack, Netfilter hooks into the Linux kernel and allows you to take control of the packet's fate.

A packet can be said to *traverse* the Netfilter system, because these hooks provide the ability to modify packet information at several points within the inbound and outbound packet processing. Figure 2-1 shows the five Linux kernel Netfilter hooks on a server with two Ethernet network interfaces called `eth0` and `eth1`.

The routing box at the center of this diagram is used to indicate the routing decision the kernel must make every time it receives or sends a packet. For the moment, however, we'll skip the discussion of how routing decisions are made. (See "Routing Packets with the Linux Kernel" later in this chapter for more information.)

For now, we are more concerned with the Netfilter hooks. Fortunately, we can greatly simplify things, because we only need to know about the three basic, or default, sets of rules that are applied by the Netfilter hooks (we will not directly interact with the Netfilter hooks). These three sets of rules, or chains, are called `INPUT`, `FORWARD`, and `OUTPUT` (written in lowercase on Linux 2.2 series kernels).

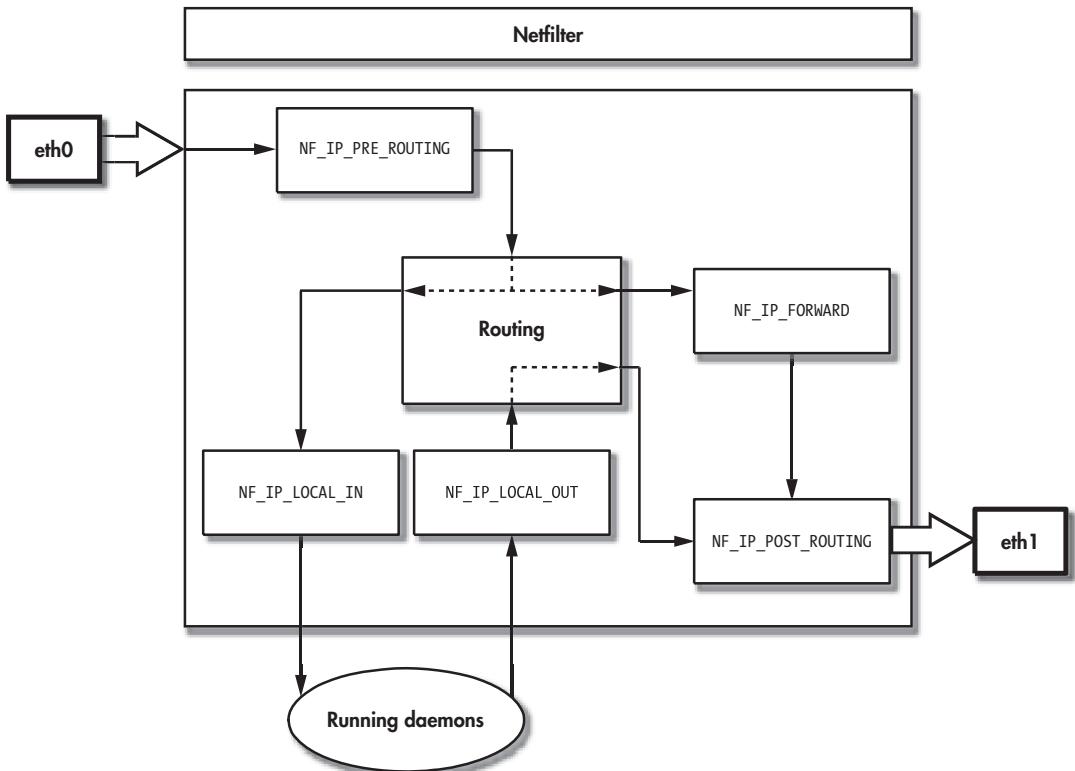


Figure 2-1: The five Netfilter hooks in the Linux kernel

**NOTE** This diagram does not show packets going out eth0.

Netfilter uses the INPUT, FORWARD, and OUTPUT lists of rules and attempts to match each packet passing through the kernel. If a match is found, the packet is immediately handled in the manner described by the rule without any attempt to match further rules in the chain. If none of the rules in the chain match, the fate of the packet is determined by the default policy you specify for the chain. On publicly accessible servers, it is common to set the default policy for the input chain to `DROP`<sup>2</sup> and then to create rules in the input chain for the packets you want to allow into the system.

## A Brief History of Netfilter

The Netfilter code changed dramatically from the 2.2 to the 2.4 series kernel (but, at least in terms of the iptables syntax, it did not change very much from the 2.4 to the 2.6 series). This means there is a big difference between ipchains and iptables. For historical purposes, we will briefly examine both in this section.

<sup>2</sup> This used to be `DENY` in ipchains.

**NOTE** You can use `ipchains` on 2.4 kernels, but not in conjunction with the Linux Virtual Server, so this book will describe `ipchains` only in connection with 2.2 series kernels.

Figure 2-2 shows the `ipchains` packet matching for the `input`, `forward`, and `output` chains in Linux 2.2 series kernels.

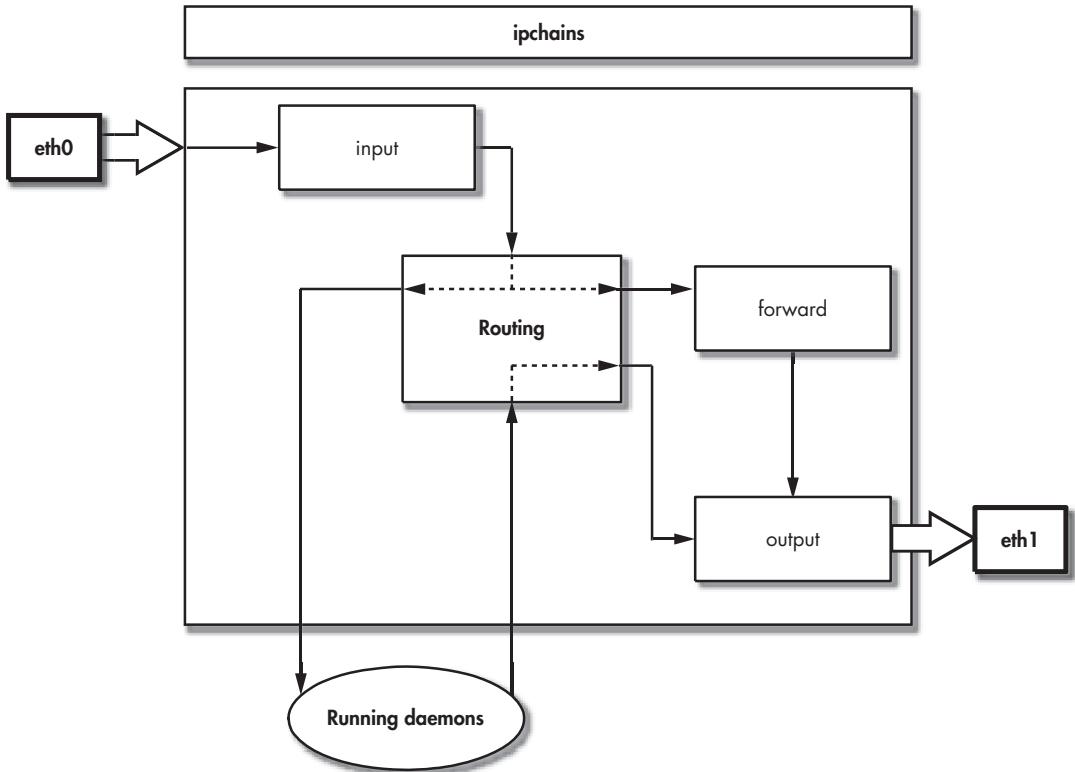


Figure 2-2: `ipchains` in the Linux kernel

Notice how a packet arriving on `eth0` and going out `eth1` will have to traverse the `input`, `forward`, and `output` chains. In Linux 2.4 and later series kernels,<sup>3</sup> however, the same type of packet would only traverse the `FORWARD` chain. When using `iptables`, each chain only applies to one type of packet: `INPUT` rules are only applied to packets destined for locally running daemons, `FORWARD` rules are only applied to packets that arrived from a remote host and need to be sent back out on the network, and `OUTPUT` rules are only applied to packets that were created locally.<sup>4</sup> Figure 2-3 shows the `iptables` `INPUT`, `FORWARD`, and `OUTPUT` rules in Linux 2.4 series kernels.

<sup>3</sup> Up to the kernel 2.6 series at the time of this writing.

<sup>4</sup> This is true for the default “filter” table. However, `iptables` can also use the “mangle” table that uses the `PREROUTING` and `OUTPUT` chains, or the “nat” table, which uses the `PREROUTING`, `OUTPUT`, and `POSTROUTING` chains. The “mangle” table is discussed in Chapter 14.

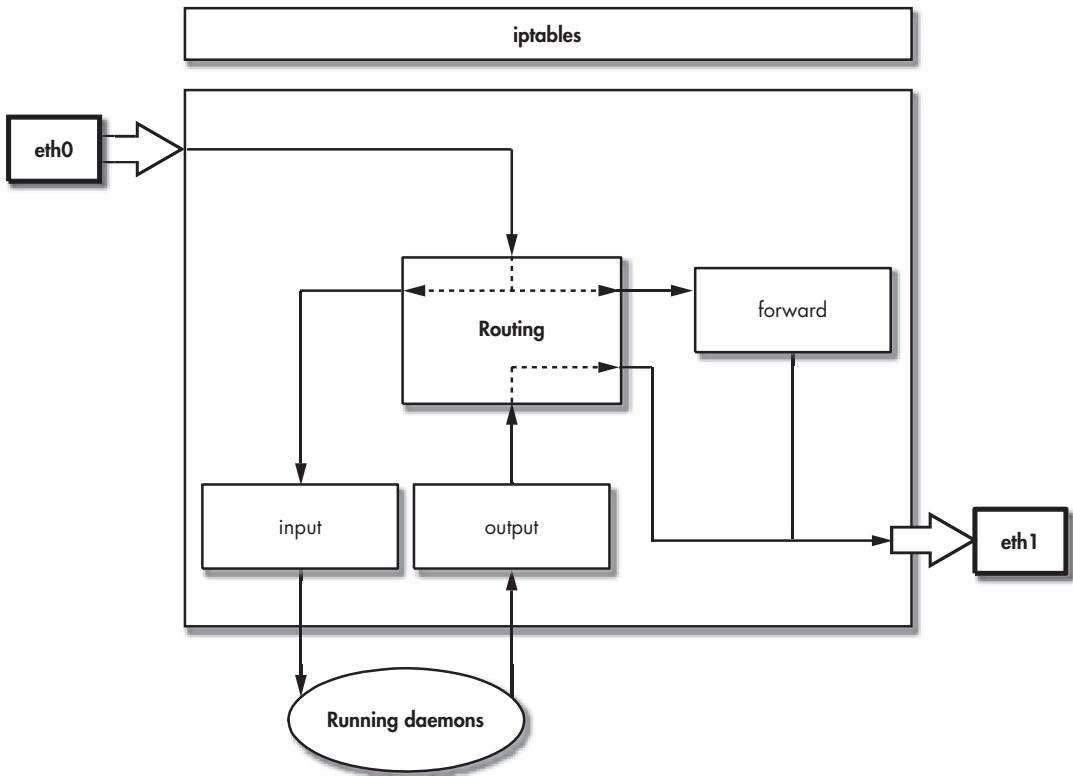


Figure 2-3: *iptables* in the Linux kernel

This change (a packet passing through only one chain depending upon its source and destination) from the 2.2 series kernel to the 2.4 series kernel reflects a trend toward simplifying the sets of rules, or chains, to make the kernel more stable and the Netfilter code more sensible. With ipchains or iptables commands (these are command-line utilities), we can use the three chains to control which packets get into the system, which packets are forwarded, and which packets are sent out without worrying about the specific Netfilter hooks involved—we only need to remember when these rules are applied. We can summarize the chains as follows:

### Summary of ipchains for Linux 2.2 series kernels

Packets destined for locally running daemons:

input

Packets from a remote host destined for a remote host:

input

forward

output

Packets originating from locally running daemons:

output

## Summary of iptables for Linux 2.4 and later series kernels

Packets destined for locally running daemons:

INPUT

Packets from a remote host destined for a remote host:

FORWARD

Packets originating from locally running daemons:

OUTPUT

In this book, we will use the term `iptables` to refer to the program that is normally located in the `/sbin` directory. On a Red Hat system, `iptables` is:

- A program in the `/sbin` directory
- A boot script in the `/etc/init.d`<sup>5</sup> directory
- A configuration file in the `/etc/sysconfig` directory

The same could be said for `ipchains`, but again, you will only use one method: `iptables` or `ipchains`.

**NOTE** *For more information about `ipchains` and `iptables`, see <http://www.netfilter.org>. Also, look for Rusty Russell’s Remarkably Unreliable Guides to the Netfilter Code or Linux Firewalls by Robert L. Ziegler, from New Riders Press. Also see the section “Netfilter Marked Packets” in Chapter 14.*

## Setting the Default Chain Policy

You can use the `ipchains` or the `iptables` utility to set the default policy for each chain. These chains normally have a default policy that allows anything to pass through (`ACCEPT`) for packets that are not explicitly matched by a rule in the chain. You can change the default policy to either `DROP` or `REJECT`. `DROP` means to ignore the packet, and `REJECT` means the sender should be told the packet was not accepted.

This control over network packets allows you to turn any Linux computer into a firewall for your network. In the next section, we’ll describe how to apply firewall rules to any Linux machine and how to make these rules permanent.

In Chapter 14, we will see how the Linux Virtual Server uses the Netfilter hooks to gain access to network packets and manipulate them based on a different set of criteria—namely, the criteria you establish for balancing incoming packets across the cluster nodes.

---

<sup>5</sup> Recall from Chapter 1 that `/etc/init.d` is a symbolic link to the `/etc/rc.d/init.d` directory on Red Hat systems.

## Using iptables and ipchains

In this section, I will provide you with the iptables (and ipchains) commands you can use to control the fate of packets as they arrive on each cluster node. These commands can also be used to build a firewall outside the cluster. This is done by adding a corresponding FORWARD rule for each of the INPUT rules provided here and installing them on the cluster load balancer. Whether you install these rules on each cluster node, the cluster load balancer, or both is up to you.

These commands can be entered from a shell prompt; however, they will be lost the next time the system boots. To make the rules permanent, you must place them in a script (or a configuration file used by your distribution) that runs each time the system boots. Later in this chapter, you will see a sample script.

**NOTE** *If you are working with a new kernel (series 2.4 or later), use the iptables rules provided (the ipchains rules are for historical purposes only).*

### Clear Existing Rules

Before adding any rules to the kernel, we want to start with a clean, rule-free configuration. ipchains and iptables are useful commands to use when troubleshooting and building a new configuration if you need to start over and don't want to reboot the system.

Use one of these commands (ipchains on 2.2 series kernels and iptables on 2.4 and later series kernels) to erase all rules in all three chains (INPUT, FORWARD, and OUTPUT):

---

```
#ipchains -F
#iptables -F
```

---

### Set the Default INPUT Chain Policy

Use one of these commands to set the default INPUT policy to not allow incoming packets:

---

```
#ipchains -P input DENY
#iptables -P INPUT DROP
```

---

Remember, if you create routing rules to route packets back out of the system (described later in this chapter), you may also want to set the default FORWARD policy to DROP and explicitly specify the criteria for packets that are allowed to pass through. This is how you build a secure load balancer (in conjunction with the Linux Virtual Server software described in Part III of this book).

Recall that the 2.2 series kernel ipchains input policy affects both locally destined packets and packets that need to be sent back out on the network, whereas the 2.4 and later series kernel iptables INPUT policy affects only locally destined packets.

**NOTE** If you need to add *iptables* (or *ipchains*) rules to a system through a remote telnet or SSH connection, you should first enter the ACCEPT rules (shown later in this section) to allow your telnet or SSH to continue to work even after you have set the default policy to DROP (or DENY).

## FTP

Use these rules to allow inbound FTP connections from anywhere:

---

```
#ipchains -A input -i eth0 -p tcp -s any/0 1024:65535 -d MY.NET.IP.ADDR 21 -j ACCEPT
#ipchains -A input -i eth0 -p tcp -s any/0 1024:65535 -d MY.NET.IP.ADDR 20 -j ACCEPT
#iptables -A INPUT -i eth0 -p tcp -s any/0 --sport 1024:65535 -d MY.NET.IP.ADDR
--dport 21 -j ACCEPT
#iptables -A INPUT -i eth0 -p tcp -s any/0 --sport 1024:65535 -d MY.NET.IP.ADDR
--dport 20 -j ACCEPT
```

---

Depending on your version of the kernel, you will use either *ipchains* or *iptables*. Use only the two rules that match your utility.

Let's examine the syntax of the two *iptables* commands more closely:

**-A INPUT**

Says that we want to add a new rule to the input filter.

**-i eth0**

Applies only to the *eth0* interface. You can leave this off if you want an input rule to apply to all interfaces connected to the system. (Using *ipchains*, you can specify the input interface when you create a rule for the output chain, but this is no longer possible under *iptables* because the output chain is for locally created packets only.)

**-p tcp**

This rule applies to the TCP protocol (required if you plan to block or accept packets based on a port number).

**-s any/0 and --sport 1024:65535**

These arguments say the source address can be anything, and the source port (the TCP/IP port we will send our reply to) can be anything between 1024 and 65535. The */0* after any means we are not applying a subnet mask to the IP address. Usually, when you specify a particular IP address, you will need to enter something like:

---

*ALLOWED.NET.IP.ADDR/255.255.255.0*

---

or

---

*ALLOWED.NET.IP.ADDR/24*

---

where *ALLOWED.NET.IP.ADDR* is an IP address such as 209.100.100.3.

In both of these examples, the *iptables* command masks off the first 24 bits of the IP address as the network portion of the address.

**NOTE** For the moment, we will skip over the details of a TCP/IP conversation. What is important here is that a normal inbound TCP/IP request will come in on a privileged port (a port number defined in /etc/services and in the range of 1-1023) and ask you to send a reply back on an unprivileged port (a port in the range of 1024 to 65535).

**-d MY.NET.IP.ADDR and --dport 21**

These two arguments specify the destination IP address and destination port required in the packet header. Replace *MY.NET.IP.ADDR* with the IP address of your eth0 interface, or enter 0.0.0.0 if you are leaving off the -i option and do not care which interface the packet comes in on. (FTP requires both ports 20 and 21; hence the two iptables commands above.)

**-j ACCEPT**

When a packet matches a rule, it drops out of the chain (the INPUT chain in this case). At that point, whatever you have entered here for the -j option specifies what happens next to the packet. If a packet does not match any of the rules, the default filter policy, DENY or DROP in this case, is applied.

## **Passive FTP**

The rules just provided for FTP do not allow for passive FTP connections to the FTP service running on the server. FTP is an old protocol that dates from a time when university computers connected to each other without intervening firewalls to get in the way. In those days, the FTP server would connect back to the FTP client to transmit the requested data. Today, most clients sit behind a firewall, and this is no longer possible. Therefore, the FTP protocol has evolved a new feature called passive FTP that allows the client to request a data connection on port 21, which causes the server to listen for an incoming data connection from the client (rather than connect back to the client as in the old days). To use passive FTP, your FTP server must have an iptables rule that allows it to listen on the unprivileged ports for one of these passive FTP data connections. The rule to do this using iptables looks like this:

---

```
#iptables -A INPUT -i eth0 -p tcp -s any/0 --sport 1024:65535 -d MY.NET.IP.ADDR  
--dport 1024:65535 -j ACCEPT
```

---

## **DNS**

Use this rule to allow inbound DNS requests to a DNS server (the “named” or BIND service):

---

```
#ipchains -A input -i eth0 -p udp -s any/0 1024:65535 -d MY.NET.IP.ADDR 53 -j ACCEPT  
#ipchains -A input -i eth0 -p tcp -s any/0 1024:65535 -d MY.NET.IP.ADDR 53 -j ACCEPT  
#iptables -A INPUT -i eth0 -p udp -s any/0 --sport 1024:65535 -d MY.NET.IP.ADDR  
--dport 53 -j ACCEPT  
#iptables -A INPUT -i eth0 -p tcp -s any/0 --sport 1024:65535 -d MY.NET.IP.ADDR  
--dport 53 -j ACCEPT
```

---

## **Telnet**

Use this rule to allow inbound telnet connections from a specific IP address:

---

```
#ipchains -A input -i eth0 -p tcp -s 209.100.100.10 1024:65535
-d MY.NETWORK.IP.ADDR 23 -j ACCEPT
#iptables -A INPUT -i eth0 -p tcp -s 209.100.100.10 --sport 1024:65535
-d MY.NETWORK.IP.ADDR --dport 23 -j ACCEPT
```

---

In this example, we have only allowed the client computer using IP address `209.100.100.10` telnet access into the system. You can change the IP address of the client computer and enter this command repeatedly to grant telnet access to additional client computers.

## **SSH**

Use this rule to allow SSH connections:

---

```
ipchains -A input -i eth0 -p tcp -s 209.200.200.10 1024:65535
-d MY.NETWORK.IP.ADDR 22 -j ACCEPT
iptables -A INPUT -i eth0 -p tcp -s 209.200.200.10 --sport 1024:65535
-d MY.NETWORK.IP.ADDR --dport 22 -j ACCEPT
```

---

Again, replace the example `209.200.200.10` address with the IP address you want to allow in, and repeat the command for each IP address that needs SSH access. The SSH daemon (`sshd`) listens for incoming requests on port 22.

## **Email**

Use this rule to allow email messages to be sent out:

---

```
ipchains -A input -i eth0 -p tcp ! -y -s EMAIL.NET.IP.ADDR 25
-d MY.NETWORK.IP.ADDR 1024:65535 -j ACCEPT
iptables -A INPUT -i eth0 -p tcp ! --syn -s EMAIL.NET.IP.ADDR --sport 25
-d MY.NETWORK.IP.ADDR --dport 1024:65535 -j ACCEPT
```

---

This rule says to allow Simple Mail Transport Protocol (SMTP) replies to our request to connect to a remote SMTP server for outbound mail delivery. This command introduces the `! --syn` syntax, an added level of security that confirms that the packet coming in is really a reply packet to one we've sent out.

The `! -y` syntax says that the acknowledgement flag must be set in the packet header—indicating the packet is in acknowledgement to a packet we've sent out. (We are sending the email out, so we started the SMTP conversation.)

This rule will not allow inbound email messages to come in to the sendmail service on the server. If you are building an email server or a server capable of processing inbound email messages (cluster nodes that do email order entry processing, for example), you must allow inbound packets to port 25 (without using the acknowledgement flag).

## HTTP

Use this rule to allow inbound HTTP connections from anywhere:

---

```
#ipchains -A input -i eth0 -p tcp -d MY.NETWORK.IP.ADDR 80 -j ACCEPT
#iptables -A INPUT -i eth0 -p tcp -d MY.NETWORK.IP.ADDR --dport 80 -j ACCEPT
```

---

To allow HTTPS (secure) connections, you will also need the following rule:

---

```
#ipchains -A input -i eth0 -p tcp -d MY.NETWORK.IP.ADDR 443 -j ACCEPT
#iptables -A INPUT -i eth0 -p tcp -d MY.NETWORK.IP.ADDR --dport 443 -j ACCEPT
```

---

## ICMP

Use this rule to allow Internet Control Message Protocol (ICMP) responses:

---

```
ipchains -A input -i eth0 -p icmp -d MY.NETWORK.IP.ADDR -j ACCEPT
iptables -A INPUT -i eth0 -p icmp -d MY.NETWORK.IP.ADDR -j ACCEPT
```

---

This command will allow all ICMP messages, which is not something you should do unless you have a firewall between the Linux machine and the Internet with properly defined ICMP filtering rules. Look at the output from the command `iptables -p icmp -h` (or `ipchains icmp -help`) to determine which ICMP messages to enable. Start with only the ones you know you need. (You probably should allow MTU discovery, for example.)

## Review and Save Your Rules

To view the rules you have created, enter:

---

```
ipchains -L -n
iptables -L -n
```

---

To save your iptables rules to the `/etc/sysconfig/iptables` file on Red Hat so they will take effect each time the system boots, enter:

---

```
#/etc/init.d/iptables save
```

---

**NOTE** *The command `service iptables save` would work just as well here.*

However, if you save your rules using this method as provided by Red Hat, you will not be able to use them as a high-availability resource.<sup>6</sup> If the cluster load balancer uses sophisticated iptables rules to filter network packets, or, what is more likely, to mark certain packet headers for special handling,<sup>7</sup> then these rules may need to be on only one machine (the load balancer) at a time.<sup>8</sup> If this is the case, the iptables rules should be placed in a script such as the one shown at the end of this chapter and configured as a high-availability resource, as described in Part II of this book.

## Routing Packets with the Linux Kernel

As shown in Figures 2-2 and 2-3, all inbound and outbound network packets must pass through the kernel's routing rules. And, like the list of rules in an iptables chain, the list of routing rules is contained in an ordered list that is examined only until a packet matches one of the rules.

Like the rules in an iptables chain, each routing rule includes:

- The criteria used to match packets
- Instructions telling the kernel what to do with packets that match this criteria

The order of the routing rules is important because the kernel stops searching the routing table when a packet matches one of the routing rules. Because the last rule in the routing table will normally match any packet, it is called the *default route* (or *default gateway*).

When you assign an IP address to one of the network interface cards, the kernel automatically adds a routing rule for the interface. The kernel constructs the routing table entry so it will match all packets containing a destination address that falls within the range of IP addresses allowed on the network or subnet that the interface is connected to.

In a typical configuration, routing table entries are constructed that will match packets based on the packet's destination IP address. This type of packet is typically matched for:

- A particular destination network
- A particular destination host
- Any destination or host

**NOTE** *In the following sections, the add argument to the route command adds a routing rule. To delete or remove a routing rule after it has been added, use the same command but change the add argument to del.*

---

<sup>6</sup> This is because the standard Red Hat iptables init script does not return something the Heartbeat program will understand when it is run with the status argument. For further discussion of this topic, see the example script at the end of this chapter.

<sup>7</sup> Marking packets with Netfilter is covered in Chapter 14.

<sup>8</sup> This would be true in an LVS-DR cluster that is capable of failing over the load-balancing service to one of the real servers inside the cluster. See Chapter 15.

## **Matching Packets for a Particular Destination Network**

If we had to type in the command to add a routing table entry for a NIC connected to the 209.100.100.0 network, the command would look like this (we do not, because the kernel will do this for us automatically):

---

```
#/sbin/route add --net 209.100.100.0 netmask 255.255.0.0 dev eth0
```

---

This command adds a routing table entry to match all packets destined for the 209.100.100.0 network and sends them out the eth0 interface.

If we also had a NIC called eth1 connected to the 172.24.150.0 network, we would use the following command as well:

---

```
#/sbin/route add --net 172.24.150.0. netmask 255.255.255.0 dev eth1
```

---

With these two routing rules in place, the kernel can send and receive packets to any host connected to either of these two networks. If this machine needs to also forward packets that originate on one of these networks to the other network (the machine is acting like a router), you must also enable the kernel's packet forwarding feature with the following command:

---

```
#/bin/echo 1 > /proc/sys/net/ipv4/ip_forward
```

---

As we will see in Part III of this book, this kernel capability is also a key technology that enables a Linux machine to become a cluster load balancer.

## **Matching Packets for a Particular Destination Host**

Now let's assume we have a Linux host called `Linux-Host-1` with two NICs connected to two different networks. We want to be able to send packets to `Linux-Host-2`, which is not physically connected to either of these two networks (see Figure 2-4). If we can only have one default gateway, how will the kernel know which NIC and which router to use?

If we only need to be able to send packets to one host on the 192.168.150.0 network, we could enter a routing command on `Linux-Host-1` that will match all packets destined for this one host:

---

```
#/sbin/route add -host 192.168.150.33 gw 172.24.150.1
```

---

This command will force all packets destined for 192.168.150.33 to the internal gateway at IP address 172.24.150.1. Assuming that the gateway is configured properly, it will know how to forward the packets it receives from `Linux-Host-1` to `Linux-Host-2` (and vice versa).

If, instead, we wanted to match *all* packets destined for the 192.168.150.0 network, and not just the packets destined for `Linux-Host-2`, we would enter a routing command like this:

---

```
#/sbin/route add --net 192.168.150.0 netmask 255.255.255.0 gw 172.24.150.1
```

---

**NOTE** In both of these examples, the kernel knows which physical NIC to use thanks to the rules described in the previous section that match packets for a particular destination network. (The 172.24.150.1 address will match the routing rule added for the 172.24.150.0 network that sends packets out the eth1 interface.)

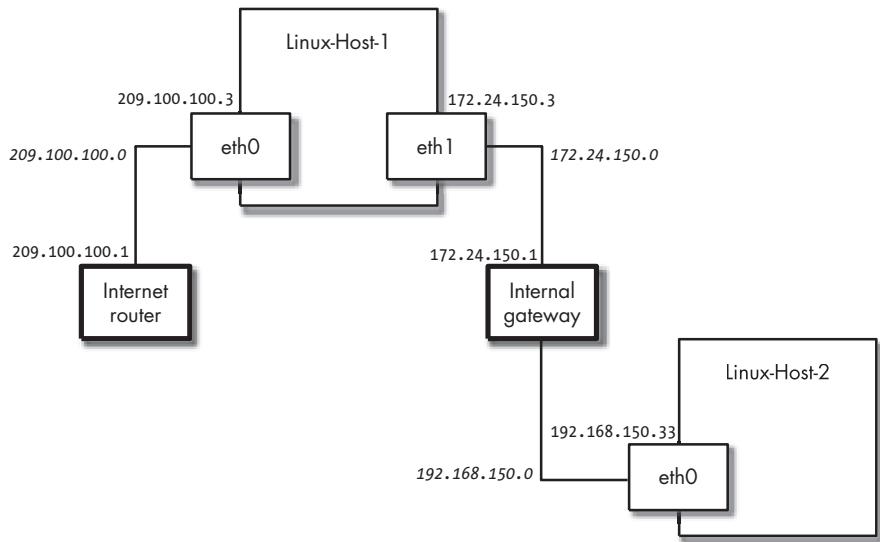


Figure 2-4: Linux firewall and router example network

### Matching Packets for Any Destination or Host

As described previously, the last rule in the routing table, the default route, is normally used to match all packets regardless of their destination address and force them to a router or internal gateway device. The command to create this type of routing rule looks like this:

---

```
#/sbin/route add --default gw 172.24.150.1
```

---

This routing rule causes the kernel to send packets that do not match any of the previous routing rules to the internal gateway at IP address 172.24.150.1.

If Linux-Host-1 needs to send packets to the Internet through the router (209.100.100.1), and it also needs to send packets to any host connected to the 192.168.150.0 network, use the following commands:

---

```
#/sbin/ifconfig eth0 209.100.100.3 netmask 255.255.255.0
#/sbin/ifconfig eth1 172.24.150.3 netmask 255.255.255.0
#/sbin/route add -net 192.168.150.0 netmask 255.255.255.0 gw 172.24.150.1
#/sbin/route add default gw 209.100.100.1
```

---

The first two commands assign the IP addresses and network masks to the eth0 and eth1 interfaces and cause the kernel to automatically add routing table entries for the 209.100.100.0 and 172.24.150.0 networks.

The third command causes the kernel to send packets for the 192.168.150.0 network out the eth1 network interface to the 172.24.150.1 gateway device. Finally, the last command matches all other packets, and in this example, sends them to the Internet router.

## To View Your Routing Rules

To view your list of routing rules, use any of the following three commands:

---

```
#netstat -rn  
#route -n  
#ip route list
```

---

The output of the first two commands (which may be slightly more readable than the list produced by the ip command) looks like this:

---

Kernel IP routing table							
Destination	Gateway	Genmask	Flags	MSS	Window	irtt	Iface
209.100.100.0	0.0.0.0	255.255.255.0	U	0	0	0	eth0
192.168.150.0	172.24.150.1	255.255.255.0	UG	0	0	0	eth1
172.24.150.0	0.0.0.0	255.255.255.0	U	0	0	0	eth1
127.0.0.0	0.0.0.0	255.0.0.0	U	0	0	0	lo
0.0.0.0	209.100.100.1	0.0.0.0	UG	0	0	0	eth0

---

Notice that the destination address for the final routing rule is 0.0.0.0, which means that any destination address will match this rule. (You'll find more information for this report on the netstat and route man pages.)

## Making Your Routing Rules Permanent

Like the Netfilter iptables rules, the routing rules do not survive a system reboot. To make your routing rules permanent, place them into a script (and add them to the boot process with chkconfig or ntsysv, as described in Chapter 1) or use the method provided by your distribution.

## *iptables and Routing Resource Script*

As with iptables, the standard Red Hat method of saving the routing rules is not the best one to use when you need your rules to failover from one server to another.<sup>9</sup> In a cluster configuration, we want to enable and disable packet handling methods just like any other service. Let's therefore expand our definition of a service to include capabilities offered *inside* the kernel. (Chapter 3 describes how to configure capabilities within the Linux kernel and recompile it.)

---

<sup>9</sup> Red Hat uses the network init script to enable routing rules. The routing rules cannot be disabled or stopped without bringing down all network communication when using the standard Red Hat init script.

To failover kernel capabilities such as the iptables and routing rules, we need to use a script that can start and stop them on demand. This can be done for the kernel's packet-handling service, as we'll call it, with a script like this:

---

```
#!/bin/bash
#
# Packet Handling Service
#
# chkconfig 2345 55 45
# description: Starts or stops iptables rules and routing

case "$1" in
start)
    # Flush (or erase) the current iptables rules
    /sbin/iptables -F
    /sbin/iptables --table nat -flush
    /sbin/iptables --table nat --delete-chain

    # Enable the loopback device for all types of packets
    # (Normally for packets created by local daemons for delivery
    # to local daemons)
    /sbin/iptables -A INPUT -i lo -p all -j ACCEPT
    /sbin/iptables -A OUTPUT -o lo -p all -j ACCEPT
    /sbin/iptables -A FORWARD -o lo -p all -j ACCEPT

    # Set the default policies
    /sbin/iptables -P INPUT DROP
    /sbin/iptables -P FORWARD DROP
    /sbin/iptables -P OUTPUT ACCEPT

    # NAT
    /sbin/iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE

    # Allow inbound packets from our private network
    /sbin/iptables -A INPUT -i eth1 -j ACCEPT
    /sbin/iptables -A FORWARD -i eth1 -j ACCEPT

    # Allow packets back in from conversations we initiated
    # from the private network.
    /sbin/iptables -A FORWARD -i eth0 --match state --state ESTABLISHED,RELATED -j ACCEPT
    /sbin/iptables -A INPUT --match state --state ESTABLISHED,RELATED -j ACCEPT

    # Allow Sendmail and POP (from anywhere, but really what we
    # are allowing here is inbound connections on the eth0 interface).
    # (Sendmail and POP are running locally on this machine).
    /sbin/iptables -A INPUT --protocol tcp --destination-port 25 -j ACCEPT
    /sbin/iptables -A INPUT --protocol tcp --destination-port 110 -j ACCEPT

    # Routing Rules --
```

```

# Route packets destined for the 192.168.150.0 network using the internal
# gateway machine 172.24.150.1
/sbin/route add -net 192.168.150.0 netmask 255.255.255.0 gw 172.24.150.1

# By default, if we don't know where a packet should be sent we
# assume it should be sent to the Internet router.
/sbin/route add default gw 209.100.100.1

# Now that everything is in place we allow packet forwarding.
echo 1 > /proc/sys/net/ipv4/ip_forward

;;
stop)
# Flush (or erase) the current iptables rules
/sbin/iptables -F

# Set the default policies back to ACCEPT
# (This is not a secure configuration.)
/sbin/iptables -P INPUT ACCEPT
/sbin/iptables -P FORWARD ACCEPT
/sbin/iptables -P OUTPUT ACCEPT

# Remove our routing rules.
/sbin/route del -net 192.168.150.0 netmask 255.255.255.0 gw 172.24.150.1
/sbin/route del default gw 209.100.100.1

# Disable packet forwarding
echo 0 > /proc/sys/net/ipv4/ip_forward

;;
status)
enabled=`/bin/cat /proc/sys/net/ipv4/ip_forward`
if [ "$enabled" -eq 1 ]; then
    echo "Running"
else
    echo "Down"
fi
;;
*)
    echo "Requires start, stop or status"
;;
esac

```

This script demonstrates another powerful iptables capability: the ability to do network address translation or NAT to masquerade internal hosts. In this example, all hosts sending requests out to the Internet through `Linux-Host-1` will have their IP addresses (the return or source address in the packet header) converted to the source IP address of the `Linux-Host-1` machine. This is a common and effective way to prevent crackers from attacking internal hosts—the internal hosts use an IP address that cannot be routed on the

Internet<sup>10</sup> and the NAT machine, using one public IP address, appears to send all requests to the Internet. (This rule is placed in the `POSTROUTING` Netfilter chain, or hook, so it can apply the masquerade for outbound packets going out `eth1` and then de-masquerade the packets from Internet hosts as their replies are returned back through `eth1`. See the `NF_IP_POSTROUTING` hook in Figure 2-1 and imagine what the drawing would look like if it were depicting packets going in the other direction—from `eth1` to `eth0`.)

The iptables' ability to perform network address translation is another key technology that enables a cluster load balancer to forward packets to cluster nodes and reply to client computers as if all of the cluster nodes were one server (see Chapter 12).

This script also introduces a simple way to determine whether the packet handling service is active, or running, by checking to see whether the kernel's flag to enable packet forwarding is turned on (this flag is stored in the kernel special file `/proc/sys/net/ipv4/ip_forward`).

To use this script, you can place it in the `/etc/init.d` directory<sup>11</sup> and enter the command:

---

```
#/etc/init.d/routing start
```

---

**NOTE** *The command `service routing start` would work on a Red Hat system.*

And to check to see whether the system has routing or packet handling enabled, you would enter the command:

---

```
#/etc/init.d/routing status
```

---

**NOTE** *In Part II of this book, we will see why the word `Running` was used to indicate packet handling is enabled.<sup>12</sup>*

### ***The ip Command***

Advanced routing rules can be built using the `ip` command. Advanced routing rules allow you to do things like match packets based on their source IP address, match packets based on a number (or mark) inserted into the packet header by the iptables utility, or even specify what source IP address to use when sending packets out a particular interface. (We'll cover packet marking with iptables in Chapter 14. If you find that you are reaching limitations with the `route` utility, read about the `ip` utility in the "Advanced Routing HOWTO."<sup>13</sup>)

**NOTE** *To build the cluster described in this book, you do not need to use the `ip` command.*

---

<sup>10</sup> RFC 1918 IP address ranges will be discussed in more detail in Chapter 12.

<sup>11</sup> This script will also need to have execute permissions (which can be enabled with the command: `#chmod 755 /etc/init.d/routing`).

<sup>12</sup> So the Heartbeat program can manage this resource. Heartbeat expects one of the following: `Running`, `running`, or `OK` to indicate the service for the resource is running.

<sup>13</sup> See <http://lartc.org>.

## In Conclusion

One of the distinguishing characteristics of a Linux Enterprise Cluster is that it is built using packet routing and manipulation techniques. As I'll describe in Part III, Linux machines become cluster load balancers or cluster nodes because they have special techniques for handling packets. In this chapter I've provided you with a brief overview of the foundation you need for understanding the sophisticated cluster packet-handling techniques that will be introduced in Part III.



# 3

## **COMPILING THE KERNEL**



In this chapter I'll describe how to compile a Linux kernel. For many system administrators not familiar with Linux, this may sound like a risky thing to do on a server that will be used in production, but once you have done it a few times you'll understand one of the reasons why Linux is so powerful: you have complete control over the operating system.

I could say a lot more about the benefits of having the source code of your operating system, but we'll save that for later. For now, we'll focus on what you need to be able to put your own Linux kernel into production.

To install a Linux kernel you need to do five things:

1. Get the source code.
2. Set the options you want.
3. Compile the code.
4. Install the object code.
5. Configure the boot loader.

## What You Will Need

Before you compile the kernel, you'll need to install the gcc compiler and its dependencies. (This software is not included with this book because your distribution should already contain it.) Here is a list of packages that are required to compile the kernel (this list is taken from the Red Hat distribution):

- binutils
- cpp
- gcc (check the README file included with the kernel source code to find out which version of the gcc compiler is required<sup>1</sup>).
- glibc-debug
- glibc-devel
- glibc-kernelheaders
- modutils (for kernel versions in the 2.4 series) or module-init-utils (for kernel versions in the 2.6 series)

If you use a text-based terminal, you will also need to have the ncurses package installed so that the `make menuconfig` command will work. Two RPM packages are required for this on Red Hat systems:

- ncurses
- ncurses-devel

Once you've installed these packages, you're ready to begin. (If you have dependency problems when installing these packages, see Appendix D for some tips.)

## Step 1: Get the Source Code

You'll have to decide whether to use the "stock" version of the Linux source code or the kernel source code included with your distribution. Here are a few of the pros and cons of each method.

### ***Using the Stock Kernel***

Pros of using the stock kernel:

- You can apply your own patches to customize the kernel.<sup>2</sup>
- It is supported by the entire kernel-development community.
- It allows more control over which patches are applied to your system.
- You can decide which kernel version you want to use.

---

<sup>1</sup> Each kernel release should only be compiled with the version of gcc specified in the README file.

<sup>2</sup> For example, you can apply the hidden loopback interface kernel patch that we'll use to build the real servers inside the cluster in Part III of this book.

Con of using the stock kernel: It may invalidate support from your distribution vendor.

## ***Using the Kernel Supplied with Your Distribution***

Pros of using the distribution kernel:

- It is usually supported by your distribution vendor (though the vendor may not support it if you recompile it with changes).
- It is known to compile.
- It is already running on your system.
- It is easy to install.
- It has been tested by your distribution vendor and is likely to contain bug fixes that are not yet a part of the stock version of the kernel.

Cons of using the distribution kernel:

- You may not be able to apply kernel patches.
- It may have bugs introduced by your distribution vendor that are not found in the stock version of the kernel, and you may depend on your distribution vendor to fix them.

## ***Decide Which Kernel Version to Use***

If you use the kernel source code supplied by your vendor, you won't have many decisions to make when it comes to picking a kernel version to use; your distribution vendor will do that for you. However, if you decide to use the stock version of the kernel, you'll have to do a little more research to determine which kernel version to use.

You can use the following command to see which kernel version you are currently running:

---

```
#uname -a
```

---

**NOTE** *If you decide to download your own kernel, select an even-numbered one (2.2, 2.4, 2.6, and so on). Even-numbered kernels are considered the stable ones.*

Here are sample commands to copy the kernel source code from the CD-ROM included with this book onto your system's hard drive:

---

```
#mount /mnt/cdrom
#cd /mnt/cdrom/chapter3
#cp linux-*.tar.gz /usr/src
#cd /usr/src
#tar xzvf kernel-*
```

---

**NOTE** *Leave the CD-ROM mounted until you complete the next step.*

Now you need to create a symbolic link so that your kernel source code will appear to be in the `/usr/src/linux` directory. But first you must remove the existing symbolic link. The commands to complete this step look like this:

---

```
#rm /usr/src/linux
#ln -s /usr/src/linux-<version> /usr/src/linux
#cd /usr/src/linux
```

---

**NOTE** *The preceding `rm` command above will not remove `/usr/src/linux` if it is a directory instead of a file. (If `/usr/src/linux` is a directory on your system, move it instead with the `mv` command.)*

You should now have your shell's current working directory set to the top level directory of the kernel source tree, and you're ready to go on to the next step.

## Step 2: Set the Options You Want

Your actions here will depend upon whether you decide to install a new kernel, upgrade your existing kernel source code, or patch your existing kernel source code.

### ***Installing a New Kernel***

If you decide to install a new kernel (rather than patch or upgrade your existing one), you must first clean the source code tree of any files left behind from previous kernel compilations with the `make clean` command. Then use the `make mrproper` command to erase all `.config` files from the top level of the source code directory tree so that you can start with a clean slate. The `make oldconfig` command is then used to build a fresh `.config` file from scratch. Thus, the commands you would enter in the top level of the kernel source tree to prepare it for a new compilation are as follows:

---

```
#make clean
#make mrproper
make oldconfig
```

---

### ***Upgrading or Patching the Kernel***

If you decide to upgrade or patch your existing kernel, save a copy of your existing `/usr/src/linux/.config` file first. Select an archival directory that you'll remember, and use it to store your old `.config` files in case you ever need to go back to an old kernel configuration. For example, to save the current `.config` file from the top level of your kernel source tree to the `/boot` directory, you would use a command that looks like this:

---

```
#cp .config /boot/config-<version>
```

---

**NOTE** When upgrading to a newer version of the kernel, you can copy your old `.config` file to the top level of the new kernel source tree and then issue the `make oldconfig` command to upgrade your `.config` file to support the newer version of the kernel.

## ***Upgrading a Kernel from a Distribution Vendor***

If you are upgrading your existing kernel source code, and this kernel source code was supplied by your distribution vendor, you should locate the `.config` file that was used to originally build the kernel you are currently running on your system. The distribution vendor may have left this configuration file in the top level of their kernel source distribution tree, or they may have a separate directory where they place a variety of `.config` files for different hardware architectures.

When you find this configuration file, be sure to place it in the top-level directory of your kernel source tree into the file `.config`. If you are upgrading to a newer version of the kernel, you should then issue the command:

---

```
#make oldconfig
```

---

This command will prompt you for any kernel options that have changed since the `.config` file was originally created by your distribution vendor, so it never hurts to run this command (even when you are not upgrading your kernel) to make sure you are not missing any of the options required to compile your kernel.

**NOTE** In the top of the kernel source tree you'll also find a `Makefile`. You can edit this file and change the value of the `EXTRAVERSION` number that is specified near the top of the file. If you do this, you will be assigning a string of characters that will be added to the end of the kernel version number. Specifying an `EXTRAVERSION` number is optional, but it may help you avoid problems when your boot drive is a SCSI device (see the "SCSI Gotchas" section near the end of this chapter).

## ***Set Your Kernel Options***

You now need to set your kernel options regardless of whether you installed a new kernel, are upgrading your existing kernel source code, or are patching your existing kernel source code.

You select the options you want to enable in your kernel from a text console screen by running the command:

---

```
#make menuconfig
```

---

One of the most important options you can select from within the `make menuconfig` utility is whether or not your kernel compiles as a monolithic or a modular kernel. When you specify the option to compile the kernel as a modular kernel (see "A Note About Modular Kernels" for details), a large portion of the compiled kernel code is stored into files in the `/lib/modules`

directory on the hard drive, and it is only used when it is needed. If you select the option to compile the kernel as a monolithic kernel (you disable modular support in the kernel), then all of the kernel code will be compiled into a single file.

Knowing which of the other kernel options to enable or disable could be the subject of another book as thick as this one, so we won't go into much detail here. However, if you are using a 2.4 kernel, you may want to start by selecting nearly all of the kernel options available and building a modular kernel. This older kernel code is likely to compile (thanks to the efforts of thousands of people finding and fixing problems), and this modular kernel will only load the kernel code (the modules) necessary to make the system run. Newer kernels, on the other hand, may require a more judicious selection of options to avoid kernel compilation problems.

**NOTE** *The best place to find information about the various kernel options is to look at the inline kernel help screens available with the make menuconfig command. These help screens usually also contain the name of the module that will be built when you compile a modular kernel.*

### A NOTE ABOUT MODULAR KERNELS

Modular kernels require more effort to implement when using a SCSI boot disk (see "SCSI Gotchas" later in this chapter). However, they are more efficient because they load or run only the kernel modules required, as they are needed. For example, the CD-ROM driver will be unloaded from memory when the CD-ROM drive has not been used for a while. Modular kernels allow you to compile a large number of modules for a variety of hardware configurations (several different NIC drivers, for example) and build a one-size-fits-all, generic kernel.

To build a modular kernel, you simply select Enable Loadable Module Support on the Loadable Module Support menu when you run the make menuconfig utility. As a general rule, you should always enable loadable module support<sup>1</sup> so you can then decide as you select each kernel option whether it will be compiled into the kernel, by placing an asterisk (\*) next to it, or compiled as a module, by placing an M next to it.

<sup>1</sup> Packages such as VMware require loadable module support.

To continue our example of using the kernel source code supplied on the CD-ROM, enter the following lines from the top of the kernel source tree:

---

```
#cp /mnt/cdrom/chapter3/sample-config .config
#umount /mnt/cdrom
#make oldconfig
#make menuconfig3
```

---

<sup>3</sup> If the make menuconfig command complains about a clock skew problem, check to make sure your system time is correct, and use the hwclock command to force your hardware clock to match your system time. See the hwclock manual page for more information.

**NOTE** *The .config file placed into the top of the kernel source tree is hidden from the normal ls command. You must use ls -a to see it.*

The sample-config file used in the previous command will build a modular kernel, but you can change it to create a monolithic kernel by disabling loadable module support. To do so, deselect it on the Loadable Module Support submenu. Be sure to enter the Processor Type and Features submenu and select the correct type of processor for your system, because even a modular kernel only compiles for one type of processor (or family of processors). See the contents of the file /proc/cpuinfo to find out which CPU the kernel thinks your system is using. Also, you normally should *always* select SMP support even if your system has only one CPU.

**NOTE** *When building a monolithic kernel, you can reduce the size of the kernel by deselecting the kernel options not required for your hardware configuration. If you are currently running a modular kernel, you can see which modules are currently loaded on your system by using the lsmod command. Also see the output of the following commands to learn more about your current configuration:*

---

```
#lspci -v | less  
#dmesg | less  
#cat /proc/cpuinfo  
#uname -m
```

---

Once you have saved your kernel configuration file changes, you are ready to begin compiling your new kernel.

## Step 3: Compile the Code

To compile your kernel, you first need to convert the kernel source code files into object code with the gcc compiler, using the make utility.

If you are using the 2.4 kernel supplied with this book, start with the make dep command to build the kernel module dependencies. However, you can skip this step if you are compiling a kernel from the 2.6 series.

Also, you must clean the kernel source tree regardless of which version of the kernel you are using, whether you are building a modular or monolithic kernel, or whether you are patching, upgrading, or compiling a new kernel. Use the make clean command, which will clean the kernel source tree and remove any temporary files that were left behind by a previous kernel compilation.

Once the kernel source tree is clean, you are ready to tell the gcc compiler to build a new kernel with the make bzImage command; then, if you are building a modular kernel, you'll need to add the make modules and make modules\_install commands to build and install the kernel modules into the /lib/modules directory.

To store the lengthy output of the compilation commands in a file, precede the `make` commands with the `nohup` command, which automatically redirects output to a file named `nohup.out`. If you have difficulty compiling the kernel, you can enter each `make` command separately and examine the output of the `nohup.out` file after each command completes to try and locate the source of the problem. In the following example, all of the `make` commands are combined into one command to save keystrokes.

---

```
#cd /usr/src/linux
#nohup make dep clean bzImage modules modules_install
```

---

**NOTE** *If you are building a monolithic kernel, simply leave off the `modules modules_install` arguments, and if you are building a 2.6 kernel, leave off the `dep` argument.*

Make sure that the kernel compiles successfully by looking at the end of the `nohup.out` file with this command:

---

```
#tail nohup.out
```

---

If you see error messages at the end of this file, examine earlier lines in the file until you locate the start of the problem. If the `nohup.out` file does not end with an error message, you should now have a file called `bzImage` located in the `/usr/src/linux/arch/*/boot` (where `*` is the name of your hardware architecture). For example, if you are using Intel hardware the compiled kernel is now in the file `/usr/src/linux/arch/i386/boot/bzImage`. You are now ready to install your new kernel.

## Step 4: Install the Object Code and Configuration File

The `make modules_install` command has already done part of this installation step, placing all of the modular kernel executables into a `/lib/modules` subdirectory based on the version number of the kernel. Now you just need to install the main part of the kernel by copying the kernel executable from the kernel source tree to a directory where your boot loader can find it at system boot time. Because boot loaders normally look for the kernel in the `/boot` directory, we'll place a copy of the kernel executable there.

Enter this command to copy the kernel executable file you just compiled, named `bzImage`, to where your boot loader expects to find it when the system first starts:

---

```
#cp /usr/src/linux/arch/i386/boot/bzImage /boot/vmlinuz-<version>
```

---

Replace `<version>` in this example command with the version number of your kernel, and `i386` with the proper name of the hardware architecture you are using. (Remember to also add the `EXTRAVERSION` number if you specified one in the Makefile in Step 2.)

## WHEN MAKE FAILS

If the `nohup.out` file ends with an error, or if the `bzImage` file is not created, or if the `/lib/modules` subdirectory does not exist,<sup>1</sup> try copying different versions of the `.config` file into place or use the `make mrproper` and `make oldconfig` commands to build a new `.config` file from scratch.<sup>2</sup> Then rerun the `make` commands to compile the kernel again to see if the kernel will compile with a different `.config` file.

You should also try turning on SMP support for your kernel through the `make menuconfig` command (if you are not already using one of the SMP `.config` files from your distribution).

If the `nohup.out` file contains just a few lines of output, and you see error messages about the `.hdepend` file, try removing this temporary file (if you are using a 2.4 or earlier kernel), and then enter the `nohup make dep clean bzImage modules modules_install` command again. Or, if you cannot get a newer kernel to compile, try an older, more stable version from the 2.4 series.

If none of this helps, try entering each of the `make` commands separately: `make dep`, then `make clean`, then `make bzImage`, and so forth, and send the output of each command to the screen and to a separate log file:

---

```
#make dep 2>&1 | tee /tmp/dep.log
#make clean 2>&1 | tee /tmp/clean.log
#make bzImage 2>&1 | tee /tmp/bzimage.log
#make modules 2>&1 | tee /tmp/modules.log
#make modules_install _2>&1 | tee /tmp/modules_install.log
```

---

This should help you isolate the problem, at which point you can go online to find a solution. Try the `comp.os.linux` newsgroups (or `comp.os.linux.redhat`, dedicated to problems with Red Hat Linux). Also, see the Linux section at <http://marc.theaimsgroup.com>.

Before posting a kernel compilation question to one of these newsgroups, read the Linux FAQ (<http://www.tldp.org>), the Linux-kernel mailing list FAQ (<http://www.tux.org/lkml>), and the Linux-kernel HOWTO.

<sup>1</sup> The `/lib/modules` subdirectory should be created when building a modular kernel.

<sup>2</sup> This method should at least get the kernel to compile even if it doesn't have the options you want. You can then go back and use the `make menuconfig` command to set the options you need.

## Install the `System.map`

Your system may need a file called `System.map` when it loads a module to resolve any module dependencies (a module may require other modules for normal operation).<sup>4</sup>

Copy the `System.map` file into place with this command:

---

```
#cp /usr/src/linux/System.map /boot/System.map
```

---

<sup>4</sup> The `System.map` file is only consulted by the `modutils` package when the currently running kernel's symbols (in `/proc/ksyms`) don't match what is stored in the `/lib/modules` directory.

## **Save the Kernel Configuration File**

For documentation purposes, you can now save the configuration file used to build this kernel. Use the following command:

---

```
#cp /usr/src/linux/.config /boot/config-<version>
```

---

Again, *<version>* in this command should be replaced with the kernel version you have just built.

## **Step 5: Configure Your Boot Loader**

The previous four steps are used to install any program, but the Linux kernel executable is part of the operating system, so we have to do a few additional things to cause the computer to recognize the kernel at boot time.

To complete this step, you must know what type of boot loader your system is using. Newer Linux distributions use GRUB as the boot loader of choice, but many older distributions use the LILO boot loader. (Reboot your system and watch it boot if you are not sure—the type of boot loader you are using will be displayed during bootup.)

**NOTE** *If you are using a SCSI boot disk, see the “SCSI Gotchas” section later in the chapter.*

Both LILO and GRUB give you the ability at boot time to select which kernel to boot from, selecting from those listed in their configuration files. This is useful if your new kernel doesn’t work properly. Therefore, you should always *add* a reference to your new kernel in the boot loader configuration file and leave the references to the previous kernels as a precaution.

If you are using LILO, you’ll modify the */etc/lilo.conf* file; if you are using GRUB, you’ll modify the */boot/grub/grub.conf* file (or the */boot/grub/menu.lst* if you installed GRUB from source).

Add your new kernel to *lilo.conf* with an entry that looks like this:

---

```
image=/boot/vmlinuz-<version>
label=linux.orig
initrd=/boot/initrd-<version>.img
read-only
root=/dev/hda2
```

---

Add your new kernel to *grub.conf* with an entry that looks like this:

---

```
title My New Kernel (<version>)
root (hd0,0)
kernel /vmlinuz-<version> ro root=/dev/hda2
initrd /initrd-<version>.img
```

---

You do not need to do anything to inform the GRUB boot loader about the changes you make to the `grub.conf` configuration file, because GRUB knows how to locate files even before the kernel has loaded. It can find its configuration file at boot time. However, when using LILO, you'll need to enter the command `lilo -v` to install a new boot sector that tells LILO where the configuration file resides on the disk (because LILO doesn't know how to use the normal Linux filesystem at boot time).

## BOOT LOADERS AND ROOT FILESYSTEMS

The name `/` is given to the Linux root filesystem mount point. This mount point can be used to mount an entire disk drive or just a partition on a disk drive, though normally the root filesystem is a partition. (On Red Hat systems, Linux uses the second partition of the first disk device at boot time; the first partition is usually the `/boot` partition).

To find out which root partition or device your system is using, use the `df` command. For example, the following `df` command shows that the root filesystem is using the `/dev/hda2` partition:

```
# df
Filesystem      1K-blocks  Used Available Use% Mounted on
/dev/hda2        8484408  1195652  6857764  15% /
/dev/hda1        99043    9324    84605  10% /boot
```

When you modify the boot loader configuration file, you must tell the Linux kernel which device, or device and partition, to use for the root filesystem. When using LILO as your boot loader, you specify the root filesystem in the `lilo.conf` file with a line that looks like this:

---

```
root=/dev/hda2
```

---

The `grub.conf` configuration file entry looks like this:

---

```
kernel /vmlinuz-<version> ro root=/dev/hda2
```

---

Note, however, that the Red Hat kernel allows you to use a symbolic name that looks like this:

---

```
kernel /vmlinuz-<version> ro root=LABEL=/
```

---

Using the syntax `LABEL=/` to specify the root filesystem is only possible when using *Red Hat kernels*. If you compile the stock kernel, you must specify the root partition device name, such as `/dev/hda2`.

One final concern when specifying the root filesystem using GRUB is that the GRUB boot loader also has its own root filesystem. The GRUB root filesystem has nothing to do with the Linux root filesystem and is usually on a different partition (it is the `/boot` partition on Red Hat systems).

To tell GRUB which disk partition to use at boot time as its root filesystem, create an entry such as the following in the `grub.conf` configuration file:

---

```
root (hd0,0)
```

---

This tells GRUB to use the first partition of the first hard drive as the GRUB root filesystem. GRUB calls the first hard drive hd0 (whether it is SCSI or IDE), and partitions are numbered starting with 0. So to use the second partition of the first drive, the syntax would be as follows:

---

```
root (hd0,1)
```

---

Note: see the `/boot/grub/device.map` file for the list of devices on your system.<sup>1</sup>

Again, this is the root filesystem used by GRUB when the system first boots, and it is *not* related to the root filesystem used by the Linux kernel after the system is up and running. To list all of the partitions on a device, you can use the command `sfdisk -l`.

---

<sup>1</sup> The `/boot/grub/device.map` file is created by the installation program anaconda on Red Hat systems.

Now you are ready to reboot and test the new kernel.

Enter `reboot` at the command line, and watch closely as the system boots.

If the system hangs, you may have selected the wrong processor type. If the system fails to mount the root drive, you may have installed the wrong filesystem support in your kernel. (If you are using a SCSI system, see the “SCSI Gotchas” section for more information.)

If the system boots but does not have access to all of its devices, review the kernel options you specified in step 2 of this chapter.

Once you can access all of the devices connected to the system, and you can perform normal network tasks, you are ready to test your application or applications.

## SCSI GOTCHAS

If you are using a SCSI disk to boot your system, you may want to avoid some complication and compile support for your SCSI driver directly into the kernel, instead of loading it as a module. If you do this on a kernel version prior to version 2.6, you will save yourself the trouble of building the initial RAM disk. However, if you are building a modular kernel from version 2.4 or an earlier series, you will still need to build an initial RAM disk so the Linux kernel can boot up far enough to load the proper module to communicate with your SCSI disk drive. (See the man page for the `mkinitrd` Red Hat utility for the easiest way to do this on Red Hat systems. The `mkinitrd` command automates the process of building an initial RAM disk.) The initial RAM disk, or `initrd` file, must then be referenced in your boot loader configuration file so that the system knows how to mount your SCSI boot device before the modular kernel runs.

Also, if you *recompile* your 2.4 modular kernel without changing the `EXTRAVERSION` number in the `Makefile`, you may accidentally overwrite your existing `/lib/modules` libraries. If this happens, you may see the error “All of your loopback devices are in use!” when you run the `mkinitrd` command. If you see this error message, you must rebuild the kernel (before rebooting the system!) and add the SCSI drivers into the kernel. Do not attempt to build the drivers as modules and skip the `mkinitrd` process altogether.

## In Conclusion

You gain more control over your operating system when you know how to configure and compile your own kernel. You can selectively enable and disable features of the kernel to improve security, enhance performance, support hardware devices, or use the Linux operating system for special purposes. The Linux Virtual Server (LVS) kernel feature, for example, is required to build a Linux Enterprise Cluster. While you can use the pre-compiled kernel included with your Linux distribution, you'll gain confidence in your skills as a system administrator and a more in-depth knowledge of the operating system when you learn how to configure and compile your own kernel. This confidence and knowledge will help to increase system availability when problems arise after the kernel goes into production.



# PART II

## **HIGH AVAILABILITY**





# 4

## **SYNCHRONIZING SERVERS WITH RSYNC AND SSH**



This chapter describes one method of automating the copying of data and configuration files from one server to another. In its simplest form, synchronizing the data on two (or more) servers is just a matter of copying files from one server to another. One server acts as a primary repository for data, and changes to the data can only be made on this server (in a high-availability configuration, only one server owns a resource at any given point in time). A regularly scheduled copy utility then sends the data after it has been changed on the primary server to the backup server so it is ready to take ownership of the resource if the primary server crashes.

In a cluster configuration all nodes need to access and modify shared data (all cluster nodes offer the same services), so you will probably not use this method of data synchronization on the nodes inside the cluster. You can,

however, use the method of data synchronization described in this chapter on highly available server pairs to copy data and configuration files that change infrequently.

## rsync

The open source software package rsync that ships with the Red Hat distribution allows you to copy data from one server to another over a normal network connection. rsync is more than a replacement for a simple file copy utility, however, because it adds the following features:

- Examines the source files and only copies blocks of data that change.<sup>1</sup>
- (Optionally) works with the secure shell to encrypt data before it passes over the network.
- Allows you to compress the data before sending it over the network.
- Will automatically remove files on the destination system when they are removed from the source system.
- Allows you to throttle the data transfer speed for WAN connections.
- Has the ability to copy device files (which is one of the capabilities that enables system cloning as described in the next chapter).

Online documentation for the rsync project is available at <http://samba.anu.edu.au/rsync> (and on your system by typing `man rsync`).

**NOTE** *rsync must (as of version 2.4.6) run on a system with enough memory to hold about 100 bytes of data per file being transferred.*

rsync can either push or pull files from the other computer (both must have the rsync software installed on them). In this chapter, we will push files from one server to another in order to synchronize their data. In Chapter 5, however, rsync is used to pull data off of a server.

**NOTE** *The Unison software package can synchronize files on two hosts even when changes are made on both hosts. See the Unison project at <http://www.cis.upenn.edu/~bcpierce/unison>, and also see the Csync2 project for synchronizing multiple servers at <http://oss.linbit.com>.*

Because we want to automate the process of sending data (using a regularly scheduled cron job) we also need a secure method of pushing files to the backup server without typing in a password each time we run rsync. Fortunately, we can accomplish this using the features of the Secure Shell (SSH) program.

---

<sup>1</sup> Actually, rsync does a better job of reducing the time and network load required to synchronize data because it creates signatures for each block and only passes these block signatures back and forth to decide which data needs to be copied.

## Open SSH 2 and rsync

The rsync configuration described in this chapter will send all data through an Open Secure Shell 2 (henceforth referred to as SSH) connection to the backup server. This will allow us to encrypt all data before it is sent over the network and use a security encryption key to authenticate the remote request (instead of relying on the plaintext passwords in the `/etc/passwd` file).

A diagram of this configuration is shown in Figure 4-1.

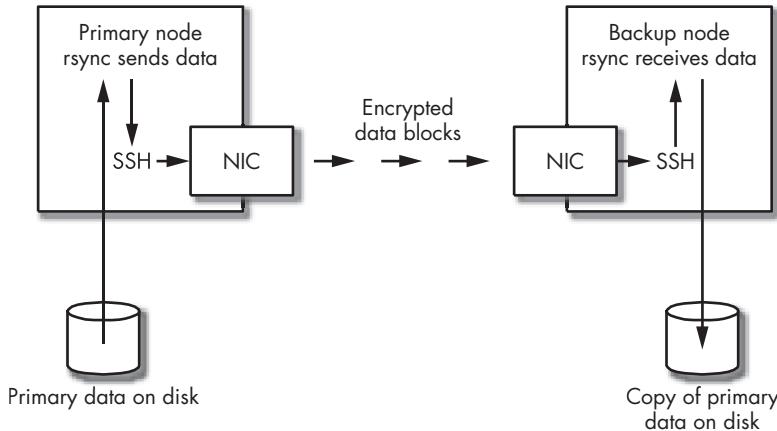


Figure 4-1: rsync and SSH

The arrows in Figure 4-1 depict the path the data will take as it moves from the disk drive on the primary server to the disk drive on the backup server. When rsync runs on the primary server and sends a list of files that are candidates for synchronization, it also exchanges data block signatures so the backup server can figure out exactly which data changed, and therefore which data should be sent over the network.

### SSH: A Secure Data Transport

Because we will push data from the primary server to the backup server, we will configure the primary server as an SSH client and then turn the backup server into an SSH server capable of accepting this data. The SSH connection between these two servers will need to enforce the following security configuration:

- When the primary server initiates a connection to the backup server, it should verify that it is in fact talking to the true backup server (and not a bogus host created by a hacker trying to hijack our data).
- The backup server should only accept connections from the primary server. In fact, the backup server should only accept connections from one trusted account on the primary server.
- All data sent over the network connection should be encrypted.

Using these security features, our data replication method can also be used on an untrusted public network such as the Internet.

This security configuration will also need to be fully automated (we should never need to walk over to the system console and enter a password to get the SSH connection working), and it should survive system reboots.

We can accomplish all of these security configuration goals using SSH encryption keys that allow us to establish a two-way trust relationship between the client and the server.

## **SSH Encryption Keys**

Encryption keys, or asymmetric encryption keys (to be exact), always have two halves: a *public half* and a *private half*. The private half should be zealously guarded and never transmitted over the network. The public half should be given to anyone who wants, or needs, to send encrypted data. For SSH to work properly, we need to configure two types of encryption keys: the SSH host encryption key and the user encryption key.

### **The Host Encryption Key**

The host encryption key is configured once for each SSH server and stored in a file on a locally attached disk drive. Red Hat stores the private and public halves of this key in separate files:<sup>2</sup>

---

```
/etc/ssh/ssh_host_rsa_key  
/etc/ssh/ssh_host_rsa_key.pub
```

---

This key, or these key files, are created the first time the `/etc/rc.d/init.d/ssh` script is run on your Red Hat system. However, we really only need to create this key on the backup server, because the backup server is the SSH server.

### **The User Encryption Key**

This key is created on the primary server manually by running the `ssh-keygen` utility. This key is stored in two files in the user's home directory in the `.ssh` subdirectory. For user `web` with a home directory of `/home/web`, the key would be stored in the following two files:

---

```
/home/web/.ssh/id_rsa  
/home/web/.ssh/id_rsa.pub
```

---

The public half of this key is then copied to the backup server (the SSH server) so the SSH server will know this user can be trusted.

---

<sup>2</sup> You will also see DSA files in the `/etc/ssh` directory. The Digital Signature Algorithm, however, has a suspicious history and may contain a security hole. See *SSH, The Secure Shell: The Definitive Guide*, by Daniel J. Barrett and Richard E. Silverman (O'Reilly and Associates).

## Session Keys

As soon as the client decides to trust the server, they establish yet another encryption key: the *session key*. As its name implies, this key is valid for only one session between the client and the server, and it is used to encrypt all network traffic that passes between the two machines. The SSH client and the SSH server create a session key based upon a secret they both share—called the *shared secret*. You should never need to do anything to administer the session key, because it is generated automatically by the running SSH client and SSH server (information used to generate the session key is stored on disk in the `/etc/ssh/primes` file).

Establishing a two-way trust relationship between the SSH client and the SSH server is a two-step process. In the first step, the client figures out if the server is trustworthy and what session key it should use. Once this is done, the two machines establish a secure SSH transport.

The second step occurs when the server decides it should trust the client.

## ***Establishing the Two-Way Trust Relationship: Part 1***

Let's examine how this two-way trust relationship is established in more detail. First, the client must decide whether it can trust the server; then, the server must decide whether it can trust the client. So Part 1 of establishing the two-way trust relationship is when the client computer determines whether it is talking to the proper server.

### **Should the SSH Client Trust the SSH Server?**

The first step in building a secure SSH transport is initiated by the client. The client needs to know if it is talking to the right server (no one has hijacked the IP address or hostname of our SSH server). This is accomplished during the SSH key exchange processes.

**NOTE** *If you need to reuse an IP address on a different machine and retrain the security configuration you have established for SSH, you can do so by copying the SSH key files from the old machine to the new machine.*

### **The SSH Key Exchange Process**

The SSH key exchange process starts when an SSH client sends a connection request on port 22 to the SSH server (as shown in Figure 4-2).

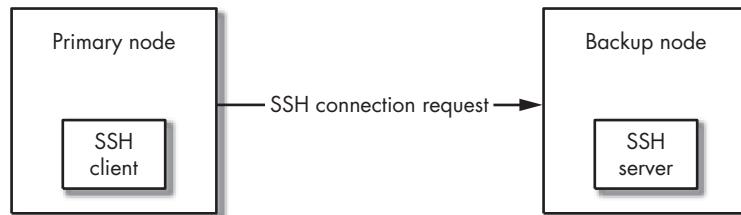


Figure 4-2: SSH client connection request

Once the server and the client make some agreements about the SSH protocol, or version, they will use to talk to each other (SSH version 1 or SSH version 2), they begin a process of synchronized mathematical gymnastics that only requires a minimum amount of data to pass over the network. The goal of these mathematical gymnastics is to send only enough information for the client and server to agree upon a shared secret for this SSH session without sending enough information over the network for an attacker to guess the shared secret. Currently, this is called the *diffie-hellman-group1-sha1 key exchange process*.

The last thing the SSH server sends back to the client at the end of the diffie-hellman-group1-sha1 key exchange process is the final piece of data the client needs to figure out the shared secret along with the public half of its host encryption key (as shown in Figure 4-3).

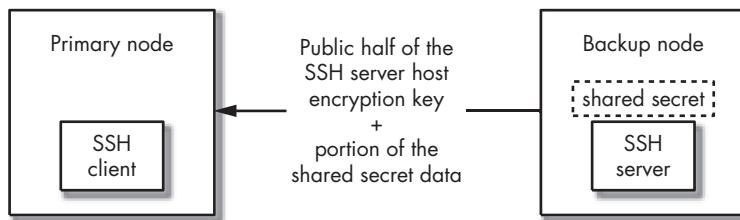


Figure 4-3: SSH server response

**NOTE** *The SSH server never sends the entire SSH session key or even the entire shared secret over the network, making it nearly impossible for a network-snooping hacker to guess the session key.*

At this point, communication between the SSH client and the SSH servers stops while the SSH client decides whether or not it should trust the SSH server. The SSH client does this by looking up the server's host name in a database of host name-to-host-encryption-key mappings called `known_hosts2` (referred to as the *known\_hosts database* in this chapter<sup>3</sup>). See Figure 4-4. If the encryption key and host name are not in the `known_hosts` database, the client displays a prompt on the screen and asks the user if the server should be trusted.

If the host name of the SSH server is in the `known_hosts` database, but the public half of the host encryption key does not match the one in the database, the SSH client does not allow the connection to continue and complains.

<sup>3</sup> Technically, the `known_hosts` database was used in Open SSH version 1 and renamed `known_hosts2` in Open SSH version 2. Because this chapter only discusses Open SSH version 2, we will not bother with this distinction in order to avoid any further cumbersome explanations like this one.

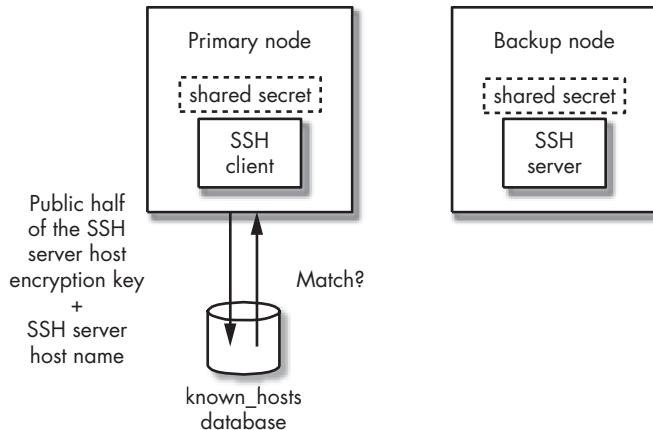


Figure 4-4: The SSH client searches for the SSH server name in its `known_hosts` database

---

```

@oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
@      WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!      @
@oooooooooooooooooooooooooooooooooooooooooooooooooooooooooooo
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that the RSA host key has just been changed.
The fingerprint4 for the RSA key sent by the remote host is
6e:36:4d:03:ec:2c:6b:fe:33:c1:e3:09:fc:aa:dc:0e.
Please contact your system administrator.
Add correct host key in /home/web/.ssh/known_hosts2 to get rid of this
message.
Offending key in /home/web/.ssh/known_hosts2:1
RSA host key for 10.1.1.2 has changed and you have requested strict checking.

```

---

(If the host key on the SSH server *has* changed, you will need to remove the offending key from the `known_hosts` database file with a text editor such as vi.)

Once the user decides to trust the SSH server, the SSH connection will be made automatically for all future connections so long as the public half of the SSH server's host encryption key stored in its `known_hosts` database doesn't change.

The client then performs a final mathematical feat using this public half of the SSH server host encryption key (against the final piece of shared secret key data sent from the server) and confirms that this data could only come from a server that knew the private half of the public host encryption key it received from the SSH server. Once the client has successfully done this, the client and the server are said to have established a secure SSH transport (see Figure 4-5).

---

<sup>4</sup> You can verify the “fingerprint” on the SSH server by running the following command:  
`#ssh-keygen -l -f /etc/ssh/ssh_host_rsa_key.pub`

Now, all information sent back and forth between the client and the server can be encrypted with the session key (the encryption key that was derived on both the client and the server from the shared secret).

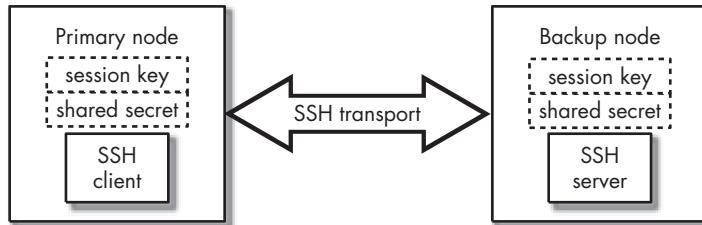


Figure 4-5: The SSH client and SSH server establish an SSH transport

## ***Establishing the Two-Way Trust Relationship: Part 2***

We've described half of the process for establishing the two-way trust relationship between the client and the server. Now, for Part 2, we need to discuss the other half, which takes place on the SSH server.

### **Can the Server Trust the Client?**

We have solved only half the problem of a two-way trust between the client and the server; the client trusts the server—now the server needs to decide whether it should trust the client.

This can be accomplished using either *hostbased authentication* or *user authentication*.

#### ***Hostbased Authentication***

Historically, hostbased authentication was accomplished using nothing more than the client's source IP address or host name. Crackers could easily break into servers that used this method of trust by renaming their server to match one of the acceptable host names, by spoofing an IP address, or by circumventing the Domain Name System (DNS) used to resolve host names into IP addresses.

System administrators, however, are still tempted to use hostbased authentication because it is easy to configure the server to trust any account (even the root account) on the client machine. Unfortunately, even with the improvements SSH has introduced, hostbased authentication is still cumbersome and weak when compared to the other methods that have evolved, and it should not be used.

#### ***User Authentication***

When using user authentication, the SSH server does not care which host the SSH connection request comes from, so long as the user making the connection is a trusted user.

SSH can use the user's normal password and account name for authentication over the secure SSH transport. If the user has the same account and password on the client and server, the SSH server knows this user is trustworthy. The main drawback to this method, however, is that we need to enter the user's password directly into our shell script on the SSH client to automate connections to the SSH server. So this method should also not be used. (Recall that we want to automate data synchronization between the primary and backup servers.)

### ***User Authentication Using SSH Encryption Keys***

Fortunately, a method that is better than either of these two methods is available using SSH encryption keys. All we have to do is store the public half of a user's encryption key on the SSH server. When we try to connect to the SSH server from the SSH client, the server will send us a challenge (a random number encrypted with the public half of the user encryption key), and only a client that knows the private half of the encryption key will be able to decrypt this challenge.

The private half of the user's encryption key on the SSH client is normally also protected by a *passphrase*. A passphrase is just a long password (it can contain spaces) selected by the user that is used to encrypt the private half of the user's real encryption key before it is stored on disk. We will not use a passphrase, however, because the passphrase would need to be added to our shell script to automate it.

**NOTE** *You can instead use the Linux Intrusion Detection System, or LIDS, to deny anyone (even root) from accessing the private half of our user encryption key that is stored on the disk drive. See also the ssh-agent man page for a description of the ssh-agent daemon.*

Here is a simple two-node SSH client-server recipe.

## **Two-Node SSH Client-Server Recipe**

List of ingredients:

- Primary server (preferably with two Network Interface Cards and cables)
- Backup server (preferably with two Network Interface Cards and cables)
- Crossover cable or mini hub
- Copy of the rsync RPM file
- Copy of Open SSH (version 2.2 or later)

Included on the CD-ROM with this book is a copy of the Open SSH package and the rsync package in RPM format in the chapter4 subdirectory. Your distribution should already contain Open SSH. To find out which version is loaded on your system, type:

---

```
#rpm -qa | grep ssh
```

---

If you do not have version 2.2 or later, you will need to install a newer version either from your distribution vendor or from the CD-ROM included with this book before starting this recipe.

**NOTE** *Please check for the latest versions of these software packages from your distribution or from the project home pages so you can be sure you have the most recent security updates for these packages.*

### ***Create a User Account That Will Own the Data Files***

Perform this step on the SSH client and the SSH server. The SSH client in this recipe is the primary server at IP address 10.1.1.1. The SSH server is the backup server at IP address 10.1.1.2 (see Figure 4-1).

You can use the root account or any user account (you can create a special non-root account dedicated to providing SSH access). You can then grant this user access to commands or scripts as the root user using the sudo program (see `man sudo` for more information). But if you want to take a shortcut and just use the root account, you will need to modify the default SSHD config file (`/etc/ssh/sshd_config`) that comes with the Red Hat distribution and change the `PermitRootLogin` variable so it looks like this:

---

```
PermitRootLogin without-password
```

---

In this recipe, we will use the more secure technique of creating a new account called `web` and then giving access to the files we want to transfer to the backup node (in the directory `/www`) to this new account; however, for most high-availability server configurations, you will need to use the `root` account or the `sudo` program.

1. Add the `web` user account with the following command:

---

```
#useradd -u 100 -g 48 -d /home/web -s /bin/bash -c "Web File Owner" -p  
passw0rd web
```

---

**NOTE** *For added security, we will (later in this recipe) remove the shell and password for this account in order to prevent normal logons as user `web`.*

The `useradd` command automatically created a `/home/web` directory and gave ownership to the `web` user, and it set the password for the `web` account to `passw0rd`.

2. Now create the `/www` directory and place a file or two in it:

---

```
#mkdir /www  
#mkdir /www/htdocs  
#echo "This is a test" > /www/htdocs/index.html  
#chown -R web /www
```

---

These commands create the `/www/htdocs` directory and subdirectory and place the file `index.html` with the words `This is a test` in it. Ownership of all of these files and directories is then given to the `web` user account we just created.

## **Configure the Open SSH2 Host Key on the SSH Server**

If you installed the `sshd` daemon on your system using the Red Hat RPM file, then all you need to do to create your SSH host keys on the backup server is start the `sshd` daemon. Check to see if it is already running on the backup server with the command:

---

```
#ps -elf | grep sshd
```

---

If it is not running, start it with the command:

---

```
#/etc/init.d/sshd start
```

---

or

---

```
#service sshd start
```

---

**NOTE** *If `sshd` was not running, you should make sure it starts each time the system boots—see Chapter 1.*

Use the following two commands to check to make sure the Open SSH version 2 (RSA) host key has been created. These commands will show you the public and private halves of the RSA host key created by the `sshd` daemon.

The private half of the Open SSH2 RSA host key:

---

```
#cat /etc/ssh/ssh_host_rsa_key
```

---

The public half of the Open SSH2 RSA host key:

---

```
#cat /etc/ssh/ssh_host_rsa_key.pub
```

---

**NOTE** *The `/etc/rc.d/init.d/sshd` script on Red Hat systems will also create Open SSH1 DSA host keys (that we will not use) with similar names.*

## **Create a User Encryption Key for This New Account on the SSH Client**

To create a user encryption key for this new account:

1. Log on to the SSH client as the `web` user you just created. If you are logged on as `root`, use the following command to switch to the `web` account:

---

```
#su - web
```

---

2. Make sure you are the web user with the command:

---

```
#whoami
```

---

3. Now create an Open SSH2 key with the command:

---

```
#ssh-keygen -t rsa
```

---

The output of this command will look like this:

---

```
Generating public/private rsa key pair.  
Enter file in which to save the key (/home/web/.ssh/id_rsa):  
Enter passphrase (empty for no passphrase): <return> ←  
Enter same passphrase again: <return> ←  
Your identification has been saved in /home/web/.ssh/id_rsa.  
Your public key has been saved in /home/web/.ssh/id_rsa.pub.  
The key fingerprint is:  
a3:ec:9f:63:52:4e:42:66:11:6a:38:8d:de:5e:66:84 web@primary.mydomain.com
```

---

Leave the lines marked with arrows (←) blank.

**NOTE** *We do not want to use a passphrase to protect our encryption key, because the SSH program would need to prompt us for this passphrase each time a new SSH connection was needed.<sup>5</sup>*

As the output of the above command indicates, the public half of our encryption key is now stored in the file:

---

```
/home/web/.ssh/id_rsa.pub
```

---

4. View the contents of this file with the command:

---

```
#cat /home/web/.ssh/id_rsa.pub
```

---

You should see something like this:

---

```
ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAQEAz1LUbTSq/  
45jE8Jy8lGAX4e6Mxu33HVRUVfrn0U9+n1b4DpiHGWR1DXZ15dZQe26pITxSJqFabTD47uGwVD  
0Hi+TxczL1lasdf9124jsleHmgg5uvgn3h9pTAkuS06KEZBSOUvlsAEvUodEhAUv7FBgqfN2VD  
mrV19uRRnm3OnXYE= web@primary.mydomain.com
```

---

**NOTE** *This entire file is just one line.*

---

<sup>5</sup> If you don't mind entering a passphrase each time your system boots, you can set a passphrase here and configure the ssh-agent for the web account. See the ssh-agent man page.

## ***Copy the Public Half of the User Encryption Key from the SSH Client to the SSH Server***

Because we are using user authentication with encryption keys to establish the trust relationship between the server and the client computer, we need to place the public half of the user’s encryption key on the SSH server. Again, you can use the `web` user account we have created for this purpose, or you can simply use the `root` account.

1. Use the copy-and-paste feature of your terminal emulation software (or use the mouse at one of the system consoles)—even if you are just typing at a plain-text terminal, this will work on Red Hat—and copy the above line (from the `id_rsa.pub` file on the SSH client) into the paste buffer.<sup>6</sup>
2. From another window, log on to the SSH server and enter the following commands:

---

```
#mkdir /home/web/.ssh  
#vi /home/web/.ssh/authorized_keys2
```

---

This screen is now ready for you to paste in the private half of the `web` user’s encryption key.

3. Before doing so, however, enter the following:

---

```
from="10.1.1.1",no-X11-forwarding,no-agent-forwarding, ssh-dss
```

---

This adds even more restriction to the SSH transport.<sup>7</sup> The `web` user is only allowed in from IP address `10.1.1.1`, and no X11 or agent forwarding (sending packets through the SSH tunnel on to other daemons on the SSH server) is allowed. (In a moment, we’ll even disable interactive terminal access.)

4. Now, without pressing ENTER, paste in the public half of the `web` user’s encryption key (on the same line). This file should now be one very long line that looks similar to this:

---

```
from="10.1.1.1",no-port-forwarding,no-X11-forwarding,no-agent-forwarding,  
ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAIEAz1LUbTSq/  
45jE8Jy8lGAX4e6Mxu33Hvfrn0U9+nlb4DpiHgWR1DXZ15dZQe26pITxSJqFabTD47uGWVD  
0Hi+TxczL1lasdf9124jsleHmgg5uvgn3h9pTAkuS06KEZBSOUvlAEvUodEhAUv7FBgqfN2VD  
mrV19uRRnm3OnXYE= web@primary.mydomain.com
```

---

<sup>6</sup> Security experts cringe at the thought that you might try and do this paste operation via telnet over the Internet because a hacker could easily intercept this transmission and steal your encryption key.

<sup>7</sup> An additional option that could be used here is `no-port-forwarding` which, in this case, could mean that the `web` user on the SSH client machine cannot use the SSH transport to pass IP traffic destined for another UDP or IP port on the SSH server. However, if you plan to use the SystemImager software described in the next chapter, you need to leave port forwarding enabled on the SystemImager Golden Client.

5. Set the security of this file:

---

```
#chmod 600 /home/web/.ssh/authorized_keys2
```

---

**NOTE** *See man sshd and search for the section titled AUTHORIZED\_KEYS FILE FORMAT for more information about this file.*

You should also make sure the security of the /home/web and the /home/web/.ssh directories is set properly (both directories should be owned by the web user, and no other user needs to have write access to either directory).

The vi editor can combine multiple lines into one using the keys SHIFT+J. However, this method is prone to errors, and it is better to use a telnet copy-and-paste utility (or even better, copy the file using the ftp utility or use the scp command and manually enter the user account password).

### ***Test the Connection from the SSH Client to the SSH Server***

Log on to the SSH client as the web user again, and enter the following command:

---

```
#ssh 10.1.1.2 hostname
```

---

Because this is the first time you have connected to this SSH server, the public half of the SSH server's host encryption key is not in the SSH client's known\_hosts database. You should see a warning message like the following:

---

```
The authenticity of host '10.1.1.2 (10.1.1.2)' can't be established.  
RSA key fingerprint is a4:76:aa:ed:bc:2e:78:0b:4a:86:d9:aa:37:bb:2c:93.  
Are you sure you want to continue connecting (yes/no)?
```

---

When you type yes and press ENTER, the SSH client adds the public half of the SSH server's host encryption key to the known\_hosts database and displays:

---

```
Warning: Permanently added '10.1.1.2' (RSA) to the list of known hosts.
```

---

**NOTE** *To avoid this prompt, we could have copied the public half of the SSH server's host encryption key into the known\_hosts database on the SSH client.*

The SSH client should now complete your command. In this case, we asked it to execute the command `hostname` on the remote server, so you should now see the name of the SSH server on your screen and control returns to your local shell on the SSH client.

A new file should have been created (if it did not already exist) on the SSH client called `/home/web/.ssh/known_hosts2`.

Instead of the remote host name, your SSH client may prompt you for a password with a display such as the following:

---

```
web@10.1.1.2's password:
```

---

## AVOIDING SSH VERSION 1

If you see (RSA1) instead of (RSA) in the above message, this means you have fallen back to SSH-level security instead of SSH2, and you will need to do the following (you will also need to do this if you see a complaint about the fact that you might be doing something NASTY):

On the SSH client, remove the known\_hosts file that was just created:

---

```
#rm /home/web/.ssh/known_hosts
```

---

On the SSH server, check the file /etc/ssh/sshd\_conf and make sure the Protocol entry looks like this:

---

```
Protocol 2,1
```

---

This indicates that we want the SSHD server to first attempt to use the SSH2 protocol and only if this doesn't work should it fall back to the SSH1 version of the protocol. Now restart the SSHD daemon on the SSH server with the command:

---

```
#/etc/rc.d/init.d/sshd restart
```

---

Now try the ssh 10.1.1.2 hostname command again to see if you can log on to the SSH server (at IP address 10.1.1.2 in this example).

If you receive the message Connection refused, this might indicate that the sshd daemon on the server is listening to the wrong network interface card or not listening at all (or you may have ipchains/iptables firewall rules blocking access to the sshd server). To see if sshd is listening on the server, run the command netstat -apn | grep ssh. You should see that sshd is listening on port 22. To check or modify which interface sshd is listening on (again on the server), examine the ListenAddress line in the /etc/ssh/sshd\_config file.

This means the SSH server was not able to authenticate your web user account with the public half of the user encryption key you pasted into the /home/web/.ssh/authorized\_keys2 file. (Check to make sure you followed all of the instructions in the last section.)

If this doesn't help, try the following command to see what SSH is doing behind the scenes:

---

```
#ssh -v 10.1.1.2 hostname
```

---

The output of this command should display a listing similar to the following:

---

```
OpenSSH_2.5.2p2, SSH protocols 1.5/2.0, OpenSSL 0x0090600f
debug1: Seeding random number generator
debug1: Rhosts Authentication disabled, originating port will not be trusted.
debug1: ssh_connect: getuid 100 geteuid 0 anon 1
debug1: Connecting to 10.1.1.2 [10.1.1.2] port 22.
debug1: Connection established.
debug1: unknown identity file /home/web/.ssh/identity
```

---

```
debug1: identity file /home/web/.ssh/identity type -1
debug1: unknown identity file /home/web/.ssh/id_rsa
debug1: identity file /home/web/.ssh/id_rsa type -1
debug1: identity file /home/web/.ssh/id_dsa type 2
debug1: Remote protocol version 1.99, remote software version OpenSSH_2.5.2p2
debug1: match: OpenSSH_2.5.2p2 pat ^OpenSSH
Enabling compatibility mode for protocol 2.0
debug1: Local version string SSH-2.0-OpenSSH_2.5.2p2
debug1: send KEXINIT
debug1: done
debug1: wait KEXINIT
debug1: got kexinit: diffie-hellman-group-exchange-sha1,diffie-hellman-group1-
sha1
debug1: got kexinit: ssh-rsa,ssh-dss
debug1: got kexinit:
aes128-cbc,3des-cbc,blowfish-cbc,cast128-cbc,arcfour,aes192-cbc,aes256-
cbc,rijndael128-cbc,rijndael192-cbc,rijndael256-cbc,rijndael-
cbc@lysator.liu.se
debug1: got kexinit:
aes128-cbc,3des-cbc,blowfish-cbc,cast128-cbc,arcfour,aes192-cbc,aes256-
cbc,rijndael128-cbc,rijndael192-cbc,rijndael256-cbc,rijndael-
cbc@lysator.liu.se
debug1: got kexinit: hmac-md5,hmac-sha1,hmac-ripemd160,hmac-
ripemd160@openssh.com,hmac-sha1-96,hmac-md5-96
debug1: got kexinit: hmac-md5,hmac-sha1,hmac-ripemd160,hmac-
ripemd160@openssh.com,hmac-sha1-96,hmac-md5-96
debug1: got kexinit: none,zlib
debug1: got kexinit: none,zlib
debug1: got kexinit:
debug1: got kexinit:
debug1: first kex follow: 0
debug1: reserved: 0
debug1: done
debug1: kex: server->client aes128-cbc hmac-md5 none
debug1: kex: client->server aes128-cbc hmac-md5 none
debug1: Sending SSH2_MSG_KEX_DH_GEX_REQUEST.
debug1: Wait SSH2_MSG_KEX_DH_GEX_GROUP.
debug1: Got SSH2_MSG_KEX_DH_GEX_GROUP.
debug1: dh_gen_key: priv key bits set: 129/256
debug1: bits set: 1022/2049
debug1: Sending SSH2_MSG_KEX_DH_GEX_INIT.
debug1: Wait SSH2_MSG_KEX_DH_GEX_REPLY.
debug1: Got SSH2_MSG_KEXDH_REPLY.
debug1: Host '10.1.1.2' is known and matches the RSA host key.
debug1: Found key in /home/web/.ssh/known_hosts2:1
debug1: bits set: 1025/2049
debug1: ssh_rsa_verify: signature correct
debug1: Wait SSH2_MSG_NEWKEYS.
debug1: GOT SSH2_MSG_NEWKEYS.
debug1: send SSH2_MSG_NEWKEYS.
```

```
debug1: done: send SSH2_MSG_NEKEYS.
debug1: done: KEX2.
debug1: send SSH2_MSG_SERVICE_REQUEST
debug1: service_accept: ssh-userauth
debug1: got SSH2_MSG_SERVICE_ACCEPT
debug1: authentications that can continue: publickey,password
debug1: next auth method to try is publickey
debug1: try pubkey: /home/web/.ssh/identity
debug1: try pubkey: /home/web/.ssh/id_dsa
debug1: try pubkey: /home/web/.ssh/id_rsa
debug1: Remote: Port forwarding disabled.
debug1: Remote: X11 forwarding disabled.
debug1: Remote: Agent forwarding disabled.
debug1: Remote: Pty allocation disabled.
debug1: input_userauth_pk_ok: pkalg ssh-dss blen 433 lastkey 0x8083a58 hint 2
debug1: read SSH2 private key done: name rsa w/o comment success 1
debug1: sig size 20 20
debug1: Remote: Port forwarding disabled.
debug1: Remote: X11 forwarding disabled.
debug1: Remote: Agent forwarding disabled.
debug1: Remote: Pty allocation disabled.
debug1: ssh-userauth2 successful: method publickey
debug1: fd 5 setting O_NONBLOCK
debug1: fd 6 IS O_NONBLOCK
debug1: channel 0: new [client-session]
debug1: send channel open 0
debug1: Entering interactive session.
debug1: client_init id 0 arg 0
debug1: Sending command: hostname
debug1: channel 0: open confirm rwindow 0 rmax 16384
debug1: client_input_channel_req: channel 0 rtype exit-status reply 0
debug1: channel 0: rcvd eof
debug1: channel 0: output open -> drain
debug1: channel 0: rcvd close
debug1: channel 0: input open -> closed
debug1: channel 0: close_read
bottompc.test.com
debug1: channel 0: obuf empty
debug1: channel 0: output drain -> closed
debug1: channel 0: close_write
debug1: channel 0: send close
debug1: channel 0: is dead
debug1: channel_free: channel 0: status: The following connections are open:
#0 client-session (t4 r0 i8/0 o128/0 fd -1/-1)
debug1: Transferred: stdin 0, stdout 0, stderr 0 bytes in 0.0 seconds
debug1: Bytes per second: stdin 0.0, stdout 0.0, stderr 0.0
debug1: Exit status 0
```

---

Notice the lines that are bold in this listing, and compare this output to yours.

If you are prompted for the password and you are using the root account, you did not set `PermitRootLogin yes` in the `/etc/ssh/sshd_config` file on the server you are trying to connect to. If you did not set the IP address correctly in `authorized_keys2` file on the server, you may see an error such as the following:

---

```
debug1: Remote: Your host '10.1.1.1' is not permitted to use this key for login.
```

---

If you see this error, you need to specify the correct IP address in the `authorized_keys2` file on the server you are trying to connect to. (You'll find this file is in the `.ssh` subdirectory under the home directory of the user you are using.)

Further troubleshooting information and error messages are also available in the `/var/log/messages` file and the `/var/log/secure` files.

## ***Using the Secure Data Transport***

Now that we have established a secure data transport between the SSH client and the SSH server, we can copy files, send commands, or create an interactive shell.

- To copy files, use the `scp` command. Here is an example command to copy the file `/etc/hosts` on the SSH client to `/tmp/hosts` on the SSH server:

---

```
#scp /etc/hosts 10.1.1.2:/tmp/hosts
```

---

- To send commands to the SSH server from the ssh client, use a command such as the following:

---

```
#ssh 10.1.1.2 who
```

---

This executes the `who` command on the `10.1.1.2` machine (the SSH server).

- To start an interactive shell on the SSH server, enter the following command on the SSH client:

---

```
#ssh 10.1.1.2
```

---

This logs you on to the SSH server and provides you with a normal, interactive shell. When you enter commands, they are executed on the SSH server. When you are done, type `exit` to return to your shell on the SSH client.

## ***Improved Security***

Once you have tested this configuration and are satisfied that the SSH client can connect to the SSH server, you can improve the security of your SSH client and SSH server. To do so, perform the following steps.

1. Remove the interactive shell access for the `web` account on the SSH server (the backup server) by editing the `authorized_keys2` file and add the `no-pty` option:

---

```
#vi /home/web/.ssh/authorized_keys2
```

---

2. Change the beginning of this one-line file to look like this:

---

```
from="10.1.1.1",no-X11-forwarding,no-agent-forwarding ssh-dss, no-pty
```

---

3. Once you have finished with the `rsync` recipe shown later in this chapter, you can deactivate the normal user account password on the backup server (the SSH server) with the following command:

---

```
#usermod -L web
```

---

This command inserts an `!` in front of the `web` user's encrypted password in the `/etc/shadow` file. (Don't use this command on the `root` account!)

### ***rsync over SSH***

Your distribution should already have the `rsync` program. Check to make sure it is installed on both systems with the command:

---

```
#rpm -qa | grep rsync
```

---

**NOTE**

*If you do not have the `rsync` program installed on your system, you can load the `rsync` source code included on the CD-ROM with this book in the `chapter4` subdirectory. (Copy the source code to a subdirectory on your system, untar it, and then type `./configure, make then make install.`)*

Now that we have a secure, encrypted method of transmitting data between our two cluster nodes, we can configure the `rsync` program to collect and ship data through this SSH transport. In the following steps, we'll create files in the `/www` directory on the SSH client and then copy these files to the SSH server using the `rsync` program and the `web` account.

1. Log on to the SSH client (the primary server) as root, and then enter:

---

```
#mkdir /www
#mkdir /www/htdocs
#echo "This is a test" > /www/htdocs/index.html
```

---

2. Give ownership of all of these files to the `web` user with the command:

---

```
#chown -R web /www
```

---

3. Now, on the SSH server (the backup server), we need to create at least the `/www` directory and give ownership of this directory to the `web` user because the `web` user does not have permission to create directories at the root level of the drive (in the `/` directory). To do so, log on to the SSH server (the backup server) as `root`, and then enter:

---

```
#mkdir /www  
#chown web /www
```

---

4. At this point, we are ready to use a simple `rsync` command on the SSH client (our primary server) to copy the contents of the `/www` directory (and all subdirectories) over to the SSH server (the backup server). Log on to the SSH client as the `web` user (from any directory), and then enter:

---

```
#rsync -v -a -z -e ssh --delete /www/ 10.1.1.2:/www
```

---

**NOTE** *If you leave off the trailing `/` in `/www/`, you will be creating a new subdirectory on the destination system where the source files will land rather than updating the files in the `/www` directory on the destination system.*

Because we use the `-v` option, the `rsync` command will copy the files verbosely. That is, you will see the list of directories and files that were in the local `/www` directory as they are copied over to the `/www` directory on the backup node.<sup>8</sup>

We are using these `rsync` options:

**-v**

Tells `rsync` to be verbose and explain what is happening.

**-a**

Tells `rsync` to copy all of the files and directories from the source directory.

**-z**

Tells `rsync` to compress the data in order to make the network transfer faster. (Our data will be encrypted and compressed before going over the wire.)

**-e ssh**

Tells `rsync` to use our `SSH` shell to perform the network transfer. You can eliminate the need to type this argument each time you issue an `rsync` command by setting the `RSYNC_RSH` shell environment variable. See the `rsync` man page for details.

**--delete**

Tells `rsync` to delete files on the backup server if they no longer exist on the primary server. (This option will not remove any primary, or source, data.)

---

<sup>8</sup> If this command does not work, check to make sure the `ssh` command works as described in the previous recipe.

/www/

This is the source directory on the SSH client (the primary server) with a trailing slash (/). A trailing slash is required if you want rsync to copy the entire directory and its contents.

**10.1.1.2:/www**

This is the destination IP address and destination directory.

**NOTE** *To tell rsync to display on the screen everything it will do when you enter this command, without actually writing any files to the destination system, add a -n to the list of arguments.*

If you now run this exact same rsync command again (#rsync -v -a -z -e ssh --delete /www/ 10.1.1.2:/www), rsync should see that nothing has changed in the source files and send almost nothing over the network connection, though it will still send a little bit of data to the rsync program on the destination system to check for disk blocks that have changed.

To see exactly what is happening when you run rsync, add the --stats option to the above command, as shown in the following example. Log on to the primary server (the SSH client) as the web user, and then enter:

---

```
rsync -v -a -z -e ssh --delete --stats /www/ 10.1.1.2:/www
```

---

rsync should now report that the Total transferred file size is 0 bytes, but you will see that a few hundred bytes of data were written.

### **Copying a Single File with rsync**

To copy just one file using rsync, enter:

---

```
#rsync -v -a -z -e ssh /etc/sysconfig/iptables 10.1.1.2:/etc/sysconfig/
```

---

Notice in this example that the destination directory is specified but that the destination file name is optional.

Here is how to copy one file from a remote machine (10.1.1.2) to the local one:

---

```
#rsync -v -a -z -e ssh 10.1.1.2:/etc/sysconfig/iptables /etc/sysconfig/iptables
```

---

In this example, we have specified the destination directory and the (optional) destination file name.

### **rsync over Slow WAN Connections**

If your primary and backup server connect to each other via the Internet (or a slow connection such as a private T1 line), you may also want to add the --bwlimit to the rsync command. The number you specify following --bwlimit is the maximum number of kilobytes that rsync should try to send over the

network per second. For example, to limit the rsync connection to 8 KB per second (or 1/24 of the total of a T1 connection), you could log on to the SSH client as the `web` user and enter:

---

```
rsync -v -a -z -e ssh --delete --bwlimit=8 /www/ 10.1.1.2:/www
```

---

## ***Scheduled rsync Snapshots***

Once you are happy with your rsync configuration, you can add a cron job to your primary server that will perform the rsync operation at regular intervals. For example, to synchronize the data in the `/www` directory at 30 minutes past the hour, every hour, you would enter the following command while logged on to the primary server as the `web` account:

---

```
#crontab -e
```

---

This should bring up the current crontab file for the `web` account in the `vi` editor.<sup>9</sup> You will see that this file is empty because you do not have any cron jobs for the `web` account yet. To add a job, press the `I` key to insert text and enter:

---

```
30 * * * * rsync -v -a -z -e ssh --delete /www/ 10.1.1.2:/www > /dev/null 2>&1
```

---

**NOTE** *You can add a comment on a line by itself to this file by preceding the comment with the `#` character. For example, your file might look like this:*

---

```
# This is a job to synchronize our files to the backup server.  
30 * * * * rsync -v -a -z -e ssh --delete /www/ 10.1.1.2:/www
```

---

Then press `ESC`, followed by the key sequence `:wq` to write and quit the file.

We have added `> /dev/null 2>&1` to the end of this line to prevent cron from sending an email message every half hour when this cron job runs (output from the `rsync` command is sent to the `/dev/null` device, which means the output is ignored). You may instead prefer to log the output from this command, in which case you should change `/dev/null` in this command to the name of a log file where messages are to be stored.

To cause this command to send an email message only when it fails, you can modify it as follows:

---

```
rsync -v -a -z -e ssh --delete /www/ 10.1.1.2:/www || echo "rsync failed"  
| mail admins@yourdomain.com
```

---

---

<sup>9</sup> This assumes that `vi` is your default editor. The default editor is controlled by the `EDITOR` shell environment variable.

## USING RSYNC TO COPY CONFIGURATION FILES FOR HIGH AVAILABILITY

In a Linux Enterprise Cluster, we need to build high-availability server pairs (using the Heartbeat package discussed later in this part of the book) to support the cluster. These high-availability servers will do things like offer print services to the cluster, load balance incoming requests to use the cluster, provide user account authentication information to the cluster nodes, and a variety of other things that help the cluster to function normally.

Most of these functions (except for load balancing) will be placed on a high-availability server pair called the *cluster node manager*. The cluster node manager will help to support the cluster (printing, user authentication, and so on) from one server while a backup cluster node manager server stands by ready to take over if this primary cluster node manager goes down. To do this, the backup cluster node manager—and every backup server in a high-availability server pair—needs to use the same configuration files as the primary server. The system administrator can create identical configuration files on a primary and backup server in a high-availability server pair by hand-editing each configuration file on both systems, but this method is prone to errors and is difficult to maintain. An easier method of keeping the backup server's configuration files the same as the primary server's is to simply copy them over the network using our secure data transport provided by Open SSH with the *rsync* utility.

Copying the configuration files from the primary server to the backup server in a high-availability server pair using Open SSH and *rsync* can also be automated by placing the *rsync* commands into a shell script that is then regularly executed by the *cron* daemon.

Here is a sample script that will copy the configuration files for the Heartbeat program itself<sup>1</sup> (which is described in Chapters 6 through 9), the *Mon* monitoring daemon (described in Chapter 17) and the *LPRng* printing system (which is described in Chapter 19).

---

```
#!/bin/bash
#
# rsync script to copy configuration files between primary
# and backup servers.
#
backup_host="10.1.1.2"
OPTS="--force --delete --recursive --ignore-errors -a -e ssh -z"
rsync $OPTS /etc/ha.d/      $backup_host:/etc/ha.d/
rsync $OPTS /etc/hosts      $backup_host:/etc/hosts
rsync $OPTS /etc/printcap   $backup_host:/etc/printcap
rsync $OPTS /var/spool/lpd/ $backup_host:/var/spool/lpd/
rsync $OPTS /etc/mon        $backup_host:/etc/mon
rsync $OPTS /usr/lib/mon/   $backup_host:/usr/lib/mon/
```

---

<sup>1</sup> As we'll see later in these chapters, the configuration files for Heartbeat should always be the same on the primary and the backup server. In some cases, however, the Heartbeat program must be notified of the changes to its configuration files (service heartbeat reload) or even restarted to make the changes take effect (when the /etc/ha.d/haresources file changes).

Note: Directories contain a / at the end. See Chapter 18 for another example of this script when used on a server that will run batch cron jobs for the cluster.

This script file is also provided on the CD-ROM included with this book. Normally, to copy configuration files, you will need to use the root account, although you can use the sudo program if you do not want to use the root account to copy the configuration files (see the sudo man page for details). This script is then run as a regularly scheduled root cron job on the backup server in the high-availability server pair with additional lines added to copy the configuration files for all of the services that need to be made highly available. (In the next section, we'll describe how to create a cron job entry.)

To view your cron job listing, enter:

---

```
#crontab -l
```

---

This cron job will take effect the next time the clock reads thirty minutes after the hour.

The format of the crontab file is in section 5 of the man pages. To read this man page, type:

---

```
#man 5 crontab
```

---

**NOTE** *See Chapter 1 to make sure the crond daemon starts each time your system boots.*

### ***ipchains/iptables Firewall Rules for rsync and SSH***

If your primary server uses its eth1 network interface to connect to the backup server, you can prevent unwanted (outside) attempts to use SSH or rsync with the following ipchains/iptables rules.

**NOTE** *You can accomplish the same thing these rules are doing by using access rules in the sshd\_config file.*

SSH uses port 22:

---

```
ipchains -A input -i eth1 -p tcp -s 10.1.1.0/24 1024:65535
-d 10.1.1.1 22 -j ACCEPT
```

---

or

---

```
iptables -A INPUT -i eth1 -p tcp -s 10.1.1.0/24 --sport 1024:65535
-d 10.1.1.1 --dport 22 -j ACCEPT
```

---

rsync uses port 873:

---

```
ipchains -A input -i eth1 -p tcp -s 10.1.1.0/24 1024:65535
-d 10.1.1.1 873 -j ACCEPT
```

---

or

---

```
iptables -A INPUT -i eth1 -p tcp -s 10.1.1.0/24 --sport 1024:65535
-d 10.1.1.1 --dport 873 -j ACCEPT
```

---

## In Conclusion

To build a highly available server pair that can be easily administered, you need to be able to copy configuration files from the primary server to the backup server when you make changes to files on the primary server. One way to copy configuration files from the primary server to the backup server in a highly available server pair is to use the `rsync` utility in conjunction with the Open Secure Shell transport.

It is tempting to think that this method of copying files between two servers can be used to create a very low-cost highly available data solution using only locally attached disk drives on the primary and the backup servers. However, if the data stored on the disk drives attached to the primary server changes frequently, as would be the case if the highly available server pair is an NFS server or an SQL database server, then using `rsync` combined with Open SSH is not an adequate method of providing highly available data. The data on the primary server could change, and the disk drive attached to the primary server could crash, long before an `rsync` cron job has the chance to copy the data to the backup server.<sup>10</sup> When your highly available server pair needs to modify data frequently, you should do one of the following things to make your data highly available:

- Install a distributed filesystem such as Red Hat's Global File System (GFS) on both the primary and the backup server (see Appendix D for a partial list of additional distributed filesystems).
- Connect both the primary and the backup server to a highly available SAN and then make the filesystem where data is stored a resource under Heartbeat's control.
- Attach both the primary and the backup server to a single shared SCSI bus that uses RAID drives to store data and then make the filesystem where data is stored a resource under Heartbeat's control.

When you install the Heartbeat package on the primary and backup server in a highly available server pair and make the shared storage file system (as described in the second two bullets) where your data is stored a resource under Heartbeat's control, the filesystem will only be mounted on one server at a time (under normal circumstances, the primary server).

When the primary server crashes, the backup server mounts the shared

---

<sup>10</sup> Using `rsync` to make snapshots of your data may, however, be an acceptable disaster recovery solution.

storage filesystem where the data is stored and launches the SQL or NFS server to provide cluster nodes with access to the data. Using a highly available server pair and shared storage in this manner ensures that no data is lost as a result of a failover.

I'll describe how to make a resource highly available using the Heartbeat package later in this part of the book, but before I do, I want to build on the concepts introduced in this chapter and describe how to clone a system using the SystemImager package.

# 5

## CLONING SYSTEMS WITH SYSTEMIMAGER



Now that we have a secure and reliable way to transfer files between computers (the subject of Chapter 4), we can make a backup copy, or system image, of one computer and store it on the disk drive of another computer. This system image can then be used to recover a system in the event of a disk crash, or it can be used to make a clone of the original system. Cloned systems can become backup servers in high-availability server pairs, or they can become cluster nodes. When changes are made to the original system, the cloned systems can be easily updated using a newly created system image of the original system.

### SystemImager

Using the SystemImager package, we can turn one of the cluster nodes into a *Golden Client* that will become the master image used to build all of the other cluster nodes. We can then store the system image of the Golden Client on

the backup server (offering the SSH server service discussed in the previous chapter). In this chapter, the backup server is called the *SystemImager server*, and it must have a disk drive large enough to hold a complete copy of the contents of the disk drive of the Golden Client, as shown in Figure 5-1.

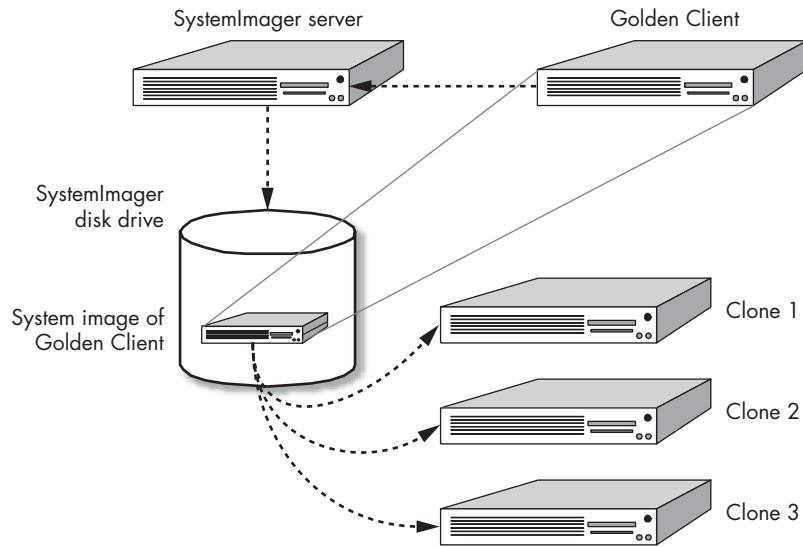


Figure 5-1: The SystemImager server and Golden Client

### Cloning the Golden Client with SystemImager

Clones 1, 2, and 3 in this diagram are all exact copies of the Golden Client—only their IP address information and host names are different.

Here, then, are the steps for cloning a system using the SystemImager package.

**NOTE** *The following recipe and the software included on the CD-ROM do not contain support for SystemImager over SSH. To use SystemImager over SSH (on RPM-based systems) you must download and compile the SystemImager source code as described at <http://www.systemimager.org/download>.*

## SystemImager Recipe

The list of ingredients follows. See Figure 5-2 for a list of software used on the Golden Client, SystemImager server, and clone(s).

Computer that will become the Golden Client (with SSH client software and rsync)

Computer that will become the SystemImager server (with SSH server software and rsync)

Computer that will become a clone of the Golden Client (with a network connection to the SystemImager server and a floppy disk drive)

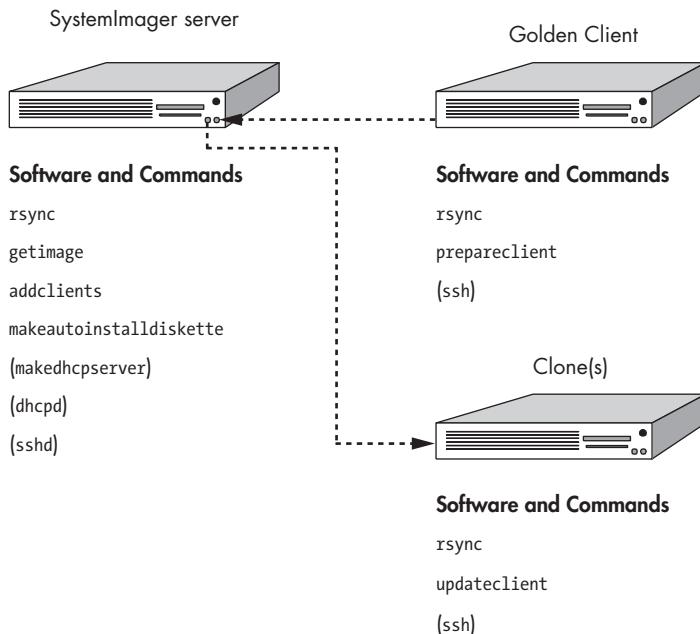


Figure 5-2: Software used on SystemImager server, Golden Client, and clone(s)

### ***Install the SystemImager Server Software on the SystemImager Server***

The machine you select to be the SystemImager server should have enough disk space to hold several system images. After you decide which machine to use, you can install the software using the installation program available on the download page of the SystemImager website, <http://www.systemimager.org>, or you can install the software using the CD-ROM included with this book. Before we can install the software, however, we need to install a few Perl modules that SystemImager depends upon for normal operation: `AppConfig`, `MLDBM`, and `XML-Simple`. The source code for these three Perl modules is included on the CD-ROM in `tar.gz` format in the `chapter5/perl-modules` directory. Copy them to a directory on the hard drive (`/usr/local/src/systemimager/perlmodules` in the following example), and compile and install them with the commands `perl Makefile.PL`, `make`, `make test`, and `make install`.

When you have finished installing these Perl modules, you're ready to install the SystemImager server package. If you are using an RPM-based distribution such as Red Hat, you can use the RPM files included with this book. Mount the CD-ROM, switch to the `chapter5` directory, and then type:

---

```
#rpm -ivh *rpm
```

---

**NOTE** *Only the RPM files for i386 hardware are included on the CD-ROM. For other types of hardware, see the SystemImager website.*

The RPM command may complain about failed dependencies. If the only dependencies it complains about are the Perl modules we just installed ( `AppConfig`, which is sometimes called `libappconfig`, `MLDBM`, and `XML-Simple`), you can force the RPM command to ignore these dependencies and install the RPM files with the command:

---

```
#rpm -ivh --nodeps *rpm
```

---

## ***Using the Installer Program from SystemImager***

If you just installed the software using the CD-ROM included with this book, you can skip this section.

The list of software installed by the SystemImager install program is currently located (and downloaded each time the script runs) at <http://sisuite.org/tags/stable.txt>. It contains URLs to locate the *common* software (common to both the SystemImager client and the SystemImager server) required on the SystemImager server, and the *server* software. You will also see different versions of the kernel boot architecture rpms. The install script will download only the packages you specify and any dependencies they have that are also in the list (`--list`). For all versions of i86 hardware, the `i386` boot rpm is used. You can download all available boot architectures, but normally you will only need one (to support the architecture you are using on all cluster nodes). The commands to list the available SystemImager packages are as follows:

---

```
#wget http://sisuite.org/install
#chmod +x install
#./install -list
Then install the packages you need using a command like this:
#./install -verbose systemimager-common systemimager-server systemimager-
i386boot-standard
```

---

**NOTE** *The installer program runs the `rpm` command to install the packages it downloads.*

## ***Install the SystemImager Client Software on the Golden Client***

Also included on the CD-ROM with this book is the SystemImager client software. This software is located in the `chapter5/client` directory.

**NOTE** *As with all of the software in this book, you can download the latest version of the SystemImager package from the Internet.*

Before you can install this software, you'll need to install the Perl  `AppConfig` module, as described in the previous step (it is located on the CD-ROM in the `chapter5/perl-modules` directory). Again, because the RPM program expects to find dependencies in the RPM database, and we have installed the  `AppConfig` module from source code, you can tell the `rpm` program to ignore the dependency failures and install the client software with the following commands.

---

```
#mount /mnt/cdrom
#cd /mnt/cdrom/chapter5/client #rpm -ivh --nodeps *
```

---

You can also install the SystemImager client software using the installer program found on the SystemImager download website (<http://www.systemimager.org/download>).

Once you have installed the packages, you are ready to run the command:

---

```
#/usr/sbin/prepareclient -server <servername>
```

---

This script will launch rsync as a daemon on this machine (the Golden Client) so that the SystemImager server can gain access to its files and copy them over the network onto the SystemImager server. You can verify that this script worked properly by using the command:

---

```
#ps -elf | grep rsync
```

---

You should see a running rsync daemon (using the file `/tmp/rsyncd.conf` as a configuration file) that is waiting for requests to send files over the network.

### ***Create a System Image of the Golden Client on the SystemImager Server***

We are now ready to copy the contents of the Golden Client's disk drive(s) to the SystemImager server. The images will be stored on the SystemImager server in the `/var/lib/systemimager` directory. Check to make sure you have enough disk space in this directory by doing the following:

---

```
#df -h /var/lib/systemimager
```

---

If this is not enough space to hold a copy of the contents of the disk drive or drives on the Golden Client, use the `-directory` option when you use the `getimage` command, or change the default location for images in the `/etc/systemimager/systemimager.conf` file and then create a link back to it, `ln -s /usr/local/systemimager /var/lib/systemimager`.

Now (optionally, if you are not running the command as root) give the `web` user<sup>1</sup> permission to run the `/usr/sbin/getimage` command by editing the `/etc/sudoers` file with the `visudo` command and adding the following line:

---

```
web siserver.yourdomain.com = NOPASSWD:/usr/sbin/getimage
```

---

where `siserver.yourdomain.com` is the host name (see the output of the `hostname` command) of the SystemImager server.

We can now switch to the `web` user account on the SystemImager server and run the `/usr/sbin/getimage` command.

---

<sup>1</sup> Or any account of your choice. The command to create the `web` user was provided in Chapter 4.

Logged on to the SystemImager server:

---

```
#sudo /usr/sbin/getimage -user-ssh web -golden-client 10.1.1.2 -image
backup.yourdomain.com
```

---

In this example, the Golden Client is at IP address 10.1.1.2.

If you are logged on as root and *do not* need to use the SSH protocol to encrypt your network traffic, use the following command:

---

```
#/usr/sbin/getimage -golden-client 10.1.1.2 -image backup.yourdomain.com
```

---

Notes:

- The `-image` name does not necessarily need to match the host name of the Golden Client, though this is normally how it is done.
- You cannot get the image of a Golden Client unless it knows the IP address and host name of the image server (usually specified in the `/etc/hosts` file on the Golden Client).
- On busy networks, the `getimage` command may fail to complete properly due to timeout problems. If this happens, try the command a second or third time. If it still fails, you may need to run it at a time of lower network activity.
- When the configuration changes on the backup server, you will need to run this command again, but thanks to `rsync`, only the disk blocks that have been modified will need to be copied over the network. (See “Performing Maintenance: Updating Clients” at the end of this chapter for more information.)

Once you answer the initial questions and the script starts running, it will spend a few moments analyzing the list of files before the copy process begins. You should then see a list of files scroll by on the screen as they are copied to the local hard drive on the SystemImager server.

When the command finishes, it will prompt you for the method you would like to use to assign IP addresses to future client systems that will be clones of the backup data server (the Golden Client). If you are not sure, select `static` for now.

When prompted, select `y` to run the `addclients` program (if you select `n` you can simply run the `addclients` command by itself later).

You will then be prompted for your domain name and the “base” host name you would like to use for naming future clones of the backup data server. As you clone systems, the SystemImager software automatically assigns the clones host names using the name you enter here followed by a number that will be incremented each time a new clone is made (`backup1`, `backup2`, `backup3`, and so forth in this example).

The script will also help you assign an IP address range to these host names so they can be added to the `/etc/hosts` file. When you are finished,

the entries will be added to the `/etc/hosts` file (on the SystemImager server) and will be copied to the `/var/lib/systemimager/scripts/` directory. The entries should look something like this:

---

```
backup1.yourdomain.com backup1
backup2.yourdomain.com backup2
backup3.yourdomain.com backup3
backup4.yourdomain.com backup4
backup5.yourdomain.com backup5
backup6.yourdomain.com backup6
```

---

and so on, depending upon how many clones you plan on adding.

### ***Make the Primary Data Server into a DHCP Server***

Normally, you will skip this section and use the static method to assign IP addresses (so that you can skip this section entirely by placing the IP address of the client on the boot floppy created in the next section). If you already use DHCP or don't want to install a DHCP server on your network, skip to the next section now.

First, we need to be sure the DHCPD and TFTPD software is loaded on the SystemImager server. On a Red Hat system, you can do this with the following commands:

---

```
#rpm -qa | grep dhcp
```

---

You may have the DHCP client software installed on your computer. The DHCP client software RPM package is called `dhpcd`, but we need the `dhcp` package.

If this command does not return the version number of the currently installed DHCP software package, you will need to install it either from your distribution CD or by downloading it from the Red Hat (or Red Hat mirror) FTP site (or using the Red Hat `up2date` command).

**NOTE** *As of Red Hat version 7.3, the init script to start the `dhcpd` server (`/etc/rc.d/init.d/dhcpd`) is contained in the `dhcp` RPM.*

We are now ready to modify the DHCPD configuration file `/etc/dhcpd.conf`. Fortunately, the SystemImager package includes a utility called `mkdhcpserver`,<sup>2</sup> which will ask questions and then do this automatically.

As root on the SSH server, type the command:

---

```
#mkdhcpserver
```

---

Once you have entered your domain name, network number, and netmask, you will be asked for the starting IP address for the DHCP range. This should be the same IP address range we just added to the `/etc/hosts` file.

---

<sup>2</sup> Older versions of SystemImager used the command `makedhcpserver`.

In this example, we will not use SSH to install the clients.<sup>3</sup>

The IP address of the image server will also be the cluster network internal IP address of the primary data server (the same IP address as the default gateway):

---

Here are the values you have chosen:

```
#####
DNS domain name:           yourdomain.com
Network number:            10.1.1.0
Netmask:                   255.255.255.0
Starting IP address for your DHCP range: 10.1.1.3
Ending IP address for your DHCP range:   10.1.1.254
First DNS server:
Second DNS server:
Third DNS server:
Default gateway:           10.1.1.1
ImageServer:                10.1.1.1
SSH files download URL:
#####
Are you satisfied? (y/[n]):
```

---

If all goes well, the script should ask you if you want to restart the DHCPD server. You can say no here because we will do it manually in a moment.

View the `/etc/dhcpd.conf` file to see the changes that this script made by typing:

---

```
#vi /etc/dhcpd.conf
```

---

Before we can start DHCPD for the first time, however, we need to create the `leases` file. This is the file that the DHCP server uses to hold a particular IP address for a MAC address for a specified lease period (as controlled by the `/etc/dhcpd.conf` file). Make sure this file exists by typing the command:

---

```
#touch /var/lib/dhcp/dhcpd.leases
```

---

Now we are ready to stop and restart the DHCP server, but as the script warns, we need to make sure it is running on the proper network interface card. Use the `ifconfig -a` command and determine what the `eth` number is for your cluster network (the `10.1.1.x` network in this example).

If you have configured your primary data server as described in this chapter, you will want DHCP to run on the `eth1` interface, so we need to edit the DHCPD startup script and modify the `daemon` line as follows.

---

<sup>3</sup> If you really need to create a newly cloned system at a remote location by sending your system image over the Internet, see the SystemImager manual online at <http://systemimager.org> for instructions.

---

```
#vi /etc/rc.d/init.d/dhcpd
```

---

Change the line that reads:

---

```
daemon /usr/sbin/dhcpd
```

---

so it looks like this:

---

```
daemon /usr/sbin/dhcpd eth1
```

---

Also, (optionally) you can configure the system to start DHCP automatically each time the system boots (see Chapter 1). For improved security and to avoid conflicts with any existing DHCP servers on your network, however, do not do this (just start it up manually when you need to clone a system).

Now manually start DHCPD with the command:

---

```
#/etc/init.d/dhcpd start
```

---

(The command `service dhcpcd start` does the same thing on a Red Hat system.)

Make sure the server starts properly by checking the messages log with the command:

---

```
#tail /var/log/messages
```

---

You should see DHCP reporting the “socket” or network card and IP address it is listening on with messages that look like this:

---

```
dhcpcd: Listening on Socket/eth1/10.1.1.0
dhcpcd: Sending on  Socket/eth1/10.1.1.0
dhcpcd: dhcpcd startup succeeded
```

---

### ***Create a Boot Floppy for the Golden Client***

We are now ready to create a floppy diskette that can be used to boot a new system and automatically copy over the entire system image to the hard drive. Create a new Linux<sup>4</sup> boot floppy on the primary data server by placing a blank floppy in the drive and typing the SystemImager command<sup>5</sup>:

---

```
#mkautoinstalldiskette
```

---

---

<sup>4</sup>This boot floppy will contain a version of Linux that is designed for the SystemImager project, called Brian’s Own Embedded Linux or BOEL. To read more about BOEL, see the following article at LinuxDevices.com (<http://www.linuxdevices.com/articles/AT9049109449.html>).

<sup>5</sup>Older versions of SystemImager used the command `makeautoinstalldiskette`.

**NOTE** We will not use the SSH protocol to install the Golden Client for the remainder of this chapter. If you want to continue to use SSH to install the Golden Client (not required to complete the install), see the help page information on the `mkautoinstalldiskette` command.

If, after the script formats the floppy, you receive the error message stating that:

---

```
mount: fs type msdos not supported by kernel
```

---

you need to recompile your kernel with support for the MSDOS filesystem.<sup>6</sup>

**NOTE** The script will use the `superformat` utility if you have the `fd-utils` package installed on your computer (but this package is not necessary for the `mkautoinstalldiskette` script to work properly).

If you did not build a DHCP server in the previous step, you should now install a `local.cfg` file on your floppy.<sup>7</sup> The `local.cfg` file contains the following lines:

---

```
#  
# "SystemImager" - Copyright (C) Brian Elliott Finley <brian@systemimager.org>  
#  
# This is the /local.cfg file.  
#  
HOSTNAME=backup1  
DOMAINNAME=yourdomain.com  
DEVICE=eth0  
IPADDR=10.1.1.3  
NETMASK=255.255.255.0  
NETWORK=10.1.1.0  
BROADCAST=10.1.1.255  
GATEWAY=10.1.1.1  
GATEWAYDEV=eth0  
IMAGESERVER=10.1.1.1
```

---

Create this file using `vi` on the hard drive of the image server (the file is named `/tmp/local.cfg` in this example), and then copy it to the floppy drive with the commands:

---

```
#mount /mnt/floppy  
#cp /tmp/local.cfg /mnt/floppy  
#umount /mnt/floppy
```

---

<sup>6</sup> In the `make menuconfig` screen, this option is under the Filesystems menu. Select (either as `M` for modular or `*` to compile it directly into the kernel) the `DOS FAT fs support` option and the `MSDOS fs support` option will appear. Select this option as well; then recompile the kernel (see Chapter 3).

<sup>7</sup> You can also do this when you execute the `mkautoinstalldiskette` command. See `mkautoinstalldiskette -help` for more information.

When you boot the new system on this floppy, it will use a compact version of Linux to format your hard drive and create the partitions to match what was on the Golden Client. It will then use the local.cfg file you installed on the floppy, or if one does not exist, the DHCP protocol to find the IP address it should use. Once the IP address has been established on this new system, the install process will use the rsync program to copy the Golden Client system image to the hard drive.

Also, if you need to make changes to the disk partitions or type of disk (IDE instead of SCSI, for example), you can make changes to the script file that will be downloaded from the image server and run on the new system by modifying the script located in the scripts directory (defined in the /etc/systemimager/systemimager.conf file).<sup>8</sup> The default setting for this directory path is /var/lib/systemimager/scripts.<sup>9</sup> This directory should now contain a list of host names (that you specified when the addclients script in Step 3 of this recipe). All of these host-name files point to a single (real) file containing a script that will run on each clone machine at the start of the cloning process. So in this example, the directory listing would look like this:

---

```
#ls -ls /var/lib/systemimager/scripts/backup*
```

---

The output of this command looks like this:

---

```
/var/lib/systemimager/backup1.sh -> backup.yourdomain.com.master
/var/lib/systemimager/backup2.sh -> backup.yourdomain.com.master
/var/lib/systemimager/backup3.sh -> backup.yourdomain.com.master
/var/lib/systemimager/backup4.sh -> backup.yourdomain.com.master
/var/lib/systemimager/backup5.sh -> backup.yourdomain.com.master
/var/lib/systemimager/backup6.sh -> backup.yourdomain.com.master
/var/lib/systemimager/backup7.sh -> backup.yourdomain.com.master
/var/lib/systemimager/backup.yourdomain.com.master
```

---

Notice how all of these files are really symbolic links to the last file on the list:

---

```
/var/lib/systemimager/backup.yourdomain.com.master
```

---

This file contains the script, which you can modify if you need to change anything about the basic drive configuration on the clone system. It is possible, for example, to create a Golden Client that uses a SCSI disk drive and then modify this script file (changing the sd to hd and making other modifications) and then install the clone image onto a system with an IDE hard drive. (The script file contains helpful comments for making disk partition changes.)

---

<sup>8</sup> See “How do I change the disk type(s) that my target machine(s) will use?” in the SystemImager FAQ for more information.

<sup>9</sup> /tftpboot/systemimager in older versions of SystemImager.

This kind of modification is tricky, and you need the right type of kernel configuration (the correct drivers) on the original Golden Client to make it all work, so the easiest configuration by far is to match the hardware on the Golden Client and any clones you create.

### ***Start rsync as a Daemon on the Primary Data Server***

You'll need to start rsync as a daemon on the SystemImager server and point it at the rsync configuration file created and maintained by the SystemImager programs in the /etc/systemimager directory. The SystemImager installation procedure should have created the init script /etc/init.d/systemimager-server-rsyncd. This script starts rsync as a daemon (rsync stays running in the background waiting for the cloning processes to start on a new system) when you enter the command:

---

```
#/etc/init.d/systemimager-server-rsyncd start
```

---

(Or service systemimager-server-rsyncd start on Red Hat systems.)

You can also start rsync manually with the command:

---

```
#rsync --daemon --config=/etc/systemimager/rsyncd.conf
```

---

rsync gets its configuration from the file /etc/systemimager/rsyncd.conf. This file tells the rsync daemon to report its error messages to the file /var/log/systemimager/rsyncd. Examine this log now with the command:

---

```
#tail -f /var/log/systemimager/rsyncd
```

---

This should show log file entries indicating that rsync was successfully started (press CTRL-C to break out of this command).

### ***Install the Golden Client System Image on the New Clone***

Connect your new clone system to the network and boot it from the floppy disk you just created. After it boots from this floppy, it will use the local.cfg file to set its IP address information or contact the DHCP server and receive the next IP address you have configured for your clones.

This newly connected virginal clone machine will then use an rsync command such as the following to pull down the Golden Client system image:

---

```
rsync -av -bwlimit=10000 -numeric-ids 10.1.1.1::backup.yourdomain.com/ /a
```

---

This command is asking the 10.1.1.1 machine (the SystemImager server) to send all of the files it has for the backup.yourdomain.com image. How does rsync running on the SystemImager server know which files to send? It examines the /etc/systemimager/rsyncd.conf file for an entry such as the following.

---

```
[backup.yourdomain.com]
  path = /var/lib/systemimager/images/backup.yourdomain.com
```

---

This allows the rsync program to resolve the rsync module name `backup.yourdomain.com` into the path `/var/lib/systemimager/images/backup.yourdomain.com` to find the correct list of files.

You can watch what is happening during the cloning process by running the `tcpdump` command on the server (see Appendix B) or with the command:

---

```
#tail -f /var/log/systemimager/rsyncd
```

---

To see the network queues, type:

---

```
#netstat -tn
```

---

When rsync is finished copying the Golden Client image to the new system's hard drive, the installation process runs **System Configurator** to customize the boot image (especially the network information) that is now stored on the hard drive so that the new system will boot with its own host and network identity.

When the installation completes, the system will begin beeping to indicate it is ready to have the floppy removed and the power cycled. Once you do this, you should have a newly cloned system.

### **WHEN THE SYSTEMIMAGER BOOT FLOPPY FAILS**

If the system does not boot properly, you can use the Linux shell prompt left by the SystemImager client software to test network connectivity to the SystemImager server using the `ping` and `telnet` commands. You can also use the sample rsync command above to test the configuration before rebooting on the floppy once you have resolved the network communication issues.

If you see an error message indicating you have exceeded the size of the disk drive when the SystemImager software attempts to write the partition information to the new hard drive, you should check the output of the `dmesg` command on both the original machine and from the shell prompt after the system fails to install from the floppy. From the output of the `dmesg` command, you can determine the raw size of the disk drives connected to the original machine (the Golden Client) and the new clone. If the disk drive is smaller on the new clone, you need to modify the installation script on the image server (the installation script is copied down to the clone after the system boots from the floppy, so there is no need to rebuild the floppy). By default, scripts are stored in the `/var/lib/systemimager/scripts` directory (see the `/etc/systemimager/systemimager.conf` file for the location on your system).

### ***Post-installation Notes***

The system cloning process modifies the Red Hat network configuration file `/etc/sysconfig/network`. Check to make sure this file contains the proper

network settings after booting the system. (If you need to make changes, you can rerun the network script with the command `service network restart`, or better yet, test with a clean reboot.)

Check to make sure the host name of your newly cloned host is in its local `/etc/host` file (the LPRng daemon called `lpd`, for example, will not start if the host name is not found in the hosts file).

Configure any additional network interface cards. On Red Hat systems, the second Ethernet interface card is configured using the file `/etc/sysconfig/network-scripts/ifcfg-eth1`, for example.

If you are using SSH, you need to create new SSH keys for local users (see the `ssh-keygen` command in the previous chapter) and for the host (on Red Hat systems, this is accomplished by removing the `/etc/ssh/ssh_host*` files). Note, however, the simplest and easiest configuration to administer is one that uses the same SSH keys on all cluster nodes. (When a client computer connects to the cluster using SSH, they will always see the same SSH host key, regardless of which cluster node they connect to, if all cluster nodes use the same SSH configuration.)

If you started a DHCP server, you can turn it back off until the next time you need to clone a system:

---

```
#/etc/rc.d/init.d/dhcpd stop
```

---

Finally, you will probably want to kill the rsync daemon started by the `prepareclient` command on the Golden Client.

## Performing Maintenance: Updating Clients

If you make changes to the Golden Client, simply enter the `getimage` command on the SystemImager server; then `updateclient` on the cloned systems (the backup servers in the high-availability server pairs or the cluster nodes inside the cluster).

**NOTE** *If you need to exclude certain directories from an update, create an entry in `/etc/systemimager/updateclient.local.exclude`.*

### **SystemInstaller**

The SystemInstaller project grew out of the Linux Utility for cluster Installation (LUI) project and allows you to install a Golden Image directly to a SystemImager server using RPM packages. SystemInstaller does not use a Golden Client; instead, it uses RPM packages that are stored on the SystemImager server to build an image for a new clone, or cluster node.<sup>10</sup>

The SystemInstaller website is currently located at <http://system-installer.sourceforge.net>.

---

<sup>10</sup> This is the method used by the OSCAR project, by the way. SystemInstaller runs on the “head node” of the OSCAR cluster.

SystemInstaller uses RPM packages to make it easier to clone systems using different types of hardware (SystemImager works best on systems with similar hardware).

## ***System Configurator***

As we've previously mentioned, SystemImager uses the System Configurator to configure the boot loader, loadable modules, and network settings when it finishes installing the software on a clone. The System Configurator software package was designed to meet the needs of the SystemImager project, but is distributed and maintained separate from the SystemImager package with the hope that it will be used for other applications. For example, System Configurator could be used by a system administrator to install driver software and configure the NICs of all cluster nodes even when the cluster nodes use different model hardware and different Linux distributions.

For more information about System Configurator, see <http://sisuite.org/systemconfig>.

System Installation Suite, SystemInstaller, SystemImager, and System Configurator are collectively called the System Installation Suite. The goal of the System Installation Suite is to provide an operating system independent method of installing, maintaining, and upgrading systems.

For more information on the System Installation Suite, see <http://sisuite.sourceforge.net>.

## **In Conclusion**

The SystemImager package uses the rsync software and (optionally) the SSH software to copy the contents of a system, called the Golden Client, onto the disk drive of another system, called the SystemImager server. A clone of the Golden Client can then be created by booting a new system off of a specially created Linux boot floppy containing software that formats the locally attached disk drive(s) and then pulls the Golden Client image off of the SystemImager server.

This cloning process can be used to create backup servers in high-availability server pairs, or cluster nodes. It can also be used to update cloned systems when changes are made to the Golden Client.



# 6

## HEARTBEAT INTRODUCTION AND THEORY



This chapter introduces a software package called Heartbeat that gives you the ability to failover a resource from one computer to another. The following three chapters will explore Heartbeat and high-availability techniques in detail. Part III of this book will build on the techniques described here, and it will explain how to build a highly available cluster. Once you know how to use Heartbeat properly, you can deploy just about any service and make it highly available.

**NOTE** *A highly available system has no single points of failure. A single point of failure is a single system component that upon failing causes the whole system to fail.*

Heartbeat works like this: you tell Heartbeat which computer owns a particular resource (which computer is the *primary server*), and the other computer will automatically be the *backup server*. You then configure the

Heartbeat daemon running on the backup server to listen to the “heartbeats” coming from the primary server. If the backup server does not hear the primary server’s heartbeat, it initiates a failover and takes ownership of the resource.

## The Physical Paths of the Heartbeats

The Heartbeat program running on the backup server can check for heartbeats coming from the primary server over the normal Ethernet network connection, but normally Heartbeat is configured to work over a separate physical connection between the two servers. This separate physical connection can be either a serial cable or another Ethernet network connection (via a crossover cable<sup>1</sup> or mini hub, for example).

Heartbeat will work over one or more of these physical connections at the same time and will consider the primary node active as long as heartbeats are received on at least one of the physical connections. Figure 6-1 shows three physical connections, or paths, between the servers. The first path, the normal Ethernet network used to connect systems to each other on the network, is the least preferred for sending the heartbeats, because it will add extra traffic to your network (though this is a trivial load under normal circumstances). Your choice of whether to use one or more new serial or Ethernet connections will depend on your situation.

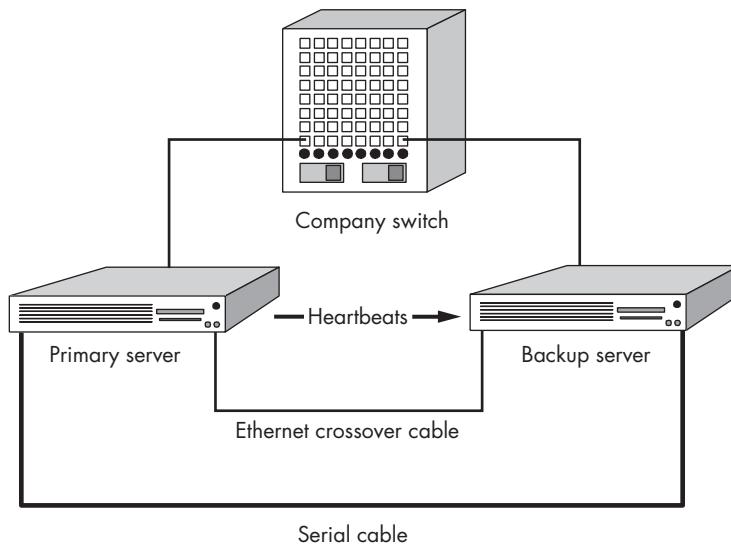


Figure 6-1: Physical paths for heartbeats

<sup>1</sup> A crossover cable is simpler and more reliable than a mini hub because it does not require external power.

**NOTE** *High-availability best practices dictate that heartbeats should travel over multiple independent communication paths.<sup>2</sup> This helps eliminate the communications path from being a single point of failure.*

### ***Serial Cable Connection***

A serial connection is slightly more secure than an Ethernet connection, because a hacker will not be able to run telnet, ssh, or rlogin over the serial cable if they break into one of the systems. (The serial cable is a simple crossover cable connected to the COM port on each system.) However, because serial cables are short,<sup>3</sup> the servers must be located near each other, usually in the same computer room.

### ***Ethernet Cable Connection***

Using a new Ethernet network (or Ethernet crossover cable) eliminates any distance limitation between the servers. It also allows you to synchronize the filesystems on the two servers (as described in Chapter 4) without placing any extra network traffic on your normal Ethernet network.

Using two physical paths to connect the primary and backup servers provides redundancy for heartbeat control messages and is therefore a requirement of a no-single-point-of-failure configuration. The two physical paths between the servers need not be of the same type; an Ethernet and a serial connection can be used together in the same configuration.

### ***Partitioned Clusters and STONITH***

For true redundancy, two physical connections should carry heartbeat control messages between the primary and backup server. These two physical connections will help prevent a situation where a network or cable failure causes both nodes to try and assume ownership of the same resources. This condition is known as a *split-brain* or *partitioned* cluster,<sup>4</sup> and it can have dire consequences if you are using two heartbeat nodes to control one physical device (such as a shared SCSI or Fibre Channel disk drive). To avoid this situation, take the following precautions:

- Create a redundant, reliable physical connection between heartbeat nodes (preferably using both a serial connection and an Ethernet connection) to carry heartbeat control messages.
- Allow for the ability to forcibly shut down one of the heartbeat nodes when a partitioned cluster is detected.

---

<sup>2</sup> In *Blueprints for High Availability*, Evan Marcus and Hal Stern define three different types of networks used in failover configurations: the Heartbeat network, the production network (for client access to cluster resources), and an administrative network (for system administrators to access the servers and do maintenance tasks).

<sup>3</sup> The original EIA-232 specification did not specify a distance limitation, but 50 feet has become the industry's de facto distance limit for normal serial communication. See the Serial HOWTO for more information.

<sup>4</sup> Sometimes the term *cluster fencing* or *i/o fencing* is used to describe what should happen when a cluster is partitioned. It means that the cluster must be able to build a fence between partitions and decide on which side of the fence the cluster resources should reside.

This second precaution has been dubbed “shoot the other node in the head,” or STONITH. Using a special hardware device that can power off a node through software commands (sent over a serial or network cable), Heartbeat can implement a Stonith<sup>5</sup> configuration designed to avoid cluster partitioning. (See Chapter 9 for more information.)

**NOTE** *It is difficult to guarantee exclusive access to resources and avoid split-brain conditions when the primary and backup heartbeat servers are not in close proximity. You will very likely reduce resource reliability and increase system administration headaches if you try to use Heartbeat as part of your disaster recovery or business resumption plan over a wide area network (WAN). In the cluster solution described in this book, no Heartbeat pairs of servers need communicate over a WAN.*

## Heartbeat Control Messages

In this chapter we will look at the three most basic heartbeat control messages<sup>6</sup> (three kinds of packets, if you are using an Ethernet network):

- Heartbeats or status messages
- Cluster transition messages
- Retransmission requests

### ***Heartbeats***

Heartbeats (sometimes called *status messages*) are broadcast, unicast, or multicast packets that are only about 150 bytes long. You control how often each computer broadcasts its heartbeat and how long the heartbeat daemon running on another node should wait before assuming something has gone wrong.

### ***Cluster Transition Messages***

The two most prevalent cluster transition messages are ip-request and ip-request-resp. These messages are relatively rare and contain the conversation between heartbeat daemons when they want to move a resource from one computer to another.

When you repair the primary server and it comes back online, it uses ip-request to ask the backup server to release the resource it took ownership of when the primary server failed. The backup server then shuts off the service and replies with an ip-request-resp message to inform the primary server that it no longer owns the resource. When the primary server receives this ip-request-resp, it starts up the service and offers it to the client computers again (it takes back ownership of the resource).

---

<sup>5</sup> Although an acronym at birth, this term has moved up in stature to become, by all rights, a word—it will be used as such throughout the remainder of this book.

<sup>6</sup> For a complete list of heartbeat control message types see this website: <http://www.linux-ha.org/comm/Heartbeat-msgfmt.html>.

## ***Retransmission Requests***

The rexmit-request (or ns\_rexmit) message—a request for a retransmission of a heartbeat control message—is issued when one of the servers running heartbeat notices that it is receiving heartbeat control messages that are out of sequence. (Heartbeat daemons use sequence numbers to ensure packets are not dropped or corrupted.) Heartbeat daemons will only ask for a retransmission of a heartbeat control message (with no sequence number, hence the “ns” in ns\_rexmit) once every second, to avoid flooding the network with these retry requests when something goes wrong.

## ***Ethernet Heartbeat Control Messages***

All three of these heartbeat control messages are sent using the UDP protocol to either the port number specified in the /etc/ha.d/ha.cf file, or to the multicast address specified in this same configuration file (when using Ethernet).

Currently Heartbeat does not support more than two nodes.<sup>7</sup> More than one pair of heartbeat servers can share the same Ethernet network connection and exchange heartbeats and heartbeat control messages, but each of these pairs of heartbeat servers must use a unique UDP port number as specified in the /etc/ha.d/ha.cf file, or a unique unicast or multicast address.

## ***Security and Heartbeat Control Messages***

In addition to using a numbering sequence to recover from dropped or corrupted packets, Heartbeat digitally signs each packet using either a 128-bit hashing algorithm called MD5 (see RFC<sup>8</sup> 1321), or the even more secure 160-bit HMAC-SHA1 (see RFC 2104). (You enter the same encryption password for either of these methods on both the primary and the backup heartbeat nodes.)

**NOTE** *The Heartbeat developers recommend that you use one of these encryption methods even on private networks to protect Heartbeat from an attacker (spoofed packets or a packet replay attack).*

## **How Client Computers Access Resources**

Normally client computers know the name of the server offering the resource they want to use (mail.mydomain.com or www.mydomain.com, for example), and they use the Domain Name System (DNS) to look up the proper IP

---

<sup>7</sup> To be more accurate, Heartbeat will only allow two nodes to share haresources entries (see Chapter 8 for more information about the proper use of the haresources file). In this book we are focusing on building a pair of high-availability LVS-DR directors to achieve high-availability clustering.

<sup>8</sup> The Requests for Comments (RFCs) can be found at <http://www.rfc-editor.org>.

address of the server. Once the routing of the packet through the Internet or WAN is done, however, this IP address has to be converted into a physical network card address called the Media Access Control (MAC) address.

At this point, the router or the locally connected client computers use the Address Resolution Protocol (ARP) to ask: “Who owns this IP address?” When the computer using the IP address responds, the router or client computer adds the IP address and its corresponding MAC address to a table in its memory called the ARP table so it won’t have to ask each time it needs to use the IP address. After a few minutes, most computers let unused ARP table entries expire, to make sure they do not hold unused (or inaccurate) addresses in memory.<sup>9</sup>

### ***Failover Using IP Address Takeover (IPAT)***

In order to move a resource (a service or daemon and its associated IP address) from one computer to another, we need a way to move the IP address from the primary computer to the backup computer. The method normally used by Heartbeat is called IP address takeover (sometimes called IPAT). To accomplish IPAT, Heartbeat uses secondary IP addresses (formerly called *IP aliases* in older Linux kernels) and gratuitous ARP broadcasts.

#### **FAILING OVER MULTIPLE IP ALIASES**

In theory, you can add as many secondary IP addresses (or IP aliases) as your network IP address numbering scheme will allow (depending upon your kernel version). However, if you want Heartbeat to move more than a few IP addresses from a primary server to a backup server when the primary server fails, you may want to use a different method for IP address failover than the method described in this chapter. In fact, Heartbeat versions prior to version 0.4.9 were limited to eight IP addresses per network interface card (NIC) unless you used this method. (See the file `/usr/share/doc/packages/heartbeat/faqntips.html` after installing the Heartbeat RPM for instructions on how to configure your router for this alternative method.)

## **Secondary IP Addresses and IP Aliases**

*Secondary IP addresses* and *IP aliases* are two different methods for adding multiple IP addresses to the same physical network card. The first IP address (also called the *primary address*) is added to the NIC at boot time. Additional IP addresses are then added by Heartbeat based on entries in the `haresources` configuration file (we’ll discuss this file in detail in Chapter 8)—additional IP addresses are either IP aliases or secondary IP addresses. As of this writing, the Linux kernel supports both IP aliases and secondary IP addresses, though IP aliases are deprecated in favor of secondary IP addresses.

---

<sup>9</sup> For more information see RFC 826, “Ethernet Address Resolution Protocol.”

**NOTE** IP aliasing (sometimes called network interface aliasing) is a standard feature of the Linux kernel when standard IPv4 networking is configured in the kernel. Older versions of the kernel required you to configure IP alias support as an option in the kernel.

## IP ALIASES AND SOURCE ADDRESS

Whether you use secondary IP addresses or IP aliases, the source address used in packets that originate from the server (connections created from the server) will use the primary IP address on the NIC. You can force the kernel to make the source address appear to be one of the secondary IP addresses or IP aliases associated with the NIC by using the ip command and the undocumented src option. For example:

```
#ip route add default via 209.100.100.254 src 209.100.100.3 dev eth0
```

In this example, 209.100.100.254 is the gateway, and 209.100.100.3 is the secondary IP address (or IP alias) you want to use as a source address for outgoing packets. This method is used by the ipsrcaddr resource script included with the Heartbeat package. (You can also use the command ip route get 209.100.100.10 to see which interface and source address is returned by the kernel's routing table for a particular IP address.)

Note: Forcing the kernel to use a secondary IP address or IP alias for the return address in the packet header does not work if both sides of an active/active Heartbeat pair of servers use the ipsrcaddr resource, or if different resources need to use different source addresses.

By using secondary IP addresses or IP aliases you can offer a service such as sendmail on one IP address and offer another service, like HTTP, on another IP address, even though these two IP addresses are really owned by the same computer (one physical network card at one MAC address).

When you use Heartbeat to offer services on a secondary IP address (or IP alias) the service is owned by the server (meaning the server is the active, or primary, node) and the server also owns the IP addresses used to access the service. The backup node must not be using this secondary IP address (or IP alias). When the primary node fails, and the service should be offered by the backup server, the backup server will not only need to start the service or daemon, it will also need to add the proper secondary IP address or IP alias to one of its network cards.

A diagram of this two-node Heartbeat cluster configuration using an Ethernet network to carry the heartbeat packets is shown in Figure 6-2.

In Figure 6-2 the IP address 209.100.100.2 is the primary IP address of the primary server, and it never needs to move to the backup server. The backup server's primary IP address is 209.100.100.5, and this IP address will likewise never need to move to another network card. However, the IP addresses 209.100.100.3 and 209.100.100.4 are each associated with a particular service running on the primary server. If the primary server goes down, these IP addresses need to move to the backup server, as shown in Figure 6-3.

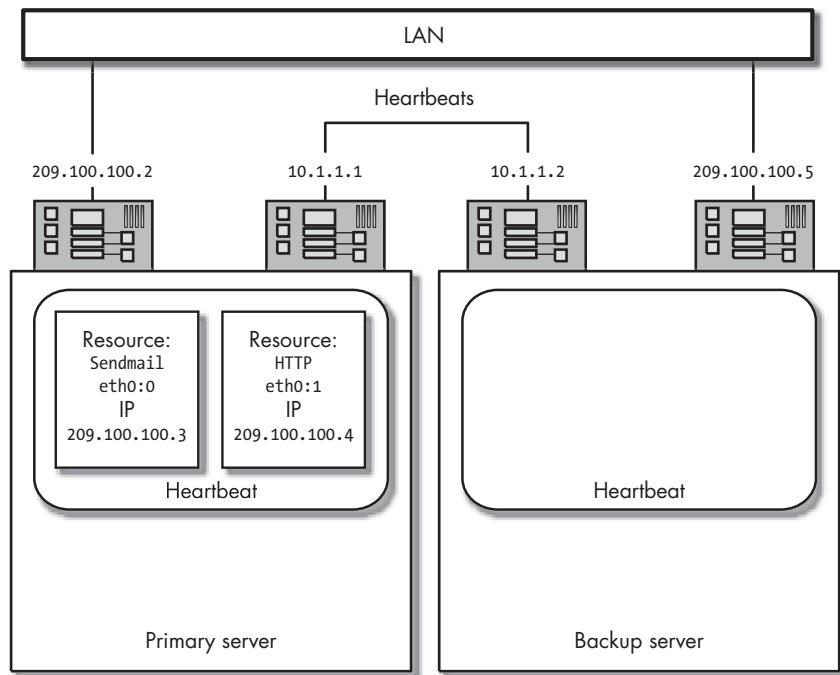


Figure 6-2: A basic Heartbeat configuration

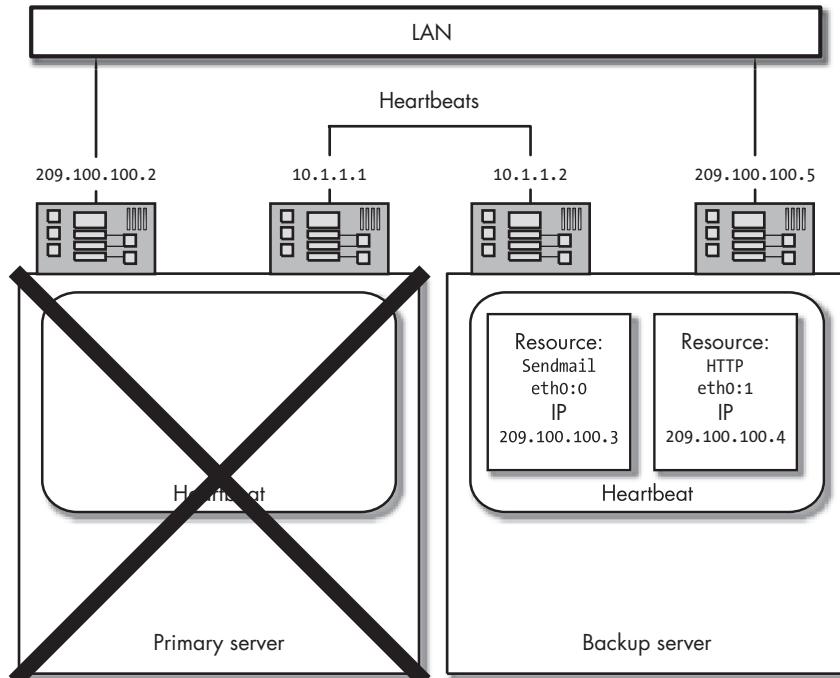


Figure 6-3: The same basic Heartbeat configuration after failure of the primary server

## **Ethernet NIC Device Names**

The Linux kernel assigns the physical Ethernet interfaces names such as eth0, eth1, eth2, and so forth. These names are assigned either at boot time or when the Ethernet driver for the interface is loaded (if you are using a modular kernel), and they are based on the configuration built during the system installation in the file `/etc/modules.conf` (or `/etc/conf.modules` on older versions of Red Hat Linux). You can use the following command to see the NIC driver, interrupt (IRQ) address, and I/O (base) address assigned to each PCI network interface card (assuming it is a PCI card) that was recognized during the boot process:

---

```
#lspci -v | less
```

---

You can then check this information against the interface configuration using the `ifconfig` command:

---

```
#ifconfig -a | less
```

---

This should help you determine which eth number is assigned to a particular physical network card by the kernel eth numbering scheme. (See Appendix C for more information about NICs.)

## **Secondary IP Address Names**

Secondary IP addresses are added after a primary IP address has already been configured for a NIC. The primary IP address can be assigned to a NIC using the `ifconfig` command (this is normally how Linux distributions assign IP addresses at boot time) or by using the `ip` command. Secondary IP addresses, however, can only be added using the `ip` command. When Heartbeat needs to add a secondary IP address to a NIC it uses the script `IPAddr2` (included with the Heartbeat distribution) to run the proper `ip` command.

Both the primary and the secondary IP addresses can be viewed with this command:

---

```
#ip addr sh
```

---

**NOTE** *Secondary IP addresses are not shown by the ifconfig command.*

### **Creating and Deleting Secondary IP Addresses with the ip Command**

Creating and deleting secondary IP addresses in Linux is easy. To create (add) a secondary IP address for the eth0 NIC, use this command:

---

```
#ip addr add 209.100.100.3/24 broadcast 209.100.100.255 dev eth0
```

---

In this example, we are assuming that the `eth0` NIC already has an IP address on the `209.100.100.0` network, and we are adding `209.100.100.3` as an additional (secondary) IP address associated with this same NIC. To view the IP addresses configured for the `eth0` NIC with the previous command, enter this command:

---

```
#ip addr sh dev eth0
```

---

To remove (delete) this secondary IP address, enter this command:

---

```
#ip addr del 209.100.100.3/24 broadcast 209.100.100.255 dev eth0
```

---

**NOTE** *The ip command is provided as part of the IProute2 package. (The RPM package name is iproute.)*

Fortunately, you won't need to enter these commands to configure your secondary IP addresses—Heartbeat does this for you automatically when it starts up and at failover time (as needed) using the `IPAddr2` script.

## **IP Aliases**

As previously mentioned, you only need to use one method for assigning IP addresses under Heartbeat's control: secondary IP addresses or IP aliases. If you are new to Linux, you should use secondary IP addresses as described in the previous sections and skip this discussion of IP aliases.

You add IP aliases to a physical Ethernet interface (that already has an IP address associated with it) by running the `ifconfig` command. The alias is specified by adding a colon and a number to the interface name. The first IP alias associated with the `eth0` interface is called `eth0:0`, the second is called `eth0:1`, and so forth. Heartbeat uses the `IPAddr` script to create IP aliases.

### **Creating and Deleting IP Aliases with the ifconfig Command**

In our previous example, we were using IP address `209.100.100.2` with a network mask of `255.255.255.0` on the `eth0` Ethernet interface. To manually add IP alias `209.100.100.3` to the same interface, use this command:

---

```
#ifconfig eth0:0 209.100.100.3 netmask 255.255.255.0 up
```

---

You can then view the list of IP addresses and IP aliases by typing the following:

---

```
#ifconfig -a
```

---

or simply

---

```
#ifconfig
```

---

This command will produce a listing that looks like the following:

---

```
eth0      Link encap:Ethernet  HWaddr 00:99:5F:0E:99:AB
          inet addr:209.100.100.2  Bcast:209.100.100.255  Mask:255.255.255.0
                  UP BROADCAST RUNNING MTU:1500  Metric:1
                  RX packets:976  errors:0  dropped:0  overruns:0  frame:0
                  TX packets:730  errors:0  dropped:0  overruns:0  carrier:0
                  collisions:0  txqueuelen:100
                  Interrupt:11  Base address:0x1400

eth0:0    Link encap:Ethernet  HWaddr 00:99:5F:0E:99:AB
          inet addr:209.100.100.3  Bcast:209.100.100.255  Mask:255.255.255.0
                  UP BROADCAST RUNNING MTU:1500  Metric:1
                  Interrupt:11  Base address:0x1400
```

---

From this report you can see that the MAC addresses (called `HWaddr` in this report) for `eth0` and `eth0:0` are the same. (The interrupt and base addresses also show that these IP addresses are associated with the same physical network card.) Client computers locally connected to the computer can now use an ARP broadcast to ask “Who owns IP address 209.100.100.3?” and the primary server will respond with “I own IP 209.100.100.3 on MAC address 00:99:5F:0E:99:AB.”

**NOTE** *The `ifconfig` command does not display the secondary IP addresses added with the `ip` command.*

IP aliases can be removed with this command:

---

```
#ifconfig eth0:0 down
```

---

This command should not affect the primary IP address associated with `eth0` or any additional IP aliases (they should remain up).

**NOTE** *Use the preceding command to see whether your kernel can properly support IP aliases. If this command causes the primary IP address associated with this network card to stop working, you need to upgrade to a newer version of the Linux kernel. Also note that this command will not work properly if you attempt to use IP aliases on a different subnet.*<sup>10</sup>

## ***Offering Services***

Once you are sure a secondary IP address or IP alias can be added to and removed from a network card on your Linux server without affecting the primary IP address, you are ready to tell Heartbeat which services it should offer, and which secondary IP address or IP alias it should use to offer the services.

---

<sup>10</sup> The “secondary” flag will not be set for the IP alias if it is on a different subnet. See the output of the command `ip addr` to find out whether this flag is set. (It should be set for Heartbeat to work properly.)

**NOTE** Secondary IP addresses and IP aliases used by highly available services should always be controlled by Heartbeat (in the `/etc/ha.d/haresources` file as described later in this chapter and in the next two chapters). Never use your operating system's ability to add IP aliases as part of the normal boot process (or a script that runs automatically at boot time) on a Heartbeat server. If you do, your server will incorrectly claim ownership of an IP address when it boots. The backup node should always be able to take control of a resource along with its IP address and then reset the power to the primary node without worrying that the primary node will try to use the secondary IP address as part of its normal boot procedure.

### **Gratuitous ARP (GARP) Broadcasts**

As mentioned previously, client computers normally use the Address Resolution Protocol (ARP) to figure out which hardware address owns a particular IP address, and then they store this address in an ARP table. The Heartbeat program uses a little trick, called Gratuitous ARP (GARP) broadcasts, to forcibly update these client computer ARP tables with a new hardware (MAC) addresses when the primary server fails, effectively convincing the client computers to talk to the backup server.<sup>11</sup>

GARP broadcasts<sup>12</sup> are just sneaky ARP broadcasts (broadcasts, remember, are only seen by locally connected nodes). The GARP broadcast asks every node connected to the network, “Who owns this IP address?” when, in fact, the ARP request packet header has a source (or reply) IP address equal to the requested IP address. This forces all nodes connected to the network to update their ARP tables with the new source address.

**NOTE** As of Heartbeat version 0.4.9.1 the `send_arp` program included with Heartbeat uses both ARP request and ARP reply packets when sending GARP (send\_arp version 1.6). If you experience problems with IP address failover on older versions of Heartbeat, try upgrading to the latest version of Heartbeat.

Heartbeat uses the `/usr/lib/heartbeat/send_arp` program (formerly `/etc/ha.d/resource.d/send_arp`) to send these specially crafted GARP broadcasts. You can use this same program to build a script that will send GARP. The following is an example script (called `iptakeover`) that uses `send_arp` to do this.

---

<sup>11</sup> To use the Heartbeat IP failover mechanism with minimal downtime and minimal disruption of the highly available services you should set the ARP cache timeout values on your switches and routers to the shortest time possible. This will increase your ARP broadcast traffic, however, so you will have to find the best trade-off between ARP timeout and increased network traffic for your situation. (Gratuitous ARP broadcasts should update the ARP table entries even before they expire, but if they are missed by your network devices for some reason, it is best to have regular ARP cache refreshes.)

<sup>12</sup> Gratuitous ARPs are briefly described in RFC 2002, “IP Mobility Support.” Also see RFC 826, “Ethernet Address Resolution Protocol.”

---

```
#!/bin/bash
#
# iptakeover script
#
# Simple script to take over an IP address.
#
# Usage is "iptakeover {start|stop|status}"
#
# SENDARP is the program included with the Heartbeat program that
# sends out an ARP request. Send_arp usage is:
#
#
# SENDARP="/usr/lib/heartbeat/send_arp"
#
# REALIP is the IP address for this NIC on your LAN.
#
# ROUTERIP is the IP address for your router.
#
# ROUTER_IP="299.100.100.1"
#
# SECONDARYIP is the first IP alias for a service/resource.
#
# SECONDARYIP1="299.100.100.3"
# or
#IPALIAS1="299.100.100.3"

#
# NETMASK is the netmask of this card.
#
#NETMASK="24"
# or
#NETMASK="255.255.255.0"
#
#BROADCAST="299.100.100.0"
#
# MACADDR is the hardware address for the NIC card.
# (You'll find it using the command "/sbin/ifconfig")
#
#MACADDR="091230910990"
case $1 in
start)
    # Make sure our primary IP is up
    /sbin/ifconfig eth0 $REALIP up
```

```

# Associate the virtual IP address with this NIC
/sbin/ip addr add $SECONDARYIP1/$NETMASK broadcast $BROADCAST dev eth0
# Or, to create an IP alias instead of secondary IP address, use the
# command:
# /sbin/ifconfig eth0:0 $IPALIAS1 netmask $NETMASK up

# Create a new default route directly to the router
/sbin/route add default gw $ROUTER_IP

# Now send out 5 Gratuitous ARP broadcasts (ffffffffffff)
# at two second intervals to tell the local computers to update
# their ARP tables.

$SENDARP -i 2000 -r 5 eth0 $SECONDARYIP1 $MACADDR $SECONDARYIP1 ffffffff
;;
stop)
# Take down the secondary IP address for the service/resource.
/sbin/ip addr del $SECONDARYIP1/$NETMASK broadcast $BROADCAST dev eth0
# or
/sbin/ifconfig eth0:0 down
;;
status)
# We check to see if we own the IPALIAS.
OWN_ALIAS=`ifconfig | grep $SECONDARYIP1`  

if [ "$OWN_ALIAS" != "" ]; then
echo "OK"
else
echo "DOWN"
fi
;;
# End of the case statement.
esac

```

---

**NOTE** *You do not need to use this script. It is included here (and on the CD-ROM) to demonstrate exactly how Heartbeat performs GARPs.*

The important line in the above code listing is marked in boldface.

This command runs `/usr/lib/heartbeat/send_arp` and sends an ARP broadcast (to hexadecimal IP address `ffffffffffff`) with a source and destination address equal to the secondary IP address to be added to the backup server.

You can use this script to find out whether your network equipment supports IP address failover using GARP broadcasts. With the script installed on two Linux computers (and with the MAC address in the script changed to the appropriate address for each computer), you can move an IP address back and forth between the systems to find out whether Heartbeat will be able to do the same thing when it needs to failover a resource.

**NOTE** When using Cisco equipment, you should be able to log on to the router or switch and enter the command `show arp` to watch the MAC address change as the IP address moves between the two computers. Most routers have similar capabilities.

## Resource Scripts

All of the scripts under Heartbeat's control are called resource scripts. These scripts may include the ability to add or remove a secondary IP address or IP alias, or they may include packet-handling rules (see Chapter 2) in addition to being able to start and stop a service. Heartbeat looks for resource scripts in the Linux Standards Base (LSB) standard init directory (`/etc/init.d`) and in the Heartbeat `/etc/ha.d/resource.d` directory.

Heartbeat should always be able to start and stop your resource by running your resource script and passing it the start or stop argument. As illustrated in the `iptakeover` script, this can be accomplished with a simple case statement like the following:

---

```
#!/bin/bash
case $1 in
start)
    commands to start my resource
;;
stop)
    commands to stop my resource
;;
status)
    commands to test if I own the resource
;;
*)
    echo "Syntax incorrect. You need one of {start|stop|status}"
;;
esac
```

---

The first line, `#!/bin/bash`, makes this file a bash (Bourne Again Shell) script. The second line tests the first argument passed to the script and attempts to match this argument against one of the lines in the script containing a closing parenthesis. The last match, `*)`, is a wildcard match that says, “If you get this far, match on anything passed as an argument.” So if the program flow ends up executing these commands, the script needs to complain that it did not receive a start, stop, or status argument.

### ***Status of the Resource***

Heartbeat should always know which computer owns the resource or resources being offered. When you write a script to start or stop a resource, it is important to write the script in such a way that it will accurately determine whether the service is currently being offered by the system. If it is, the script

should respond with the word OK, Running, or running when passed the status argument—Heartbeat requires one of these three responses if the service is running. What the script returns when the service is not running doesn't matter to Heartbeat—you can use DOWN or STOPPED, for example. However, do *not* say not running or not OK for the stopped status. (See the “Using init Scripts as Heartbeat Resource Scripts” section later in this chapter for more information on proposed standard exit codes for Linux.)

**NOTE** *If you want to write temporary scratch files with your resource script and have Heartbeat always remove these files when it first starts up, write your files into the directory /var/lib/heartbeat/rsctmp (on a standard Heartbeat installation on Linux).*

## **Resource Ownership**

So how do you write a script to properly determine if the machine it is running on currently owns the resource? You will need to figure out how to do this for your situation, but the following are two sample tests you might perform in your resource script.

### **Testing for Resource Ownership—Is the Daemon Running?**

Red Hat Linux and SuSE Linux both ship with a program called pidof that can test to see if a daemon is running. The pidof program finds the process identifier (PID) of a running daemon when you give it the name of the program file that was used to start the daemon.

When you run the pidof script from within your resource script you should always pass it the full pathname of the daemon you want to search for in the process table. For example, to determine the PID of the sendmail program, you would use this command:

---

```
#pidof /usr/sbin/sendmail
```

---

If sendmail is running, its PID number will be printed, and pidof exits with a return code of zero. If sendmail is not running, pidof does not print anything, and it exits with a nonzero return code.

**NOTE** *See the /etc/init.d/functions script distributed with Red Hat Linux for an example use of the pidof program.*

A sample bash shell script that uses pidof to determine whether the sendmail daemon is running (when given the status argument) might look like this:

---

```
#!/bin/bash
#
case $1 in
status)
```

```
if /sbin/pidof /usr/sbin/sendmail >/dev/null; then
    echo "OK"
else
    echo "DOWN"
fi
;;
esac
```

---

If you place these commands in a file, such as `/etc/ha.d/resource.d/myresource`, and `chmod` the file to 755, you can then run the script with this command:

---

```
#/etc/ha.d/resource.d/myresource status
```

---

If `sendmail` is running, the script will return the word `OK`; if not, the script will say `DOWN`. (You can stop and start the `sendmail` daemon for testing purposes by running the script `/etc/init.d/myresource` and passing it the `start` or `stop` argument.)

### Testing for Resource Ownership—Do You Own the IP Address?

If you use Heartbeat's built-in ability to automatically failover secondary IP addresses or IP aliases, you do not need to worry about this second test (see the discussion of the `IPaddr2` resource script and the discussion of resource groups in Chapter 8). However, if you are using the `iptakeover` script provided on the CD-ROM, or if you are building a custom configuration that makes changes to network interfaces or the routing table, you may want to check to see if the system really does have the secondary IP address currently configured as part of your script's status test.

To perform this check, the script needs to look at the current secondary IP addresses configured on the system. We can modify the previous script and add the `ip` command to show the secondary IP address for the `eth0` NIC as follows:

---

```
#!/bin/bash
#
case $1 in
status)
    if /sbin/pidof /usr/sbin/sendmail >/dev/null; then
        ipaliasup=`/sbin/ip addr show dev eth0 | grep 200.100.100.3`#
        if [ "$ipaliasup" != "" ]; then
            echo "OK"
            exit 0
        fi
    fi
    echo "DOWN"
;;
esac
```

---

Note that this script will only work if you use secondary IP addresses. To look for an ip alias, modify this line in the script:

---

```
ipaliasup=`/sbin/ip addr show dev eth0 | grep 200.100.100.3`
```

---

to match the following:

---

```
ipaliasup=`/sbin/ifconfig eth0 | grep 200.100.100.3`
```

---

This changed line of code will now check to see if the IP alias 200.100.100.3 is associated with the interface eth0. If so, this script assumes the interface is up and working properly, and it returns a status of OK and exits from the script with a return value of 0.

**NOTE** *Testing for a secondary IP address or an IP alias should only be added to resource scripts in special cases; see Chapter 8 for details.*

### Testing a Resource Script

You can test to make sure the Heartbeat system will work properly with your newly created resource script by running the following command:

---

```
#/usr/lib/heartbeat/ResourceManager status <resource-name>
```

---

where `<resource-name>` is the name of your newly created resource script in the `/etc/rc.d/init.d` or `/etc/ha.d/resource.d` directory. Then tell your shell to display the value of your resource script's return code with this command:

---

```
#echo $?
```

---

The `$?` must be the next command you type after running the script—otherwise the return value stored in the shell variable `?` will be overwritten.

If the return code is 3, it means your script returned something other than OK, Running, or running, which means that Heartbeat thinks that it does not own the resource. If this command returns 0, Heartbeat considers the resource active.

Once you have built a proper resource script and placed it in the `/etc/rc.d/init.d` directory or the `/etc/ha.d/resource.d` directory and thoroughly tested its ability to handle the `start`, `stop`, and `status` arguments, you are ready to configure Heartbeat.

### Using init Scripts as Heartbeat Resource Scripts

Most of the `/etc/init.d` scripts (the scripts used at system boot time by the init process) will already properly handle the `start`, `stop`, and `status` arguments. In fact, the Linux Standard Base Specification (see <http://www.linuxbase.org>)

states that in future releases of Linux, all Linux init scripts running on all versions of Linux that support LSB must implement the following: `start`, `stop`, `restart`, `reload`, `force-reload`, and `status`.

The LSB project also states that if the `status` argument is sent to an init script it should return 0 if the program is running.

For the moment, however, you need only rely on the convention that a `status` command should return the word `OK`, or `Running`, or `running` to standard output (or “standard out”).<sup>13</sup> For example, Red Hat Linux scripts, when passed the `status` argument, normally return a line such as the following example from the `/etc/rc.d/init.d/sendmail` `status` command:

---

```
sendmail (pid 4511) is running...
```

---

Heartbeat looks for the word `running` in this output, and it will ignore everything else, so you do not need to modify Red Hat’s init scripts to use them with Heartbeat (unless you want to also add the test for ownership of the secondary IP address or IP alias).

However, once you tell Heartbeat to manage a resource or script, you need to make sure the script does not run during the normal boot (init) process. You can do this by entering this command:

---

```
#chkconfig --del <scriptname>
```

---

where `<scriptname>` is the name of the file in the `/etc/init.d` directory. (See Chapter 1 for more information about the `chkconfig` command.)

The init script must not run as part of the normal boot (init) process, because you want Heartbeat to decide when to start (and stop) the daemon. If you start the Heartbeat program but already have a daemon running that you have asked Heartbeat to control, you will see a message like the following in the `/var/log/messages` file:

---

```
WARNING: Non-idle resources will affect resource takeback.
```

---

If you see this error message when you start Heartbeat, you should stop all of the resources you have asked Heartbeat to manage, remove them from the init process (with the `chkconfig --del <scriptname>` command), and then restart the Heartbeat daemon (or reboot) to put things in a proper state.

---

<sup>13</sup> Programs and shell scripts can return output to standard output or to standard error. This allows programmers to control, through the use of redirection symbols, where the message will end up—see the “REDIRECTION” section of the bash man page for details.

## Heartbeat Configuration Files

Heartbeat uses three configuration files:

**/etc/ha.d/ha.cf**

Specifies how the heartbeat daemons on each system communicate with each other.

**/etc/ha.d/haresources**

Specifies which server should normally act as the primary server for a particular resource and which server should own the IP address that client computers will use to access the resource.

**/etc/ha.d/authkeys**

Specifies how the Heartbeat packets should be encrypted.

**NOTE** *The ha in these scripts stands for high availability. (The name “haresources” is not related to bunny rabbits.)*

## In Conclusion

This chapter has introduced the Heartbeat package and the theory behind properly deploying it. To ensure client computers always have access to a resource, use the Heartbeat package to make the resource highly available. A highly available resource does not fail even if the computer it is running on crashes.

In Part III of this book I'll describe how to make the cluster load-balancing resource highly available. A cluster load-balancing resource that is highly available is the cornerstone of a cluster that is able to support your enterprise.

In the next few chapters I'll focus some more on how to properly use the Heartbeat package and avoid the typical pitfalls you are likely to encounter when trying to make a resource highly available.

# 7

## A SAMPLE HEARTBEAT CONFIGURATION



The last chapter introduced the three configuration files used to control the operation of Heartbeat: `ha.cf`, `haresources`, and `authkeys`. The recipe in this chapter describes how to use these files on two servers, or nodes, so that Heartbeat can deploy resources in a high-availability configuration.

In a normal Heartbeat configuration, these files will be the same on the primary and the backup server (complex and subtle problems can be introduced into your high-availability configuration if they are not). The recipe in this chapter is a lab exercise that demonstrates how the Heartbeat system starts resources and fails them over to a backup server.

## Recipe

List of ingredients:

- 2 servers running Linux (each with two network interface cards and cables)
- 1 crossover cable (or standard network cables and a mini hub) and/or a serial cable
- 1 copy of the Heartbeat software package (from <http://www.linux-ha.org/download> or the CD-ROM)

## Preparations

In this recipe, one of the Linux servers will be called the *primary server* and the other the *backup server*. Begin by connecting these two systems to each other using a crossover network cable, a normal network connection (through a mini hub or separate VLAN), or a null modem serial cable. (See Appendix C for information on adding NICs and connecting the servers.) We'll use the network connection or serial connection exclusively for heartbeats messages (see Figure 7-1).

If you are using an Ethernet heartbeat connection, assign IP addresses to the primary and backup servers for this network connection using an IP address from RFC 1918. For example, use 10.1.1.1 on the primary server and 10.1.1.2 on the backup server as shown in Figure 7-1. (These are IP addresses that will only be known to the primary and backup servers.)

RFC 1918 defines the following IP address ranges for “private internets”:

- 10.0.0.0 to 10.255.255.255 (10/8 prefix)
- 172.16.0.0 to 172.31.255.255 (172.16/12 prefix)
- 192.168.0.0 to 192.168.255.255 (192.168/16 prefix)

In this recipe, we will use 10.1.1.1 as the primary server's heartbeat IP address and 10.1.1.2 as the secondary server's heartbeat IP address.

Before taking the following steps, be sure that you can ping between these two systems on the network connection that you added (the crossover cable or separate network connection through a mini hub/dedicated VLAN). That is, you should get a response when you enter `ping 10.1.1.2` on the primary server, and `ping 10.1.1.1` on the backup server. (See Appendix C if you need help.)

## Step 1: Install Heartbeat

The CD-ROM included with this book contains the Heartbeat RPMs. You can install these versions using the following commands (though you may instead want to check for the latest versions available for download from <http://www.linux-ha.org/download>).

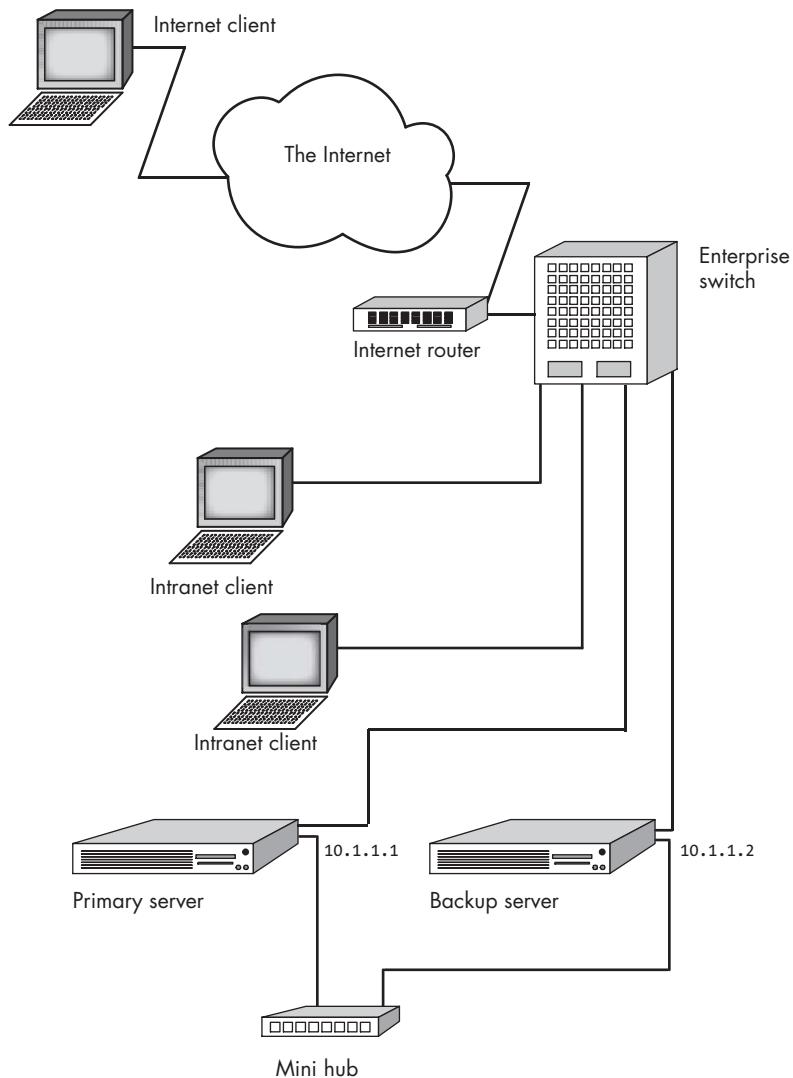


Figure 7-1: The Heartbeat network configuration

---

```
#mount /mnt/cdrom
#rpm -ivh /mnt/cdrom/chapter7/heartbeat-pils-*.rpm1
#rpm -ivh /mnt/cdrom/chapter7/heartbeat-stonith-*.rpm
#rpm -ivh /mnt/cdrom/chapter7/heartbeat-*i386.rpm
```

---

**NOTE** You do not need the source (src) RPM file for this recipe. The ldirectord software is also not required in this chapter—it will be discussed in Chapter 15.

---

<sup>1</sup> The PILS package was introduced with version 0.4.9d of Heartbeat. PILS is Heartbeat's generalized plug-in and interface loading system, and it is required for normal Heartbeat operation.

Once the RPM package finishes installing, you should have an `/etc/ha.d` directory where Heartbeat's configuration files and scripts reside. You should also have a `/usr/share/doc/packages/heartbeat` directory containing sample configuration files and documentation.

### IN CASE OF ERRORS

If you receive error messages about failed dependencies for crypto software when you attempt to install the Stonith RPM, issue the following command to make sure that you have the `openssl` and `openssl-devel` packages installed:

---

```
rpm -q -a | grep openssl
```

---

These packages are only required for Stonith devices that use SSL for communications, but because of the RPM dependency mechanisms, this package will be marked as required for all installations.

For SNMP dependency failures, run this command:

---

```
rpm -q -a | grep snmp
```

---

If the version numbers returned by these commands are lower than the version number required by Stonith, upgrade your packages.<sup>1</sup>

If the installation still complains about failed dependencies, even though you know you have the latest versions installed, you can force the RPM to override this error message with the following command:

---

```
#rpm -ivh --nodeps /usr/local/src/heartbeat/heartbeat-*i386.rpm
```

---

If you receive dependency errors that complain about `lib.so` or `GLIBC`, see Appendix D.

Note: Instead of using the method just described to install Heartbeat, you can use the Heartbeat `src.rpm` package and the `rpm --rebuild` option, as in this command:

---

```
#rpm --rebuild /usr/local/src/heartbeat/heartbeat-*src.rpm
```

---

<sup>1</sup> Note that the SNMP package used to be called `ucd-snmp`, but it has been renamed `net-snmp`.

## Step 2: Configure `/etc/ha.d/ha.cf`

Now you need to tell the heartbeat daemons that you want them to use the new Ethernet network (or the Ethernet crossover or serial cable) to send and receive heartbeat packets.

1. Find the `ha.cf` sample configuration file that the Heartbeat RPM installed by using this command:

---

```
#rpm -qd heartbeat | grep ha.cf
```

---

2. Copy the sample configuration file into place with this command:

---

```
#cp /usr/share/doc/packages/heartbeat/ha.cf /etc/ha.d/
```

---

3. Edit the /etc/ha.d/ha.cf file, and uncomment these lines:

---

```
#udpport      694
#bcast      eth0      # Linux
```

---

**NOTE** *Older versions of Heartbeat use udp instead of bcast.*

For example, to use eth1 to send heartbeat messages between your primary and backup server, the second line would look like this:

---

```
bcast eth1
```

---

If you use two physical network connections to carry heartbeats, change the second line to this:

---

```
bcast eth0 eth1
```

---

If you use a serial connection and an Ethernet connection, uncomment the lines for serial Heartbeat communication:

---

```
serial /dev/ttyS0
baud 19200
```

---

4. Also, uncomment the `keepalive`, `deadtime`, and `initdead` lines so that they look like this:

---

```
keepalive 2
deadtime 30
initdead 120
```

---

The `initdead` line specifies that after the heartbeat daemon first starts, it should wait 120 seconds before starting any resources on the primary server, or making any assumptions that something has gone wrong on the backup server. The `keepalive` line specifies how many seconds there should be between heartbeats (status messages, which were described in Chapter 6), and the `deadtime` line specifies how long the backup server will wait without receiving a heartbeat from the primary server before assuming something has gone wrong. If you change these numbers, Heartbeat may send warning messages that indicate that you have set the values improperly (for example, you might set a `deadtime` too close to the `keepalive` time to ensure a safe configuration).

5. Add the following two lines to the end of the /etc/ha.d/ha.cf file:

---

```
node primary.mydomain.com
node backup.mydomain.com
```

---

Or, you can use one line for both nodes with an entry like this:

---

```
node primary.mydomain.com backup.mydomain.com
```

---

The `primary.mydomain.com` and `backup.mydomain.com` entries should be replaced with the names you have assigned to your two hosts when you installed Linux on them (as returned by the `uname -n` command).

The host names of the primary and backup servers are not usually related to any of the services offered by the servers. For example, the host name of your primary server might be `primary.mydomain.com`, even though it will host a web server called `mystuff.mydomain.com`.

**NOTE** *On Red Hat systems, the host name is specified in the /etc/sysconfig/network file using the `hostname` variable, and it is set at boot time (though it can be changed on a running system by using the `hostname` command).*

## Step 3: Configure /etc/ha.d/haresources

Normally the `/etc/ha.d/haresources` file will contain the names of the resources the primary server should own. Heartbeat generally controls its resources with the normal init script that comes with your distribution, or with a script you build, such as the `iptakeover` script shown in Chapter 6, but for now we will use a very simple test script to see how they work. Follow these steps:

1. Create a script called `test1` in the `/etc/ha.d/resource.d` directory with the following command (this script is also included on the CD-ROM in the `chapter7` subdirectory):

---

```
#vi /etc/ha.d/resource.d/test1
```

---

2. Now press the `I` key to enter insert mode, and then enter the following simple bash script:

---

```
#!/bin/bash
logger $0 called with $1
case "$1" in
  start)
    # Start commands go here
    ;;
  stop)
    # Stop commands go here
    ;;
  status)
    # Status commands go here
    ;;
esac
```

---

The logger command in this script is used to send a message to the syslog daemon. syslog will then write the message to the proper log file based on the rules in the `/etc/syslog.conf` file. (See the man pages for `logger` and `syslog` for more information about custom message logging.)

**NOTE** *The case statement in this script does not do anything. It is included here only as a template for further development of your own custom resource script that can handle the start, stop, and status arguments that Heartbeat will use to control it.*

3. Quit this file and save your changes (press ESC, and then type `:wq`).
4. Make this script executable by entering this command:

---

```
#chmod 755 /etc/ha.d/resource.d/test
```

---

5. Run the script by entering the following command:

---

```
#/etc/ha.d/resource.d/test start
```

---

6. You should be returned to the shell prompt where you can enter this command to see the end of the messages:

---

```
#tail /var/log/messages
```

---

The last message line in the `/var/log/messages` file should look like this:

---

```
[timestamp] localhost root: /etc/ha.d/resource.d/test called with start
```

---

This line indicates that the test resource script was called with the `start` argument and is using the `case` statement inside the script properly.

## ***Configure the haresources File***

You can use this simple test script to see what Heartbeat does with resources by telling it that `test` is a resource. To do so, locate and copy the sample `haresources` file into place with these commands:

---

```
#rpm -qd heartbeat | grep haresources
#cp /usr/share/doc/packages/heartbeat/haresources /etc/ha.d
```

---

Now edit `/etc/ha.d/haresources` (which really just contains commented documentation at this point), and add the following line to the end of the file:

---

```
primary.mydomain.com test
```

---

In this example, `primary.mydomain.com` should be replaced with the name of the primary server as returned by the command `uname -n`.

The `haresources` file tells the Heartbeat program which machine owns a resource. The resource name is really a script in either the `/etc/init.d` directory or the `/etc/ha.d/resource.d` directory. (A copy of this script must exist on both the primary and the backup server.)

Heartbeat uses the `haresources` configuration file to determine what to do when it first starts up. For example, if you specify that the `httpd` script (`/etc/init.d/httpd`) is a resource, Heartbeat will run this script and pass it the `start` command when you start the heartbeat daemon on the primary server. If you tell Heartbeat to stop (with the command `/etc/init.d/heartbeat stop` or `service heartbeat stop`), Heartbeat will run the `/etc/init.d/httpd` script and send it the `stop` command. Therefore, if Heartbeat is not running, the resource daemon (`httpd` in this case) will not be running.

If you use a previously existing init script (from the `/etc/init.d` directory) that was included with your distribution, be sure to tell the system not to start this script during the normal boot process with the `chkconfig --del <scriptname>` command. (See Chapter 1 for a detailed description of the `chkconfig` command.)

**NOTE** *If you're not using Red Hat or SuSE, you should also make sure the script prints `OK`, `Running`, or `running` when it is passed the `status` argument. See Chapter 6 for a discussion of using init scripts as Heartbeat resource scripts.*

The power and flexibility available for controlling resources using the `haresources` file will be covered in depth in Chapter 8.

### RESPAWNING DAEMONS WITH HEARTBEAT

You can tell Heartbeat to start a daemon each time it starts, and then to restart this daemon automatically if the daemon stops running. To do so, use a line like this in the `/etc/ha.d/ha.cf` file:

---

```
respawn userid /usr/bin/mydaemon
```

---

Note, though, that such a daemon (called `/usr/bin/mydaemon` in this example) will not failover to the backup server. If you use this method to start a daemon, it will always be running when Heartbeat is running, so this technique is probably only useful for a daemon that needs to run on the primary and the backup server<sup>1</sup> in conjunction with Heartbeat (the `ipfail` daemon described in Chapter 9 is an example of one such daemon).

For high-availability daemons that would normally be started from init with the `respawn` option, and that need to failover from the primary server to the backup server (such as a serial telecommunications program like Hylafax) you must use the `cl_respawn` program included with the Heartbeat package, or a separate package, such as Daemontools (discussed in Chapter 8) in conjunction with Heartbeat.

---

<sup>1</sup> In a good high-availability configuration, the `ha.cf`, `haresources`, and `authkeys` files will be the same on the primary and the backup server.

## Step 4: Configure /etc/ha.d/authkeys

In this step we will install the security configuration file called `/etc/ha.d/authkeys`. The sample configuration file included with the Heartbeat distribution should be modified to protect your configuration from an attack by doing the following:

1. Locate and copy the sample authkeys file into place with these commands:

---

```
#rpm -qd heartbeat | grep authkeys
#cp /usr/share/doc/packages/heartbeat/authkeys /etc/ha.d
```

---

2. Edit the `/etc/ha.d/authkeys` file so the only uncommented lines look like this:

---

```
auth1
1 sha1 testlab
```

---

**NOTE** *In these lines, don't mistake the number 1 for the letter l. The first line is auth followed by the digit one, and the second line is the digit one followed by sha, the digit one, and then testlab.*

In this example, `testlab` is the digital signature key used to digitally sign the heartbeat packets, and Secure Hash Algorithm 1 (`sha1`) is the digital signature method to be used. Change `testlab` in this example to a password you create, and make sure it is the same on both systems.

3. Make sure the authkeys file is only readable by `root`:

---

```
#chmod 600 /etc/ha.d/authkeys
```

---

**CAUTION** *If you fail to change the security of this file using this chmod command, the Heartbeat program will not start, and it will complain in the `/var/log/messages` file that you have secured this file improperly.*

## Step 5: Install Heartbeat on the Backup Server

Install the RPM packages on the backup server as described in the “Step 1: Install Heartbeat” section of this chapter, or use the `SystemImager` package described in Chapter 5 to clone the primary server. Then execute the following command on the primary server to copy all of the configuration files over to the backup system:

---

```
#scp -r /etc/ha.d backupnode:/etc/ha.d
```

---

Here `backupnode` is the IP address or host name (as defined in `/etc/hosts` on the primary server) of the backup server. The `scp` command is the secure copy command that uses the SSH protocol to copy data between the two

nodes. The `-r` option to `scp` recursively copies all of the files and directories underneath the `/etc/ha.d` directory on the primary server.

The first time you run this command, you will be asked if you are sure you want to allow the connection before the copy operation begins. Also, if you have not inserted the private key for the backup server into the primary server's SSH configuration file for the `root` account, you will be prompted for the `root` password on the backup server. (See Chapter 4 for more information about the SSH protocol.)

## Step 6: Set the System Time

While Heartbeat does not require the primary and the backup servers to have synchronized system clocks, the system times on the two servers should be within a few minutes of each other, or some high-availability services may misbehave under some circumstances. You should manually check and set the system time (with the `date` command) before starting Heartbeat on both systems.

**NOTE** *For a better long-term solution you should synchronize the clocks on both systems using the NTP software.*

## Step 7: Launch Heartbeat

Before starting the heartbeat daemon, run the following `ResourceManager` tests on the primary server and the backup server to make sure you have set up the configuration files correctly:

---

```
#/usr/lib/heartbeat/ResourceManager listkeys ` /bin/uname -n`
```

---

This command looks at the `/etc/ha.d/haresources` file and returns the list of resources (or resource keys) from this file. The only resource we have defined so far is the `test` resource, so the output of this command should simply be the following:

---

```
test
```

---

### ***Launch Heartbeat on the Primary Server***

Once you have the `test` resource script configured properly on the primary server, start the Heartbeat program, and watch what happens in the `/var/log/` messages file.

Start Heartbeat (on the primary server) with one of these commands:

---

```
#/etc/init.d/heartbeat start
```

---

or

---

```
#service heartbeat start
```

---

Then look at the system log again, with this command:

---

```
#tail /var/log/messages
```

---

To avoid retyping this command every few seconds while Heartbeat is coming up, tell the tail command to display new information on your screen as it is appended to the /var/log/messages file by using this command:

---

```
#tail -f /var/log/messages
```

---

Press CTRL-C to break out of this command.

**NOTE** *To change the logging file Heartbeat uses, uncomment the following line in your /etc/ha.d/ha.cf file:*

---

logfile	/var/log/ha-log
---------	-----------------

---

*You can then use the tail -f /var/log/ha-log command to watch what Heartbeat is doing more closely. However, the examples in this recipe will always use the /var/log/messages file.<sup>2</sup> (This does not change the amount of logging taking place.)*

Heartbeat will wait for the amount of time configured for initdead in the /etc/ha.d/ha.cf file before it finishes its startup procedure, so you will have to wait at least two minutes for Heartbeat to start up (initdead was set to 120 seconds in the test configuration file).

When Heartbeat starts successfully, you should see the following message in the /var/log/messages file (I've removed the timestamp information from the beginning of each of these lines to make the messages more readable):

---

```
primary root: test called with status
primary heartbeat[4410]: info: ****
primary heartbeat[4410]: info: Configuration validated. Starting heartbeat
<version>
primary heartbeat[4411]: info: heartbeat: version <version>
primary heartbeat[2882]: WARN: No Previous generation - starting at 13
primary heartbeat[4411]: info: Heartbeat generation: 1
primary heartbeat[4411]: info: UDP Broadcast heartbeat started on port 694
(694) interface eth1
primary heartbeat[4414]: info: pid 4414 locked in memory.
```

---

<sup>2</sup> The recipe for this chapter is using this method so that you can also watch for messages from the resource scripts with a single command (tail -f /var/log/messages).

<sup>3</sup> The Heartbeat “generation” number is incremented each time Heartbeat is started. If Heartbeat notices that its partner server changes generation numbers, it can take the proper action depending upon the situation. (For example, if Heartbeat thought a node was dead and then later receives another Heartbeat from the same node, it will look at the generation number. If the generation number did not increment, Heartbeat will suspect a split-brain condition and force a local restart.)

```
primary heartbeat[4415]: info: pid 4415 locked in memory.
primary heartbeat[4416]: info: pid 4416 locked in memory.
primary heartbeat[4416]: info: Local status now set to: 'up'
primary heartbeat[4411]: info: pid 4411 locked in memory.
primary heartbeat[4416]: info: Local status now set to: 'active'
primary logger: test called with status
primary last message repeated 2 times
primary heartbeat: info: Acquiring resource group: primary.mydomain.com test
primary heartbeat: info: Running /etc/init.d/test start
primary logger: test called with start
primary heartbeat[4417]: info: Resource acquisition completed.
primary heartbeat[4416]: info: Link primary.mydomain.com:eth1 up.
```

---

The test script created earlier in the chapter does not return the word `OK`, `Running`, or `running` when called with the `status` argument, so Heartbeat assumes that the daemon is not running and runs the script with the `start` argument to acquire the test resource (which doesn't really do anything at this point). This can be seen in the preceding output. Notice this line:

---

```
primary.mydomain.com heartbeat[2886]: WARN: node backup.mydomain.com: is dead
```

---

Heartbeat warns you that the backup server is dead because the heartbeat daemon hasn't yet been started on the backup server.

**NOTE** *Heartbeat messages never contain the word `ERROR` or `CRIT` for anything that should occur under normal conditions (even during a failover). If you see an `ERROR` or `CRIT` message from Heartbeat, action is probably required on your part to resolve the problem.*

### ***Launch Heartbeat on the Backup Server***

Once Heartbeat is running on the primary server, log on to the backup server, and start Heartbeat with this command:

---

```
# /etc/init.d/heartbeat start
```

---

The `/var/log/messages` file on the backup server should soon contain the following:

---

```
backup heartbeat[4650]: info: ****
backup heartbeat[4650]: info: Configuration validated. Starting heartbeat
<version>
backup heartbeat[4651]: info: heartbeat: version <version>
backup heartbeat[4651]: info: Heartbeat generation: 9
backup heartbeat[4651]: info: UDP Broadcast heartbeat started on port 694
(694) interface eth1
backup heartbeat[4654]: info: pid 4654 locked in memory.
backup heartbeat[4655]: info: pid 4655 locked in memory.
backup heartbeat[4656]: info: pid 4656 locked in memory.
backup heartbeat[4656]: info: Local status now set to: 'up'
```

---

```
backup heartbeat[4651]: info: pid 4651 locked in memory.
backup heartbeat[4656]: info: Link backup.mydomain.com:eth1 up.
backup heartbeat[4656]: info: Node primary.mydomain.com: status active
backup heartbeat: info: Running /etc/ha.d/rc.d/status status
backup heartbeat: info: Running /etc/ha.d/rc.d/ifstat ifstat
backup heartbeat: info: Running /etc/ha.d/rc.d/ifstat ifstat
backup heartbeat[4656]: info: No local resources [/usr/lib/heartbeat/
ResourceManager listkeys backup.mydomain.com]
backup.mydomain.com heartbeat[4656]: info: Resource acquisition completed.
```

---

Notice in this output how Heartbeat declares that this machine (the backup server) does not have any local resources in the `/etc/ha.d/haresources` file. This machine will act as a backup server and sit idle, simply listening for heartbeats from the primary server until that server fails. Heartbeat did not need to run the test script (`/etc/ha.d/resource.d/test`). The Resource acquisition completed message is a bit misleading because there were no resources for Heartbeat to acquire.

**NOTE** *All resource script files that you refer to in the haresources file must exist and have execute permissions<sup>4</sup> before Heartbeat will start.*

### ***Examining the Log Files on the Primary Server***

Now that the backup server is up and running, Heartbeat on the primary server should be detecting heartbeats from the backup server. At the end of the `/var/log/messages` file you should see the following:

```
primary heartbeat[2886]: info: Heartbeat restart on node backup.mydomain.com
primary heartbeat[2886]: info: Link backup.mydomain.com:eth2 up.
primary heartbeat[2886]: info: Node backup.mydomain.com: status up
primary heartbeat: info: Running /etc/ha.d/rc.d/status status
primary heartbeat: info: Running /etc/ha.d/rc.d/ifstat ifstat
primary heartbeat[2886]: info: Node backup.mydomain.com: status active
primary heartbeat: info: Running /etc/ha.d/rc.d/status status
```

---

If the primary server *does not* automatically recognize that the backup server is running, check to make sure that the two machines are on the same network, that they have the same broadcast address, and that no firewall rules are filtering out the packets. (Use the `ifconfig` command on both systems and compare the bcast numbers; they should be the same on both.) You can also use the `tcpdump` command to see if the heartbeat broadcasts are reaching both nodes:

---

```
#tcpdump -i all -n -p udp port 694
```

---

<sup>4</sup> Script files are normally owned by user `root`, and they have their permission bits configured using a command such as this: `chmod 755 <scriptname>`.

This command should capture and display the heartbeat broadcast packets from either the primary or the backup server.

## Stopping and Starting Heartbeat

Now is a good time to practice starting and stopping Heartbeat on both the primary and backup machines to see what happens.

Go ahead and stop Heartbeat on the primary server with one of these commands:

---

```
#/etc/init.d/heartbeat stop
```

---

or

---

```
#service heartbeat stop
```

---

You should see the backup server declare that the primary server has failed, and then it should run the `/etc/ha.d/resource.d/test` script with the `start` argument. The `/var/log/messages` file on the backup server contain messages like these:

```
backup.mydomain.com heartbeat[5725]: WARN: node primary.mydomain.com: is dead
backup.mydomain.com heartbeat[5725]: info: Link primary.mydomain.com:eth1
dead.
backup.mydomain.com heartbeat: info: Running /etc/ha.d/rc.d/status status
backup.mydomain.com heartbeat: info: Running /etc/ha.d/rc.d/ifstat ifstat
backup.mydomain.com heartbeat: info: Taking over resource group test
*** /etc/ha.d/resource.d/test called with status
backup.mydomain.com heartbeat: info: Acquiring resource group:
primary.mydomain.
com test
backup.mydomain.com heartbeat: info: Running /etc/ha.d/resource.d/test start
*** /etc/ha.d/resource.d/test called with start
backup.mydomain.com heartbeat: info: mach_down takeover complete.
```

---

The `/etc/ha.d/resource.d/test` resource or script was first called with the `status` argument and then with the `start` argument to complete the failover.

Once that's done, try starting Heartbeat on the primary server again, and watch what happens. The test script on the backup server should be called with the `stop` argument, and it should be called with the `start` argument on the primary server.<sup>5</sup>

Once you have performed these tests, you may also want to test other means of initiating a failover (such as unplugging the power or network connection on the primary server).

---

<sup>5</sup> This assumes you have enabled `auto_fallback`, which will be discussed in Chapter 9.

**NOTE** *No failover will take place if heartbeat packets continue to arrive at the backup server. So, if you specified two paths for heartbeats in the /etc/ha.d/ha.cf file (a null modem serial cable and a crossover Ethernet cable, for example) unplugging only one of the physical paths will not cause a failover—both must be disconnected before a failover will be initiated by the backup server.*

## Monitoring Resources

Heartbeat currently does not monitor the resources it starts to see if they are running, healthy, and accessible to client computers. To monitor these resources, you need to use a separate software package called Mon (discussed in Chapter 17).

With this in mind, some system administrators are tempted to place Heartbeat on their production network and to use a single network for both heartbeat packets and client computer access to resources. This sounds like a good idea at first, because a failure of the primary computer's connection to the production network means client computers cannot access the resources, and a failover to the backup server *may* restore access to the resources. However, this failover may also be unnecessary (if the problem is with the network and not with the primary server), and it may not result in any improvement. Or, worse yet, it may result in a split-brain condition.

For example, if the resource is a shared SCSI disk, a split-brain condition will occur if the production network cable is disconnected from the backup server. This means the two servers incorrectly think that they have exclusive write access to the same disk drive, and they may then damage or destroy the data on the shared partition. Heartbeat should only failover to the backup server—according to the high-availability design principles used to develop it—if the primary server is no longer healthy. Using the production network as the one and only path for heartbeats is a bad practice and should be avoided.

**NOTE** *Normally the only reason to use the Heartbeat package is to ensure exclusive access to a resource running on a single computer. If the service can be offered from multiple computers at the same time (and exclusive access is not required) you should avoid the unnecessary complexity of a Heartbeat high-availability failover configuration.*

## In Conclusion

This chapter used a “test” resource script to demonstrate how Heartbeat starts a resource and fails it over to a backup server when the primary server goes down. We've looked at a sample configuration using the three configuration files /etc/ha.d/ha.cf, /etc/ha.d/haresources, and /etc/ha.d/authkeys. These configuration files should always be the same on the primary and the backup servers to avoid confusion and errant behavior of the Heartbeat system.

The Heartbeat package was designed to failover all resources when the primary server is no longer healthy. Best practices therefore dictate that we should build multiple physical paths for heartbeat status messages in order to avoid unnecessary failovers. Do not be tempted to use the production network as the only path for heartbeats between the primary and backup servers.

The examples in this chapter did not contain an IP address as part of the resource configuration, so no Gratuitous ARP broadcasts were used. Neither did we test a client computer's ability to connect to a service running on the servers.

Once you have tested resource failover as described in this chapter (and you understand how a resource script is called by the Heartbeat program), you are ready to insert an IP address into the `/etc/ha.d/haresources` file (or into the `iptakeover` script introduced in Chapter 6). This is the subject of Chapter 8, along with a detailed discussion of the `/etc/ha.d/haresources` file.

# 8

## HEARTBEAT RESOURCES AND MAINTENANCE



This chapter describes how to use the high-availability resources file, `haresources`, to control resources on a pair of servers<sup>1</sup> running Heartbeat. We will also explore some of the common maintenance tasks required to keep the Heartbeat high-availability system functioning properly.

### The `Haresources` File Syntax

The `/etc/ha.d/haresources` file must be the same on both the primary and the backup Heartbeat servers.

Each line in the `haresources` file usually contains the following:

- The name of the server where the resource should normally run (the primary server), followed by a space or tab.

---

<sup>1</sup> In this book, we are only concerned with Heartbeat's ability to failover a resource from a primary server to a backup server.

- An (optional) IP alias that Heartbeat should add to the system before launching the resource, followed by a space. (The IP alias definition may include a network subnet mask and a broadcast address separated from each other by the forward slash (/) character.)
- A resource script (the script used to start and stop the resource) located in either the /etc/init.d or the /etc/ha.d/resource.d directory.<sup>2</sup> If arguments need to be passed to the resource script, they are added after two colons and are separated from each other by two colons.

Additional resource scripts can be added to the line using the space character as a separator.

**NOTE** *If you need to create a haresources line that is longer than the line of text that fits on your screen, you can add the backslash character (\) to indicate that the haresources entry continues on the next line.*

A simplified summary of this syntax, for a single line with two resources, each with two arguments, looks like this:

---

```
primary-server [IPaddress] resource1[::arg1::arg2] [resource2[::arg1::arg2]]
```

---

In practice, this line might look like the following on a server called primary.mydomain.com running sendmail and httpd on IP address 209.100.100.3:

---

```
primary.mydomain.com 209.100.100.3 sendmail httpd
```

---

Let's examine each element of the haresources file in more detail.

### ***Haresources File Syntax: Primary-Server Name***

The primary-server name you enter at the start of the haresources line should match one of the server names you've already specified in the /etc/ha.d/ha.cf file. It should also match the name returned by the uname -n command on the primary server.

### ***Haresources File Syntax: IP Alias***

Although it is not required, an IP alias<sup>3</sup> is usually specified in the haresources file. This IP alias can then be offered from either the primary or the backup server, depending upon which system is healthy. For example:

---

```
primary.mydomain.com 209.100.100.3
```

---

<sup>2</sup> Recall from Chapter 6 that this is the system init directory (/etc/init.d, /etc/rc.d/init.d, /sbin/init.d, /usr/local/etc/rc.d or /etc/rc.d) and the Heartbeat resource directory (/etc/ha.d/resource.d).

<sup>3</sup> IP aliases were introduced in Chapter 6.

Heartbeat will add 209.100.100.3 as an IP alias to one of the existing NICs connected to the system and send Gratuitous ARP<sup>4</sup> broadcasts out of this NIC to the locally connected computers when it first starts up. It will only do this on the backup server if the primary server goes down.

Actually, when Heartbeat sees an IP address in the `haresources` file, it runs the resource script in the `/etc/ha.d/resource.d` directory called `IPAddr` and passes it the requested IP address as an argument. The `/etc/ha.d/resource.d/IPAddr` script then calls the program included with the Heartbeat package, `findif` (find interface), and passes it the IP alias you want to add. This program then automatically selects the physical NIC that this IP alias should be added to, based on the kernel's network routing table.<sup>5</sup> If the `findif` program cannot locate an interface to add the IP alias to, it will complain in the `/var/log/messages` file with a message such as the following:

---

```
heartbeat: ERROR: unable to find an interface for 200.100.100.3
```

---

**NOTE** *You cannot place the primary IP address for the interface in the `haresources` file. Heartbeat can add an IP alias to an existing interface, but it cannot be used to bring up the primary IP address on an interface.*

### **Heartbeat's Automated Network Interface Card Selection Process**

Heartbeat uses the `findif` program to select which NIC the IP alias you specify in the `haresources` file will be added by comparing the IP alias to each of the destination addresses listed in your kernel's network routing table. As described in Chapter 2, you can view this routing table with the `route -n` command. For example:

---

```
#route -n
Kernel IP routing table
Destination     Gateway         Genmask        Flags Metric  Ref  Use     Iface
200.100.100.2   0.0.0.0        255.255.255.0  U      0      0      0      eth1
10.1.1.0        0.0.0.0        255.255.255.0  U      0      0      0      eth2
127.0.0.0        0.0.0.0        255.0.0.0      U      0      0      0      lo
0.0.0.0          200.100.100.1  0.0.0.0      UG     0      0      0      eth1
```

---

**NOTE** *The output of this command is based on entries the kernel stores in the `/proc/net/route` file that is created each time the system boots using the `route` commands in the `/etc/init.d/network` script on a Red Hat system. See Chapter 2 for an introduction to the kernel network routing table.*

When `findif` is able to match the network portion of the destination address in the routing table with the network portion of the IP alias from the `haresources` file, it returns the interface name (eth1, for example) associated

---

<sup>4</sup> GARP broadcasts were also introduced in Chapter 6.

<sup>5</sup> See "Routing Packets with the Linux Kernel" in Chapter 2.

with the destination address to the `IPAddr` script. The `IPAddr` script then adds the IP alias to this local interface and adds another entry to the routing table to route packets destined for the IP alias to this local interface. So the routing table, after the 209.100.100.3 IP alias is added, would look like this:

---

Kernel IP routing table							
Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
200.100.100.2	0.0.0.0	255.255.255.0	U	0	0	0	eth1
200.100.100.3	0.0.0.0	255.255.255.0	U	0	0	0	eth1
10.1.1.0	0.0.0.0	255.255.255.0	U	0	0	0	eth2
127.0.0.0	0.0.0.0	255.0.0.0	U	0	0	0	lo
0.0.0.0	200.100.100.1	0.0.0.0	UG	0	0	0	eth1

---

The `IPAddr` script has executed the command `route add -host 209.100.100.3 dev eth1`.

Finally, to complete the process of adding the IP alias to the system, the `IPAddr` script sends out five gratuitous ARP broadcasts to inform locally connected computers that this IP alias is now associated with this interface.

**NOTE** *If more than one entry in the routing table matched the IP alias being added, the `findif` program will use the metric entry in the routing table to select the interface with the fewest hops (the lowest metric).*

### Finding the Right Network

To compare the network portion of the IP alias to the network portion of the “destination” address entry in the routing table, `findif` needs to know which portion of the address represents a network and which portion represents a node. In other words, it needs to know which network mask to apply to both the destination address in the routing table and the IP alias from the `haresources` file before it can determine whether the network portions of the two addresses match.

The `findif` program will use the network mask for each entry in the routing table and apply it to *both* the routing table entry and the IP alias being requested to see if they match. If the two network portions of the addresses match, they are on the same network, and `findif` knows to add the IP alias being requested to the interface associated with this routing table entry.

The `findif` program will not use the default route specified in your routing table (the entry with a destination address of 0.0.0.0) unless you specify a network mask in your `haresources` file. For example, the entry:

---

```
primary.mydomain.com 209.100.100.3/24 myresource
```

---

says that Heartbeat should use a network mask of 255.255.255.0. It also says that if no other entry in the routing table with its associated network mask applied to it matches this address, the default route in the routing table should be used.

However, under normal circumstances your routing table has an entry that will match your IP alias correctly without the need to consult the default route, so you will probably never need to enter the network mask in the `haresources` file. In the above routing table, for example, before Heartbeat added the IP alias, the first entry looked like this:

---

Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
200.100.100.2	0.0.0.0	255.255.255.0	U	0	0	0	eth1

---

This entry matches the `209.100.100.3` IP alias once the `255.255.255.0` netmask from this routing table entry is applied to both addresses for the comparison (both addresses are on the `209.100.100` network). So the correct interface (`eth1` in this case) is selected even though the default route was not used in the interface selection process.

### ***Specifying a Network Interface Card***

You can avoid this auto-selection process by specifying the interface Heartbeat should use in the `haresources` file with an entry like this:

---

```
primary.mydomain.com 209.100.100.3/24/eth0/209.100.100.255
```

---

This entry uses the following syntax:

---

```
primary-server IPalias/number-of-netmask-bits/interface-name/broadcast-address
```

---

Thus, in this example, the IP alias is `209.100.100.3` with a 24-bit netmask (equivalent to a network mask of `255.255.255.0`) on network interface card `eth0`, using a broadcast address of `209.100.100.255`.

To specify this as the IP alias and interface to be used for the `httpd` daemon, enter the following line:

---

```
primary.mydomain.com 209.100.100.3/24/eth0/209.100.100.255 httpd
```

---

With this entry in the `haresources` file, Heartbeat will always use the `eth0` interface for the `209.100.100.3` IP alias when adding it to the system for the `httpd` daemon to use.<sup>6</sup>

### ***Customizing IP Address Takeover with the `iptakeover` Script***

If you need to modify the routing table as part of the process of taking over an IP address, then you may want to use the `iptakeover` script described in Chapter 7 to perform Gratuitous ARP broadcasts yourself. To use this script, add a line like the following to the `haresources` file.

---

<sup>6</sup> Note that the resource daemon (`httpd` in this case) may also need to be configured to use this interface or this IP address as well.

---

```
primary.mydomain.com iptakeover myresource
```

---

Heartbeat will then run `/etc/ha.d/resource.d/iptakeover` status followed by `/etc/ha.d/resource.d/myresource start`. This makes it possible for you to decide exactly which interface you want your IP alias to appear on, to modify the routing table if need be, and to perform the Gratuitous ARP broadcasts from a single script. However, under most circumstances this is not required and should be avoided if possible—specify an IP address in the `haresources` file and let Heartbeat do all of this work for you.<sup>7</sup>

## ***The Haresources File Syntax: Resources***

Each line in the `haresources` file can contain one or more resource script names. The resources are separated by a space. Arguments can be passed to the resource scripts using two colons between the arguments. For example, if you need Heartbeat to send a special argument (let's say `FILE1`) to your resource script (before the word `start`, `status` or `stop`), you would use the syntax:

---

```
primary.mydomain.com myresource::FILE1
```

---

Assuming you add this line to the `haresources` file on both the primary and the backup server, Heartbeat will run<sup>8</sup> `/etc/ha.d/resource.d/myresource FILE1 start` when it first starts on the primary server, and then again on the backup server when the primary server fails. When the resource needs to be “released” or stopped, Heartbeat will run the script with the command `/etc/ha.d/resource.d/myresource FILE1 stop`.

If we wanted to combine our `iptakeover` script with the `myresource` script and its `FILE1` argument, we would use the line:

---

```
primary.mydomain.com iptakeover myresource::FILE1
```

---

To send your resource script several arguments, enter them all on the same line after the script name with each argument separated by a pair of colons. For example, to send the `myresource` script the arguments `FILE1`, `UNAME=JOHN`, and `L3`, your `haresources` entry would look like this:

---

```
primary.mydomain.com iptakeover myresource::FILE1::UNAME=JOHN::L3
```

---

**NOTE** *The `haresources` syntax is also documented online at <http://wiki.trick.ca/linux-ha/HeartbeatResourceAgent>.*

---

<sup>7</sup> Also, as we'll see in Part III, you'll want to leave IP alias assignment under Heartbeat's control so Ldirectord can failover to the backup load balancer.

<sup>8</sup> This assumes the script `myresource` is located in the `/etc/ha.d/resource.d` directory and not in the `/etc/init.d` directory—in which case the command would be `/etc/rc.d/resource.d FILE1 start`.

## Resource Groups

Until now, we have only described resources as independent entries in the `haresources` file. In fact, Heartbeat considers all the resources specified on a single line as one *resource group*. When Heartbeat wants to know if a resource group is running, it asks only the first resource specified on the line. Multiple resource groups can be added by adding additional lines to the `haresources` file.

For example, if the `haresources` file contained an entry like this:

---

```
primary.mydomain.com iptakeover myresource::FILE1::UNAME=JOHN::L3
```

---

only the `iptakeover` script would be called to ask for a status when Heartbeat was determining the status of this resource group. If the line looked like this instead:

---

```
primary.mydomain.com myresource::FILE1::UNAME=JOHN::L3 iptakeover
```

---

Heartbeat would run the following command to determine the status of this resource group (assuming the `myresource` script was in the `/etc/ha.d/resource.d` directory):

---

```
/etc/ha.d/resource.d/myresource FILE1 UNAME=JOHN L3 status
```

---

**NOTE** *If you need your daemon to start before the IP alias is added to your system, enter the IP address after the resource script name with an entry like this:*

---

```
primary.mydomain.com myresource IPAddr::200.100.100.3
```

---

## Resource Script Arguments and Resource Groups

To combine Heartbeat's ability to send arguments to a script with its ability to create multiple resource groups, you could, for example, write one script that started both SERVICE-A and SERVICE-B based upon the argument it was passed. For example, let's call this combined resource script `resAB` and assume it can handle the argument SERVICE-A or SERVICE-B followed by the word start, stop, or status to control both daemons. You could then create `haresources` entries like this:

---

```
primary.mydomain.com resAB::SERVICE-A
primary.mydomain.com resAB::SERVICE-B
```

---

Using this `haresources` file entry, when Heartbeat wanted to know if these resource groups were active, it would run the commands:

---

```
/etc/ha.d/resource.d/resAB SERVICE-A status
/etc/ha.d/resource.d/resAB SERVICE-B status
```

---

and it would start the resource group by executing:

---

```
/etc/ha.d/resource.d/resAB SERVICE-A start  
/etc/ha.d/resource.d/resAB SERVICE-B start
```

---

## Load Sharing with Heartbeat

Using Heartbeat and two computers, we can offer one daemon, or service, from the primary server and then offer a *different* service from the backup server. If either server fails, the other one will start offering both services.

This is a form of load sharing called an active-active server configuration, but to use it you will have to ensure that each system has the processing power and ability to handle the network load that will allow it to offer both services in the event of a failure. This configuration, however, is much more difficult to administer and support in a production environment. Neither server can go down for maintenance or upgrades without causing a failover of at least one service. Figure 8-1 shows a sample diagram of this type of Heartbeat configuration.

The two-line `haresources` entry used to create the configuration shown in Figure 8-1 (again, the primary and the backup server should always have identical `haresources` files) looks like this:

---

```
primary.mydomain.com 209.100.100.3 sendmail  
backup.mydomain.com 209.100.100.4 httpd
```

---

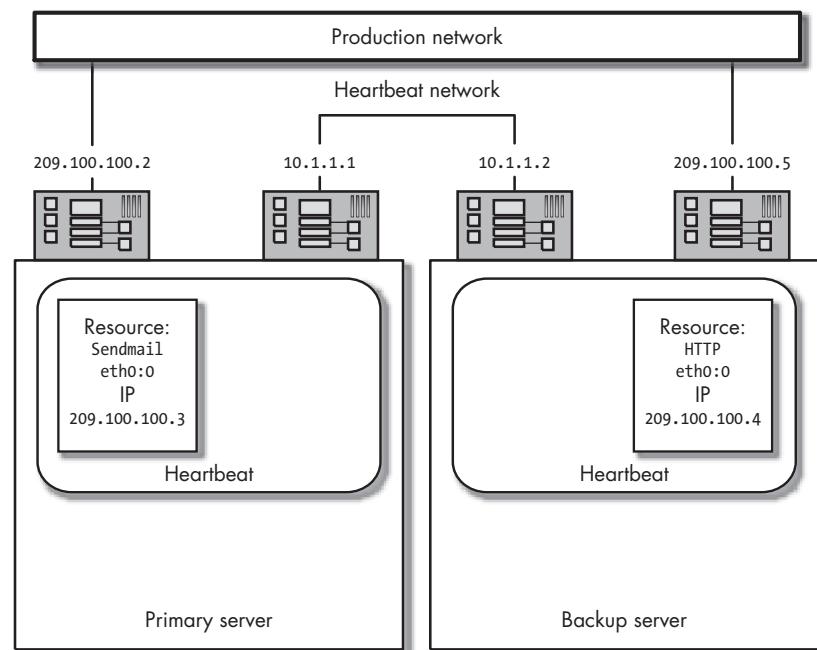


Figure 8-1: Heartbeat active-active configuration

Once Heartbeat is running on both servers, the primary.mydomain.com computer will offer sendmail at IP address 209.100.100.3, and the backup.mydomain.com computer will offer httpd at IP address 209.100.100.4. If the backup server fails, the primary server will perform Gratuitous ARP broadcasts for the IP address 209.100.100.4 and run httpd with the start argument. (The names “primary” and “backup” are really meaningless in this configuration because both computers are acting as both a primary and a backup server.) Client computers would always look for the sendmail resource at 209.100.100.3 and the http resource at 209.100.100.4, and even if one of these servers went down, the resource would still be available once Heartbeat started the resource and moved the IP alias over to the other computer.

**NOTE** *This configuration is more difficult to administer than an active-standby configuration, because you will always be making changes on a “live” system unless you failover all resources so they run on a single server before you do your maintenance.*

This configuration is also not recommended when using local data (data stored on a locally attached disk drive of either server) because complex data replication and synchronization methods must be used.<sup>9</sup>

### ***Load Sharing with Heartbeat: Round-Robin DNS***

But what if you wanted to offer just one resource from both computers and have them share the work? This is the goal of the cluster described in this book, and it will be possible to attain that goal using load-balancing software described in Part III. However, for now we can achieve a simple form of load balancing using round-robin DNS.

One feature of the Domain Name System called round-robin DNS is that it allows you to offer one service on two (or more) IP addresses. For example, the host name (or web URL) is first resolved at IP address 209.100.100.4, then at 209.100.100.3, and then back at 209.100.100.4 again, and so on in round-robin fashion. The DNS (BIND version 4.9.3 or later) entry for your server might look like this:

---

```
;Round-robin entry for www.mydomain.com
www.mydomain.com    IN  A   209.100.100.3
www.mydomain.com    IN  A   209.100.100.4
```

---

with reverse address entries for the 209.100.100.3 and 209.100.100.4 IP addresses that look like this:

---

```
3      IN PTR www.mydomain.com
4      IN PTR www.mydomain.com
```

---

---

<sup>9</sup> See Chapter 7, “Failover Configurations and Issues,” in *Blueprints for High Availability* by Evan Marcus and Hal Stern.

The entry in your `haresources` file for this type of configuration would then look like this:

---

```
primary.mydomain.com 209.100.100.3 httpd
backup.mydomain.com 209.100.100.4 httpd
```

---

## ***Problems with Round Robin DNS Load Balancing***

When using round-robin DNS, the two servers would, in theory, each get half of the client requests, and Heartbeat would ensure that both IP addresses are available even if one of the servers goes down. Most client computers, however, have a name services caching daemon or NSCD that will cause them to remember (at least for a while) an IP address once they learn it. Storing this IP address on the client computer reduces the need for the client computer to repeatedly ask, “What is the IP address for this host name?” and helps improve the chances that client computers will not enter into a dialog (such as an HTTPS secure transaction) with one web server only to end up improperly sending a response (such as a credit card number) to another web server.

Caching IP address-to-host name can cause a cache-only DNS server on the Internet to respond to client requests for an IP address with a non-authoritative reply using only one of the IP addresses. This intervening, cache-only DNS server effectively blocks the round-robin DNS replies from your authoritative DNS server.

You can try to stop this behavior by setting a very low time to live (TTL) value for your DNS replies. Once the amount of time specified in your time to live entry elapses, the intervening DNS server should drop the IP address-to-host name mapping it has stored in its memory and ask your authoritative DNS server once again for the proper IP address.

But there’s a problem: If everyone on the Internet started using a one-second DNS TTL value, every client computer on the Internet would effectively end up asking only the authoritative DNS server for the correct IP address when doing a host name-to-IP address resolution. This would circumvent the design of the DNS system whereby intervening DNS servers cache this information and thus reduce both the time it takes to resolve a host name and the amount of DNS traffic that has to be passed around on the Internet. (A very low TTL value for DNS entries means a more heavily loaded authoritative DNS server.<sup>10</sup>)

Using round-robin DNS and this type of heartbeat configuration for load sharing, however, lets you locate your two servers at two different physical locations. In the event of a true disaster, one of the servers would be able to take over both (or all) IP addresses, and once the routers on the Internet figured out where the IP address had moved to, eventually allow the client computers to continue to connect to your web server at its new location.

---

<sup>10</sup> To prevent this problem from ever happening, many DNS clients and servers ignore small TTL values and use cached information anyway.

**NOTE** *This configuration is very susceptible to a split-brain condition. If exclusive access to resources is required, automatic failover mechanisms that require heartbeats to traverse a WAN or a public network should not be used.*

## **Wide-Area Load Balancing**

Now, what if the client computer on the Internet is much closer to one of the two web servers offering the same web page and it accidentally ends up with the IP address that happens to be on a server located on the other side of the world? It would make sense to offer your resources from the servers closest to the Internet client and only force the client to route to another server in the event of a failure or system crash.

This is called *wide-area load balancing* or *globally distributed content* and can be accomplished with an open source program for Linux called Super Sparrow. A server running Super Sparrow will examine the client's source address and determine whether the client computer would be better off talking to a different server with synchronized content that is closer to the client computer. If so, the client computer's request is redirected to the closest available server. (For more information, see the Super Sparrow website at <http://www.supersparrow.org>.)

## **Operator Alerts: Audible Alarm**

To cause an alarm to sound when the backup server has to take over for the primary server, use a `haresources` entry like this:

---

```
primarynode AudibleAlarm::primarynode
```

---

This entry says that the host named `primarynode` should normally “own” the resource `AudibleAlarm`, but that the `AudibleAlarm` should never sound on the primary server. The `AudibleAlarm` resource, or script, allows you to specify a list of host names that should never sound an alarm. In this case, we are telling the `AudibleAlarm` script not to run or sound on the `primarynode`. When a failover occurs, Heartbeat running on the backup server *will* sound the alarm (an audible beep every one second).

**NOTE** *These haresources entries rely on the scripts /etc/ha.d/resource.d/AudibleAlarm and /etc/ha.d/resource.d/MailTo. These scripts are located in the chapter8 subdirectory on the CD-ROM. You can also download them from the Linux-ha CVS repository (see the heartbeat/resource.d directory for additional scripts).*

The `AudibleAlarm` script can also be modified to flash the floppy drive light if you have installed the `fdutils` packages. The `fdutils` package contains a utility called `floppycontrol`. You can easily download and compile the `fdutils` package (download the tar file from <http://fdutils.linux.lu>, then run `./configure`, then `make`) and uncomment the lines in the `/etc/ha.d/resource.d/AudibleAlarm` script to make the floppy drive light flash.

## Operator Alerts: Email Alerts

To send an email alert, use the `MailTo` resource script with a `haresources` entry like this:

---

```
primarynode MailTo::operator@mailhost.com,root@mailhost.com
```

---

or to specify a subject line for an email alert, use:

---

```
primarynode MailTo::operator@mailhost.com,root@mailhost.com::Mysubject
```

---

## Heartbeat Maintenance

Thanks to the fact that Heartbeat resource scripts are called by the heartbeat daemon with `start`, `stop`, or `status` requests, you can restart a resource without causing a cluster transition event. For example, say your Apache web server daemon is running on the `primary.mydomain.com` web server, and the `backup.mydomain.com` server is not running anything; it is waiting to offer the web server resource in the event of a failure of the primary computer. If you needed to make a change to your `httpd.conf` file (on both servers!) and you wanted to stop and restart the Apache daemon on the primary computer, you would not want this to cause Heartbeat to start offering the service on the backup computer. Fortunately, you can run the `/etc/init.d/httpd restart` command (or `/etc/init.d/httpd stop` followed by the `/etc/init.d/httpd start` command) without causing any change to the cluster status as far as Heartbeat is concerned.

Thus, you can safely stop and restart all of the cluster resources Heartbeat has been asked to manage, with perhaps the exception of filesystems, without causing any change in resource ownership or causing a cluster transition event. Of course, many daemons will also recognize the `SIGHUP` (or `kill -HUP <process-ID-number>`) command as well, so you can force a resource daemon to reload its configuration files after making a change without stopping and restarting it.

Again, in the case of the Apache `httpd` daemon, if you change the `httpd.conf` file and want to notify the running daemons of the change, you would send them the `SIGHUP` signal with the following command:

---

```
#kill -HUP `cat /var/run/httpd.pid`
```

---

**NOTE** *The file containing the `httpd` parent process ID number is controlled by the `PidFile` entry in the `httpd.conf` file (this file is located in the `/etc/httpd/conf` directory on Red Hat Linux systems).*

### Changing Heartbeat Configuration Files

If you need to make a change to the heartbeat configuration file `/etc/ha.d/authkeys`, or `/etc/ha.d/ha.cf`, you can force the running heartbeat daemon to reload these configuration files with the following command.

---

```
#/etc/init.d/heartbeat reload
```

---

or

---

```
#service heartbeat reload
```

---

When you change a `haresources` file, you need to restart Heartbeat on both the primary and the backup server to make your changes take effect (the `reload` option will not work).

### ***Server Maintenance and the Heartbeat `auto_fallback` Option***

Normally, when the primary server crashes and the backup server takes ownership of a resource, the backup server will only hold this resource until the primary server comes back up. Once the primary server is up and running again, the backup server will release the resource and the primary server will assume ownership once again; it will start the resource script and start offering the service to client computers. This is the default `heartbeat fallback` configuration.

To modify this Heartbeat behavior, add the following line before the node entries in your `/etc/ha.d/ha.cf` file (through version 1.1.2 of Heartbeat):

---

```
nice_fallback on
```

---

For Heartbeat versions 1.1.2 and later, the syntax is more intuitively obvious:

---

```
auto_fallback off
```

---

These options tell Heartbeat to leave the resource on the backup server even after the primary server comes back on line. Make this change to the `ha.cf` file on both heartbeat servers and then issue the following command on both servers to tell Heartbeat to re-read its configuration files:

---

```
#/etc/init.d/heartbeat reload
```

---

You should see a message like the following in the `/var/log/messages` file:

---

```
heartbeat[1032]: info: nice_fallback is in effect.
```

---

This configuration is useful when you want to perform system maintenance tasks that require you to reboot the primary server. When you take down the primary server and the resources are moved to the backup server, they will not automatically move back to the primary server.

Once you are happy with your changes and want to move the resource back to the primary server, you would remove the `nice_fallback` option from the `ha.cf` file and again run the following command on the backup server.

---

```
#/etc/init.d/heartbeat reload
```

---

(You do not need to run this command on the primary server because we are about to restart the heartbeat daemon on the primary server anyway.) Now, force the resource back over to the primary server by entering the following command on the primary server:

---

```
#/etc/init.d/heartbeat restart
```

---

**NOTE** *If you want auto\_failback turned off as the default, or normal, behavior of your Heartbeat configuration, be sure to place the auto\_failback option in the ha.cf file on both servers. If you neglect to do so, you may end up with a split-brain condition; both servers will think they should own the cluster resources. The setting you use for the auto\_failback (or the deprecated nice\_failback) option has subtle ramifications and will be discussed in more detail in Chapter 9.*

### **Forcing the Primary Server into Standby Mode**

In a two-node Heartbeat configuration, you can force the primary server to relinquish its resources, without stopping Heartbeat, by forcing the primary server into standby mode. This causes the backup server to start the resource scripts and take ownership of the Heartbeat resources. Run this command as root on the primary server:

---

```
#/usr/lib/heartbeat/hb_standby
```

---

The primary server will not go into standby mode if it cannot talk to the heartbeat daemon on the backup server. In Heartbeat versions through 1.1.2, the hb\_standby command required nice\_failback to be turned on. With the change in syntax from nice\_failback to auto\_failback, Heartbeat no longer requires auto\_failback to be turned off. However, if you are using an older version of Heartbeat that still supports nice\_failback, it must still be turned on to use the hb\_standby command.

In this example, the primary server (where we ran the hb\_standby command) is requesting that the backup server take over the resources. When the backup server receives this request, it asks the primary server to release its resources. If the primary server does not release the resources,<sup>11</sup> the backup server will not start them.

**NOTE** *The hb\_standby command allows you to specify an argument of local, foreign, or all to specify which resources should go into standby (or failover to the backup server).*

---

<sup>11</sup> There is a 20-minute timeout period (it was a 10-second timeout period prior to version 1.0.2).

## **Tuning Heartbeat's Deadtime Value**

Sometimes Heartbeat will report that it cannot hear its own heartbeat, or that heartbeat times are too long. If the heartbeat logs indicate that a heartbeat was not received within the `deadtime` timeout period and the backup server tried to take over for the primary server when you did not want it to, you need to properly tune your `deadtime` value to account for system and network environmental conditions that may be causing heartbeats to get lost or to not be heard. This can occur on systems that are heavily loaded with network processing tasks, or even with heavy CPU utilization.

To tune the heartbeat `deadtime` value for these conditions, set the `deadtime` value to a large value such as 60 seconds or higher, and set the `warntime` value to the number of seconds you would like to use for your `deadtime` value.

Now run the system for a few weeks and carefully watch the `/var/log/messages` file and the logfile `/var/log/ha-log` for `warntime` messages indicating the longest period of time your system went without hearing a heartbeat. Armed with that information, set your `warntime` to this amount, and multiply this `warntime` value by 1.5 to 2 to arrive at the smallest possible value you should use for your `deadtime`. Leave logging enabled and continue to monitor your logs to make sure you have not set the value too low.

## **Informational Messages in Heartbeat's Log**

You may see messages such as the following in your message log file:

---

```
heartbeat: info: RealMalloc stats: 976 total malloc bytes. pid [369/HBREAD
heartbeat: info: MSG stats: 0/441708 age 0 [pid370/MST_STATUS]
heartbeat: info: ha_malloc stats: 0/9035987 0/0
```

---

These messages will appear every 24 hours, beginning when Heartbeat was started. After a few days of operation, the total number of bytes used should not grow. These informational messages from Heartbeat can be ignored when Heartbeat is operating normally.

## **Failover and Respawn (Automatically Restarting Failed Resources)**

On a normal Unix/Linux server, the `init` daemon will start daemons (usually serial line communication services or `tty` related services) based on entries in the `/etc/inittab` file. If the entry contains the word `respawn`, `init` will monitor the daemon and restart it if it dies for any reason.

If you need to run a service that needs to be restarted or respawned automatically when it fails, you have a few options:

- If the application should run on both the primary and the backup server all of the time, create an entry for the application in `/etc/inittab` (see the man page for `inittab` for syntax details).

- If the application should only run when Heartbeat is running, you can create a `respawn` entry for the service in the `/etc/ha.d/ha.cf` file that looks like this:

---

```
respawn root /usr/sbin/faxgetty ttyQ01e0
```

---

This line tells Heartbeat to run the `/usr/sbin/faxgetty` program and pass it the argument `ttyQ01e0`. Heartbeat will do this when it first starts up on both the primary and the backup Heartbeat servers (recall that the Heartbeat configuration files should always be the same on both servers).

- If the application should run on only one of the Heartbeat servers at a time, you will have to implement a service under Heartbeat's control that knows how to restart failed daemons, such as the Daemontools package (<http://cr.yp.to/daemontools.html>), or that can use the `cl_respawn` utility included with the Heartbeat package. To use the `cl_respawn` utility, create a line such as the following in the start section of your resource script:

---

```
cl_respawn /usr/sbin/faxgetty ttyQ01e0
```

---

When Heartbeat calls your resource script and passes it the `start` argument, it will then run the utility `cl_respawn` with the arguments `/usr/sbin/faxgetty` and `ttyQ01e0`. The `cl_respawn` utility is a small program that is unlikely to crash, because it doesn't do much—it just hangs around in the background and watches the daemon it started (`faxgetty` in this example) and restarts the daemon if it dies. (Run `cl_respawn -h` from a shell prompt for more information on the capabilities of this utility.)

## ***License Manager Failover***

A license manager daemon such as `lmgrd` from GlobeTrotter Software can be configured to failover in conjunction with an IP address, just like any other daemon. However, before you can use a license manager, you will need a second set of licenses from your software vendor for the backup server's hostid. Some software vendors will allow you to have two sets of licenses if you agree to use the second set only when the primary license server goes down.

## **In Conclusion**

To make a resource highly available, you need to eliminate single points of failure and understand how to properly administer a highly available server pair. To administer a highly available server pair, you need to know how to failover a resource from the primary server to the backup server manually so that you can do maintenance tasks on the primary server without affecting the resource's availability. Once you know how to failover a resource manually, you can place it under Heartbeat's control by creating the proper entry or entries in the `haresources` file to automate the failover process.

# 9

## STONITH AND IPFAIL



Recall from Chapter 6 that a split-brain condition occurs when more than one server thinks it has exclusive ownership of a resource. The consequences of a split-brain condition can include the inability to answer client computer requests for service, or perhaps even worse, the ability to answer requests inaccurately. For example, if two servers have the ability to offer client computers access to a warehouse inventory database that is stored locally on each server's disk drive in a high-availability configuration, only one server should be able to modify the database at a time. If each server allows users to modify its copy of the inventory database, then neither database will be accurate, and correcting the errors in the data will be difficult, if not impossible.

**NOTE** *The situation is even worse if both servers share access to a single disk drive (using a shared SCSI bus, for example). In the scenario just described, the local copy of the database will be out of date on both servers, but if a split-brain condition occurs when the primary and backup server both try to write to the same database (stored on the shared disk drive), the entire database may become corrupted.*

This chapter explains how to avoid a split-brain condition using a component in the Heartbeat package called Stonith. We'll discuss how to use a feature of Heartbeat called ipfail, which allows Heartbeat servers to detect which server should own the resources based on the Heartbeat servers' ability to communicate on the network. We'll also show you how to use the kernel capability called Watchdog, which allows the system to reboot itself if the Heartbeat program hangs, and we'll conclude with the basic tests you should perform on your high-availability servers before they go into production.

## Stonith

Stonith, or “shoot the other node in the head,”<sup>1</sup> is a component of the Heartbeat package that allows the system to automatically reset the power of a failing server using a remote or “smart” power device connected to a healthy server. A Stonith device is a device that can turn power off and on in response to software commands. A serial or network cable allows a server running Heartbeat to send commands to this device, which controls the electrical power supply to the other server in a high-availability pair of servers. The primary server, in other words, can reset the power to the backup server, and the backup server can reset the power to the primary server.

**NOTE** *Although there is no theoretical limitation on the number of servers that can be connected to a remote or “smart” power device capable of cycling system power, the majority of Stonith implementations use only two servers. Because a two-server Stonith configuration is the simplest and easiest to understand, it is likely in the long run to contribute to—rather than detract from—system reliability and high availability.<sup>2</sup>*

This section will describe how to get started with Stonith using a two-server (primary and backup server) configuration with Heartbeat. When the backup server in this two-server configuration no longer hears the heartbeat of the primary server, it will power cycle the primary server before taking ownership of the resources. There is no need to configure sophisticated cluster quorum election algorithms in this simple two-server configuration;<sup>3</sup> the backup server can be sure it will be the exclusive owner of the resource while the primary server is booting. If the primary server cannot boot and reclaim its resources, the backup server will continue to maintain ownership of them indefinitely. The backup server may also keep control of the resources if you enable the `auto_failback` option in the Heartbeat `ha.cf` configuration file (as discussed in Chapter 8).

---

<sup>1</sup> Other high-availability solutions sometimes call this *Stomith*, or “shoot the other machine in the head.”

<sup>2</sup> “Complexity is the enemy of reliability,” writes Alan Robertson, the lead Heartbeat developer.

<sup>3</sup> With three or more servers competing for cluster resources, a quorum, or majority-wins, election process is possible (quorum election will be included in Release 2 of Heartbeat).

Forcing the primary server to reboot with a power reset is the crudest and surest way to avoid a split-brain condition. As mentioned in Chapter 6, a split-brain condition can have dire consequences when two servers share access to an external storage device such as a single SCSI bus connection to one disk drive. If the server with write permission, the primary server, malfunctions, the backup server must take great precautions to ensure that it will have exclusive access to the storage device before it modifies data.<sup>4</sup>

Stonith also ensures that the primary server is not trying to claim ownership of an IP address after it fails over to a backup server. This is more important than it sounds, because many times a failover can occur when the primary server is simply not behaving properly and the lower-level networking protocols that allow the primary server to respond to ARP requests (“Who owns this IP address?”) are still working. The backup server has no reliable way of knowing that the primary server is engaging in this sort of improper behavior once communication with the daemons on the primary server, especially with the Heartbeat daemon, is lost.

## An Unconventional Approach: Using a Single Stonith Device

In this chapter, I will use an unconventional approach to deploying a high-availability configuration. I will describe how to make resources highly available using only one Stonith device. Normally, high-availability configurations are built using two Stonith devices (one connected to the primary server and one connected to the backup server). In this chapter, however, I will describe how to deploy Heartbeat using only one Stonith device, which introduces two important limitations into your high-availability configuration:

1. All resources must run on the primary server (no resources are allowed on the backup server as long as the primary server is up).
2. A failover event can only occur one time and in one direction. In other words, the resources running on the primary server can only failover to the backup server *once*. When the backup server takes ownership of the resources, the primary server is shut down, and operator intervention is required to restore the primary server to normal operation.

When you use only one Stonith device, you must run all of the highly available resources on the primary server, because the primary server will not be able to reset the power to the backup server (the primary server needs to reset the power to the backup server when the primary server wants to take back ownership of its resources after it has recovered from a crash). Resources running on the backup server, in other words, are not highly available without a second Stonith device. Operator intervention is also required after a

---

<sup>4</sup> More sophisticated methods are possible through advanced SCSI commands, which are not implemented in all SCSI devices and are not currently a part of Heartbeat.

failover event when you use only one Stonith device, because the primary server will go down and stay down—you will no longer have a highly available server pair.

With a clear understanding of these two limitations, we can continue our discussion of Heartbeat and Stonith using a sample configuration that only uses one Stonith device.

### **Sample Heartbeat with Stonith Configuration**

Figure 9-1 shows a two-server, high-availability configuration using Heartbeat and a single Stonith device with three client computers connecting to resources on the primary server.

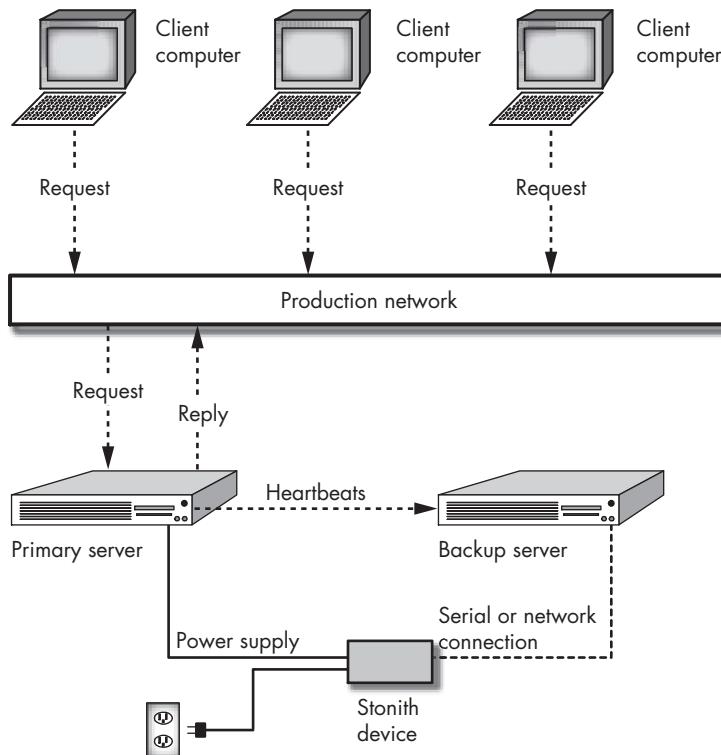


Figure 9-1: Two-server Heartbeat with Stonith—normal operation

Normally, the primary server broadcasts its heartbeats and the backup server hears them and knows that all is well. But when the backup server no longer hears heartbeats coming from the primary server, it sends the proper software commands to the Stonith device to cycle the power on the primary server, as shown in Figure 9-2.

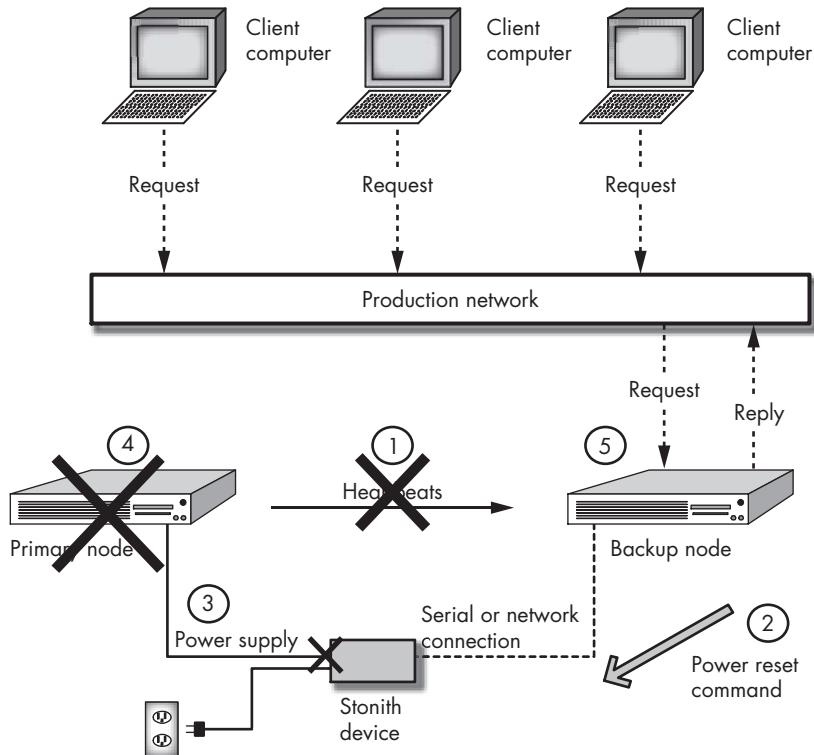


Figure 9-2: Stonith sequence of events

### **Stonith Sequence of Events**

As shown in Figure 9-2, the Stonith operation proceeds as follows:

1. The Stonith event begins when heartbeats are no longer heard on the backup server.

**NOTE** *This does not necessarily mean that the primary server is not sending them. The heartbeats may fail to reach the backup server for a variety of reasons. This is why at least two physical paths for heartbeats are recommended in order to avoid false positives.*

2. The backup server issues a Stonith reset command to the Stonith device.
3. The Stonith device turns off the power supply to the primary server and then turns it back on.
4. As soon as the power is cut off from the primary server, it is no longer able to access cluster resources and is no longer able to offer resources to client computers over the network. This *guarantees* that client computers are unable to access the resources on the primary server and eliminates the possibility of a split-brain condition.

- The backup server then acquires the primary server's resources. Heartbeat runs the proper resource script(s) with the start argument and performs gratuitous ARP broadcasts so that client computers will begin sending their requests to its network interface card.

Once the primary server finishes rebooting it will attempt to reclaim ownership of its resources again by asking the backup server to relinquish them, unless both servers have the `auto_failback` turned off.<sup>5</sup>

**NOTE** *Client computers should always access cluster resources on an IP address that is under Heartbeat's control. This should be an IP alias and not an IP address that is automatically added to the system at boot time.*

## Stonith Devices

Here is a partial listing of the Stonith devices supported by Heartbeat:

Stonith Name	Device	Company Website
apcmaster	APC Master Switch AP9211	<a href="http://www.apcc.com">www.apcc.com</a>
apcmastersnmp	APC Masterswitch (SNMP)	<a href="http://www.apcc.com">www.apcc.com</a>
apcsmart	APC Smart-UPS <sup>a</sup> Tested with 900XLI	<a href="http://www.apcc.com">www.apcc.com</a>
baytech	Tested with RPC-5 <sup>b</sup>	<a href="http://www.baytechdcd.com">www.baytechdcd.com</a>
nw_rpc100s	Micro Energetics Night/Ware RPC100S	<a href="http://microenergeticscorp.com">microenergeticscorp.com</a>
rps10	Western Telematics RPS-10M	<a href="http://www.wti.com">www.wti.com</a>
wti_nps	Western Telematics Network Power Switches NPS-xxx Western Telematics Telnet Power Switches (TPS-xxx)	

<sup>a</sup> Comments in the code state that no configuration file is used with this device. Apparently it will always use the `/dev/ups` device name to send commands, so you need to create a link to `/dev/ups` from the proper `/dev/tty` device.

<sup>b</sup> The code implies other RPC devices should work as well.

Stonith supports two additional methods of resetting a system:

<b>Meatware</b>	Operator alert; this tells a human being to do the power reset. It is used in conjunction with the <code>meatclient</code> program.
<b>SSH</b>	Usually used only for testing. This causes Stonith to attempt to use the secure shell <code>SSH<sup>a</sup></code> to connect to the failing server and perform a reboot command.

<sup>a</sup> See Chapter 5 for a description of how to set up and configure SSH between cluster, or Heartbeat, servers.

<sup>5</sup> Formerly (prior to version 1.1.2) the `nice_failback` option turned on.

You may also still see a `null` device. This “device” was originally used by the Heartbeat developers for coding purposes before the `SSH` device was developed.

## ***Viewing the Current List of Supported Stonith Devices***

If you download the source RPM file, you can view the list of currently supported Stonith devices in the file:

---

```
/usr/src/redhat/SOURCES/heartbeat-<version>/STONITH/README
```

---

Or you can list the Stonith device names with the command:

---

```
#/usr/sbin/stonith -L
```

---

## ***The Stonith Meatware “Device”***

Before purchasing one of the supported Stonith devices, you can use a meatware “device” to experiment with Stonith. A *meatware device* is a whimsical reference to a human being. When you use a meatware device, Heartbeat simply raises an operator alert (instead of resetting the power using software commands to a hardware device connected to a serial or network cable). After following the recipe in Chapter 7 that told you how to download and install the Heartbeat and Stonith RPMs, enter the following command to create a meatware device for a nonexistent host we will call `chilly`:

---

```
#/usr/sbin/STONITH -t meatware -p "" chilly
```

---

This command tells Stonith to create a device of type (`-t`) `meatware` with no parameters (`-p ""`) for host `chilly`.

The Stonith program normally runs as a daemon in the background, but we’re running it in the foreground for testing purposes. Log on to the same machine again (if you are at the console you can press `CTRL+ALT+F2`), and take a look at the messages log with the command:

---

```
#tail /var/log/messages
STONITH: OPERATOR INTERVENTION REQUIRED to reset test.
STONITH: Run "meatclient -c test" AFTER power-cycling the machine.
```

---

Stonith has created a special file in the `/tmp` directory that you can examine with the command:

---

```
#file /tmp/.meatware.chilly
/tmp/.meatware.chilly: fifo (named pipe)
```

---

Now, because we don't really have a machine named `chilly` we will just pretend that we have obeyed the instructions of the Heartbeat program when it told us to reset the power to host `chilly`. If this were a real server, we would have walked over and flipped the power switch off and back on before entering the following command to clear this Stonith event:

---

```
#meatclient -c chilly
```

---

The meatclient program should respond:

---

**WARNING!**

If server "chilly" has not been manually power-cycled or disconnected from all shared resources and networks, data on shared disks may become corrupted and migrated services might not work as expected.

Please verify that the name or address above corresponds to the server you just rebooted.

PROCEED? [yn]

---

After entering `y` you should see:

---

```
Meatware_client: reset confirmed.
```

---

Stonith should also report in the `/var/log/messages` file:

---

```
STONITH: server chilly Meatware-reset.
```

---

### ***Using the Stonith Meatware Device with Heartbeat***

Now that we have used the Stonith software commands and the meatware device manually, let's configure Heartbeat to do the same thing for us automatically at the time of a failover. On both of your Heartbeat servers add the following entry to the `/etc/ha.d/ha.cf` file:

---

```
stonith_host * meatware
```

---

Normally the first parameter after the word `stonith_host` is the name of the Heartbeat server that has a physical connection to the Stonith device. If both (or all) Heartbeat servers can connect to the same Stonith device, you can use the wildcard character (\*) to indicate that any Heartbeat server can perform a power reset using this device. (Normally this would be used with a smart or remote power device that is connected via an Ethernet network capable which allows both Heartbeat servers to connect to it.)

Because `meatware` is an operator alert message sent to the `/var/log/messages` file and not a real device we do not need to worry about any additional cabling and can safely assume that both the primary and backup servers will have "access" to this Stonith "device." They need only be able to send messages to their log files.

**NOTE** When following this recipe, be sure that the `auto_failback` option is turned on in your `ha.cf` file. With Heartbeat version 1.1.2, the `auto_failback` option can be set to either on or off and `ipfail` will work. Prior to version 1.1.2 (when `nice_failback` changed to `auto_failback`), the `nice_failback` option had to be set to on for `ipfail` to work.

1. Use a simple `haresources` entry like this:

---

```
#vi /etc/ha.d/haresources
primary.mydomain.com sendmail
```

---

The second line says that the primary server should normally own the `sendmail` resource (it should run the `sendmail` daemon).

2. Now start Heartbeat on both the primary and backup servers with the commands:

---

```
primaryserver> service heartbeat start
backupserver> service heartbeat start
```

---

or

---

```
primaryserver> /etc/init.d/heartbeat start
backupserver> /etc/init.d/heartbeat start
```

---

**NOTE** The examples above and below use the name of the server followed by the `>` character to indicate a shell prompt on either the primary or the backup Heartbeat server.

3. Now kill the Heartbeat daemons on the primary server with the following command:

---

```
primaryserver> killall -9 heartbeat
```

---

**NOTE** Stopping Heartbeat on the primary server using the Heartbeat init script (`service heartbeat stop`) will cause Heartbeat to release its resources. Thus the backup server will not need to reset the power of the primary server. To test a Stonith device, you'll need to kill Heartbeat on the primary server without allowing it to release its resources. You can also test the operation of your Stonith configuration by disconnecting all physical paths for heartbeats between the servers.

4. Watch the `/var/log/messages` file on the backup server with the command:

---

```
backupserver> tail -f /var/log/messages
```

---

You should see Heartbeat issue the Meatware Stonith warning in the log and then wait before taking over the resource (`sendmail` in this case):

---

```
backupserver heartbeat[835]: info: ****
backupserver heartbeat[835]: info: Configuration validated. Starting
heartbeat <version>
backupserver heartbeat[836]: info: heartbeat: version <version>
```

---

```
backupserver heartbeat[836]: info: Heartbeat generation: 3
backupserver heartbeat[836]: info: UDP Broadcast heartbeat started on port
694 (694) interface eth0
backupserver heartbeat[841]: info: Status update for server backupserver:
status up
backupserver heartbeat: info: Running /etc/ha.d/rc.d/status status
backupserver heartbeat[841]: info: Link backupserver:eth0 up.
backupserver heartbeat[841]: WARN: server primaryserver: is dead
backupserver heartbeat[841]: info: Status update for server backupserver:
status active
backupserver heartbeat[847]: info: Resetting server primaryserver with
[Meatware Stonith device]
backupserver heartbeat[847]: OPERATOR INTERVENTION REQUIRED to reset
primaryserver.
backupserver heartbeat[847]: Run "meatclient -c primaryserver" AFTER
power-cycling the machine.
backupserver heartbeat: info: Running /usr/local/etc/ha.d/rc.d/status
status
backupserver heartbeat[852]: info: No local resources [/usr/local/lib/
heartbeat/ResourceManager listkeys backupserver]
backupserver heartbeat[852]: info: Resource acquisition completed.
```

---

Notice that Heartbeat did not start the sendmail resource; it is waiting for you to clear the Stonith Meatware event.

5. Clear that event by entering the command:

---

```
backupserver> meatclient -c primaryserver
```

---

The /var/log/messages file should now indicate that Heartbeat has started the sendmail resource on the backup server:

---

```
backupserver heartbeat[847]: server primaryserver Meatware-reset.
backupserver heartbeat[847]: info: server primaryserver now reset.
backupserver heartbeat[841]: info: Resources being acquired from
primaryserver.
backupserver heartbeat: info: Running /usr/local/etc/ha.d/rc.d/stonith
STONITH
backupserver heartbeat: info: Running /usr/local/etc/ha.d/rc.d/status
status
backupserver heartbeat: stonith complete
backupserver heartbeat: info: Taking over resource group sendmail
backupserver heartbeat: info: Acquiring resource group: primaryserver
sendmail
backupserver heartbeat: info: Running /etc/init.d/sendmail start
```

---

Heartbeat on the backup server is now satisfied: it owns the primary server's resources and will listen for Heartbeats to find out if the primary server is revived.

To complete this simulation, you can really reset the power on the primary server, or simply restart the Heartbeat daemon and watch the `/var/log/messages` file on both systems. The primary server should request that the backup server give up its resources and then start them up again on the primary server where they belong. (The `sendmail` daemon should start running on the primary server and stop running on the backup server in this example.)

**NOTE** *Notice in this example how you will have a much easier time performing system maintenance when you place all of your resources on a primary server and leave the backup server idle. However, you may want to place active services on both the primary and backup Heartbeat server and use the `hb_standby` command to failover resources when you need to do maintenance on one or the other server.*

### **Using a “Real” Stonith Device**

Each Stonith device has unique configuration options. To view the configuration options for any Stonith device, enter the command:

---

```
#stonith -help
```

---

You can also obtain syntax configuration information for a particular Stonith device using the `stonith` command from a shell prompt as in the following example:

---

```
# /usr/sbin/stonith -l -t rps10 test
```

---

This causes Stonith to report:

---

```
STONITH: Cannot open /etc/ha.d/rpc.cfg
STONITH: Invalid config file for rps10 device.
STONITH: Config file syntax: <serial_device> <server> <outlet> [ <server>
<outlet> [...] ]
All tokens are white-space delimited.
Blank lines and lines beginning with # are ignored
```

---

This output provides you with the syntax documentation for the rps10 Stonith device. The syntax for the parameters you can pass to this Stonith device (also called the configuration file syntax) include a /dev serial device name (usually `/dev/ttyS0` or `/dev/ttyS1`); the server name that is being supplied power by this device; and the physical socket outlet number on the rps10 device. Thus, the following syntax can be used in the `/etc/ha.d/ha.cf` file on both the primary and the backup server in the `/etc/ha.d/ha.cf` configuration file:

---

```
STONITH_host backupserver rps10 /dev/ttyS0 primaryserver.mydomain.com 0
```

---

This line tells Heartbeat that the backup server controls the power supply to the primary server using the serial cable connected to the /dev/ttyS0 port on the backup server, and the primary server's power cable is plugged in to outlet 0 on the rps10 device.

**NOTE** *For Stonith devices that connect to the network you will also need to specify a login and password to gain access to the Stonith device over the network before Heartbeat can issue the power reset command(s). If you use this type of Stonith device, however, be sure to carefully consider the security risk of potentially allowing a remote attacker to reset the power to your server, and be sure to secure the ha.cf file so that only the root account can access it.*

### **Avoiding Multiple Stonith Events**

One potential advantage of offering all of the resources from the primary server and not running any resources on the backup server (which you will recall is a limitation of the single-Stonith configuration I have been describing in this chapter) is that you can simply power off the primary server when the backup server takes ownership of the resources. This prevents the situation where confused Heartbeat servers start resetting each other's power and resources move back and forth between the primary and backup server.

To cause Heartbeat to power down, rather than power reset the primary server during a Stonith event, you can do one of two things:

1. Look at the driver code in the Heartbeat package for your Stonith device and see if it supports a poweroff (rather than a power reset)—most do. If it does, you can simply change the source code in heartbeat that does the stonith command and change it from this:

---

```
rc = s->s_ops->reset_req(s, ST_GENERIC_RESET, nodename);
```

---

to this:

---

```
rc = s->s_ops->reset_req(s, ST_POWEROFF, nodename);
```

---

This allows you to power off rather than power reset any Stonith device that supports it.

2. If you don't want to compile Heartbeat source code, you may be able to simply configure the system BIOS on your primary server so that a power reset event will not cause the system to reboot. (Many systems support this, check your system documentation to see if yours is one of them.)

## Network Failures

Even with these preparations, we still have not eliminated all single points of failure in our two-server Heartbeat failover design. For example, what happens if the primary server simply loses the ability to communicate properly with the client computers over the normal, or production network?

In such a case, if you have a properly configured Heartbeat configuration the heartbeats will continue to travel to the backup server. This is thanks to the redundancy you built into your Heartbeat paths (as described in Chapter 8), and no failover will occur. Still, the client computers will no longer be able to access the resource daemons on the primary server (the cluster resources).

We can solve this problem in at least two ways:

- Run an external monitoring package, like the Perl program Mon, on the primary server and watch for the failure of the public NIC. When Mon detects the failure of this NIC it should shut down the Heartbeat daemon (or force it into standby mode) on the primary server. The backup server will then takeover the resources and, assuming it is healthy and can communicate on its public network interface, the client computers will once again have access to the resources. (See Chapter 17 for more information about Mon.)
- Use the ipfail API plug-in, which allows you to specify one or more ping servers in the Heartbeat configuration file. If the primary server suddenly fails to see one of the ping servers, it asks the backup server, “Did you see that ping server go down too?” If the backup server can still talk to the ping server, it knows that the primary server is not communicating on the network properly and it should now take ownership of the resources.

### **ipfail**

Beginning with Heartbeat version 0.4.9d, the ipfail plug-in is included with the Heartbeat RPM package as part of the standard Heartbeat distribution. To use ipfail, decide which network device (IP address) both Heartbeat servers should be able to ping at all times (such as a shared router, a network switch that is never supposed to go offline, and so on). Next, enter this IP address in your `/etc/ha.d/ha.cf` file and tell Heartbeat to start the ipfail plug-in each time it starts up:

---

```
#vi /etc/ha.d/ha.cf
```

---

Add three lines before the final server lines at the end of the file like so:

---

```
respawn hacluster /usr/lib/heartbeat/ipfail
ping 10.1.1.254 10.1.1.253
auto_fallback off
```

---

The first line above tells Heartbeat to start the ipfail program on both the primary and backup server,<sup>6</sup> and to restart or respawn it if it stops, using the hacluster user created during installation of the Heartbeat RPM package. The second line specifies one or more ping servers or network nodes that Heartbeat should ping at heartbeat intervals to be sure that its network connections are working properly. (If you are building a firewall machine, for example, you will probably want to use ping servers on both interfaces, or networks.<sup>7</sup>)

**NOTE** *If you are using a version of Heartbeat prior to version 1.1.2, you must turn nice\_fallback on. Version 1.1.2 and later allow auto\_fallback (the replacement for nice\_fallback but with the opposite meaning) to be either on or off.*

Now start Heartbeat on both servers and test your configuration. You should see a message in `/var/log/messages` indicating that Heartbeat started the ipfail child client process. Try removing the network cable on the primary server to break the primary server's ability to ping one of the ping servers, and watch as ipfail forces the primary server into standby mode. The backup server should then take over the resources listed in `haresources`.

## Watchdog and Softdog

The kernel has its own method to handle a hung system, called watchdog. Watchdog is simply a kernel module that checks a timer to make sure the system is healthy. If watchdog thinks the kernel is hung, it can take drastic action such as a system reboot. If you want to protect your high-availability server configuration from a server hang that causes an interruption in services *even when the server hang is not detected by Heartbeat*, you should enable watchdog in your kernel.

**NOTE** *We are talking about a server hang here and not an application problem. Heartbeat (prior to Heartbeat release 2, which is not yet available as of this writing) does not monitor resources or the applications under its control to see if they are healthy—to do this you need to use another package such as the Mon monitoring system discussed in Part IV.*

---

<sup>6</sup> The `/etc/ha.d/ha.cf` configuration file should be the same on the primary and backup server.

<sup>7</sup> And, of course, be sure to configure your iptables or ipchains rules to accept ICMP traffic (see Chapter 2).

A watchdog device is normally connected to a system to allow the kernel to determine whether the system has hung (when the kernel no longer sees the external timer device updating properly, it knows that something has gone wrong).

The watchdog code also supports a software replacement for external hardware timers called softdog. Softdog maintains an internal timer that is updated as soon as another process on the system writes to the `/dev/watchdog` device file. If softdog doesn't see a process write to the `/dev/watchdog` file, it assumes that the kernel must be malfunctioning, and it will initiate a kernel panic. Normally a kernel panic will cause a system to shut down, but you can modify this default behavior and, instead, cause the system to reboot.

### ***Enable Watchdog in the Kernel***

To enable watchdog in the kernel, you first need to make sure that the softdog module is compiled for your kernel.

**NOTE** *On a normal Red Hat or SuSe distribution you will not need to add watchdog to your kernel because the modular version of the Red Hat kernel contains a compiled copy of the softdog module already.*

If you have compiled your own kernel from source code, run `makemenu config` from the `/usr/src/linux` directory, and check or enable the "Software Watchdog" option on the following submenu:

```
Character Devices
  Watchdog Cards -->
    [*] Watchdog Timer Support
    [M] Software Watchdog (NEW)
```

If this option was not already selected in the kernel, follow the steps in Chapter 3 to recompile and install your new kernel. If you are using the standard modular version of the kernel included with Red Hat (or if you have just finished compiling your own kernel with modular support for the Software Watchdog), enter the following commands to make sure that the module loads into your running kernel:

---

```
#insmod softdog
#lsmod
```

---

You should see softdog listed. Normally the Heartbeat init script will insert this module for you if you have enabled watchdog support in `/etc/ha.d/ha.cf` as described later in this section. Assuming that watchdog is enabled, you should now remove it from the kernel and allow Heartbeat to add it for you when it starts. Remove softdog from the kernel with the command:

---

```
#modprobe -r softdog
```

---

## **Kernel Panic—Hang or Reboot?**

To force the system to reboot instead of halt if the kernel panics, modify the boot arguments passed to the kernel. To do this on a system using the Lilo<sup>8</sup> boot loader, for example, edit `/etc/lilo.conf` near the top of the file, and before `image=` lines so it will take effect for all versions of the kernel you have configured, and add the following line:

---

```
append="panic=60"
```

---

Then be sure to run the command:

---

```
#lilo -v
```

---

Alternatively, you could also use the command:

---

```
#echo 60 > /proc/sys/kernel/panic
```

---

But you would need to add this command to an init script so that it would be executed each time the system boots.

## **Configure Heartbeat to Support Watchdog**

In addition to using the softdog timer as we've just described (as part of the normal configuration of your server to improve its reliability when the system hangs) you can tell Heartbeat to update the softdog timer. This lets watchdog know that Heartbeat is running and healthy. If the timer doesn't get updated, watchdog will notice and force a kernel panic. In effect, we are telling watchdog to watch Heartbeat.

**NOTE** *With Heartbeat release 1.2.3, you can have apphbd watch Heartbeat and then let watchdog watch apphbd instead.*

When you enable the watchdog option in your `/etc/ha.d/ha.cf` file, Heartbeat will write to the `/dev/watchdog` file (or device) at an interval equal to the `deadtime` timer raised to the next second. Thus, should anything cause Heartbeat to fail to update the watchdog device, watchdog will initiate a kernel panic once the watchdog timeout period has expired (which is one minute, by default).

---

```
#vi /etc/ha.d/ha.cf
```

---

Uncomment the line:

---

```
watchdog /dev/watchdog
```

---

---

<sup>8</sup> See Chapter 3 for a discussion of the Lilo boot loader.

Now restart Heartbeat to give the Heartbeat init script the chance to properly configure the watchdog device, with the command:

---

```
#service heartbeat restart
```

---

You should see softdog listed when you run:

---

```
#lsmod
```

---

**NOTE** *You should do this on all of your Heartbeat servers to maintain a consistent Heartbeat configuration.*

To test the watchdog behavior, kill all of the running Heartbeat daemons on the primary server with the following command:

---

```
#killall -9 heartbeat
```

---

You will see the following warning on the system console and in the `/var/log/messages` file:

---

```
Softdog: WDT device closed unexpectedly. WDT will not stop!
```

---

This error warns you that the kernel will panic. Your system should reboot instead of halting if you have modified the `/proc/sys/kernel/panic` value as described previously.

## Testing Your Heartbeat Configuration

Before you put your Heartbeat high-availability server pair into production, here are a few things to try:

### Unplug the power cord on the primary server

Heartbeat on the backup server should detect the loss of heartbeat packets from the primary server and initiate a failover. Using Stonith, the backup server should turn off or reset the power to the primary server. Heartbeat on the backup server should then run the proper resource scripts (when the Stonith event has “cleared” or completed) to take ownership of the resources. Heartbeat on the backup server should also send gratuitous ARP broadcasts to notify clients and/or network equipment that the MAC addresses for the resource IP addresses have changed.

Client computers, and network equipment such as routers and switches should update their ARP caches to reflect the new MAC address of the backup server. Check the ARP cache on your Cisco IOS network equipment, for example, with the command:

---

```
show ip arp
```

---

Or use the command:

---

```
show ip arp 209.100.100.3
```

---

where 209.100.100.3 is the IP alias that fails over to the backup server. The MAC address should change automatically when the backup server sends out gratuitous ARP broadcasts.<sup>9</sup>

Check all of the client computers that share the same network broadcast address with this command, which works on Windows PCs and Linux hosts:

---

```
arp -a
```

---

#### **Test the behavior of the hb\_standby command**

Use the hb\_standby command on the primary server to force resources to failover to the backup server. Then use the command again on the backup server to force the resources back to the primary server. ipfail will not work properly if the hb\_standby command is not working properly.

#### **Unplug the production network cable on the primary server**

Using ipfail (or Mon,<sup>10</sup> or a similar monitoring tool), the network connection failure should be detected and the resources and IP aliases should failover to the backup server.

#### **Remove one of the heartbeat paths between the two servers**

Use more than one heartbeat path between the servers to avoid false positives (the backup server incorrectly assumes the primary server has died). When you remove only one of these paths, such as the crossover network cable or serial cable connecting them, nothing should happen.

#### **Remove all of the heartbeat paths between the two servers**

What happens when you remove all heartbeat paths between the two servers? If you are using Stonith, the backup server should assume that the primary server has died, initiate a Stonith event, and take over the resources. What happens next depends upon how you have Stonith configured and whether or not you are using the auto\_failback option.

With two Stonith devices (each server controlling the other server's power supply) and the auto\_failback option turned on, the two servers may start repeatedly cycling each other's power or Stonith-ing each other. To avoid this, you can disable auto\_failback or use the method described earlier in this chapter to power off rather than power cycle the primary server.

---

<sup>9</sup>You may need to make sure the Cisco equipment accepts Gratuitous ARP broadcasts with the Cisco IOS command `ip gratuitous-arp`.

<sup>10</sup>See Chapter 17.

### **Kill the heartbeat daemon on the primary server (`killall -9 heartbeat`)**

Stonith is especially important when you are using IP aliases to offer resources to client computers. The backup server must Stonith or power off/reset the primary server before trying to assume ownership of the resources to avoid a split-brain condition.

### **Kill the resource daemon(s) on the primary server**

This case was not addressed by the Heartbeat configuration used in this chapter. Depending on your needs, you can run `cl_status`, included with the Heartbeat package, or `cl_respawn`, which is also included with the Heartbeat package, to monitor or automatically restart services when they fail. You can also use the `Mon` application (described in detail in Chapter 17) to monitor daemons and then take specific action (such as sending an alert to your cell phone or pager) when the service daemons fail.

### **Power reset (or reboot) both servers**

Do the servers boot properly and leave the resources on the primary server where they belong once both systems have finished booting? You may need to adjust the `initdead` time in the `/etc/ha.d/ha.cf` file if the backup server tries to grab the resources away from the primary server before it finishes booting.

## **In Conclusion**

Preventing the possibility of a split-brain condition while eliminating single points of failure in a high-availability server configuration is tricky. The goal of your high-availability server configuration should be to accomplish this with the least amount of complexity required. When you finish building your high-availability server pair you should test it thoroughly and not feel satisfied that you are ready to put it into production until you know how it will behave when something goes wrong (or to use the terminology sometimes used in the high-availability field: test your configuration for its ability to properly handle a fault, and make sure you have good fault isolation).

This ends Part II. I have spent the last several chapters describing how to offer client computers access to a service or daemon on a single IP address without relying on a single server. This capability—the ability to move a resource from one server to another—is an important building block of the high-availability cluster configuration that will be the subject of Part III.



# PART III

## CLUSTER THEORY AND PRACTICE





# 10

## **HOW TO BUILD A LINUX ENTERPRISE CLUSTER**



So far this book has focused on how to build a highly available pair of servers that can support mission-critical applications with no single point of failure. In this part of the book, we will look at how to build a *cluster* of servers that are all capable of running the same services to support the end users.

A cluster is more than a highly available pair of servers, because all of the nodes in the cluster can share the processing load. When one node goes down, all of the users that were logged on to that node are affected, and they can simply log back on again to reconnect to a working cluster node.

## Steps for Building a Linux Enterprise Cluster

To build a Linux Enterprise Cluster, you need to do several things, each of which are outlined in this chapter:

- Decide which NAS server you will use.
- Understand the basic concepts of the Kernel Netfilter and kernel packet routing.
- Learn how to clone a Linux machine.
- Decide on a naming scheme for your cluster.
- Learn how to apply system configuration changes to all cluster nodes.
- Build a Linux Virtual Server Network Address Translation (LVS-NAT) cluster that uses a separate physical network for the cluster.
- Build an LVS Direct Routing (LVS-DR) cluster.
- Install software to automatically remove failed cluster nodes.
- Install software to monitor the cluster.
- Learn how to monitor the performance of the cluster nodes.
- Learn how to update software packages on your cluster nodes and servers using an automated tool.
- Decide which method you will use to centralize user account administration.
- Install a printing system that supports the cluster.
- Install a highly available batch job-scheduling system.
- Purchase the cluster nodes.

### **NAS Server**

If your cluster will run legacy mission-critical applications that rely on normal Unix lock arbitration methods (discussed in detail in Chapter 16) the Network Attached Storage (NAS) server will be the most important performance bottleneck in your cluster, because all filesystem I/O operations will pass through the NAS server.

You can build your own highly available NAS server on inexpensive hardware using the techniques described in this book,<sup>1</sup> but an enterprise-class cluster should be built on top of a NAS device that can commit write operations to a nonvolatile RAM (NVRAM) cache while guaranteeing the integrity of the cache *even if* the NAS system crashes or the power fails before the write operation has been committed to disk.

---

<sup>1</sup> Techniques for building high-availability servers are described in this book, but building an NFS server is outside the scope of this book; however, in Chapter 4 there is a brief description of synchronizing data on a backup NAS server. (Another method of synchronizing data using open source software is the drbd project. See <http://www.drbd.org>.)

**NOTE** For testing purposes, you can use a Linux server as an NFS server. (See Chapter 16 for a discussion of asynchronous Network File System (NFS) operations and why you can normally only use asynchronous NFS in a test environment and not in production.)

## ***Kernel Netfilter and Kernel Packet Routing***

Before you can build a cluster load balancer, you'll need to understand how you can alter the fate of the network packets as they pass through the Linux kernel. The ability to alter the fate of the packet as it passes through the kernel allows you to build a cluster load balancer that distributes incoming requests for services across all cluster nodes. The command-line tools used to alter the fate of a packet are iptables, route, and the ip utility. You should be familiar with these tools before using them to build an enterprise-class cluster on top of them. (See Chapter 2 for more information on these tools.)

## ***Cloning a Linux Machine***

Chapters 4 and 5 describe a method for cloning a Linux machine using the SystemImager package. It would not be practical to build a cluster of nodes that are all configured the same way without using system-cloning software.

## ***Cluster Naming Scheme***

To automate the cloning process, each node should begin with a common string of characters followed by a sequential cluster-node number. For example, the first cluster-node host name could be clnode1, the second could be clnode2, and so forth.

## ***Applying System Configuration Changes to All Nodes***

A cluster administrator needs to know how to automatically apply system administration changes to all cluster nodes. This can be done using the SystemImager package's updateclient command. Before the cluster goes into production, practice using the updateclient command to apply changes made on the Golden Client to all of the cluster nodes (see Chapter 5 for more information).

## ***Building an LVS-NAT Cluster***

Building a Linux Virtual Server Network Address Translation (LVS-NAT) cluster will help you understand how the Linux Virtual Server (LVS) software works. It will also help you ensure that your load balancer is in charge of allocating nodes inside the cluster for inbound connection requests. (See Chapter 11 for an introduction to load balancing and Chapter 12 for an introduction to LVS-NAT.)

## **Building an LVS-DR Cluster**

Once you know how to build an LVS-NAT cluster, you are ready to convert it to a Linux Virtual Server Direct Routing (LVS-DR) cluster, as described in Chapter 13. An enterprise-class cluster that is based on LVS-DR is superior to an LVS-NAT cluster for mission-critical applications for several reasons:

- The LVS-DR cluster is easier to administer. LVS-DR cluster nodes can be administered from outside the cluster network using telnet and ssh to make a connection to the cluster nodes. In an LVS-NAT cluster, the physical network cabling or VLAN configuration prevents you from making direct connections to the cluster nodes.
- The LVS-DR cluster can send replies from the cluster nodes back to the client computers without passing the packets through an intermediate machine (the load balancer).
- The LVS-DR cluster load balancer can malfunction and not render all of the cluster nodes useless. In contrast, if the primary and backup LVS-NAT cluster load balancers both crash at the same time, the entire LVS-NAT cluster is down. In an LVS-DR cluster, if the primary and backup LVS load balancers crash at the same time, the cluster nodes can still be used as separate or distributed servers. (See Chapter 13 for the details of how to build an LVS-DR cluster.) In practice, however, this is a selling point to management and not a “feature” of an LVS-DR cluster.

**NOTE** *The Linux Enterprise Cluster should be protected from an outside attack by a firewall. If you do not protect your cluster nodes from attack with a firewall, or if your cluster must be connected to the Internet, shell access to the cluster nodes should be physically restricted. You can physically restrict shell access to the cluster nodes by building a separate network (or VLAN) that connects the cluster nodes to an administrative machine. This separate physical network is sometimes called an administrative network.<sup>2</sup>*

## **Installing Software to Remove Failed Cluster Nodes**

In this book we will use the ldirectord software package (included on the CD-ROM) to automatically remove nodes from the cluster when they fail. Chapter 15 will describe how to install and configure ldirectord.

As cluster administrator, you will also want to know how to manually remove a node from the cluster for maintenance purposes without affecting users currently logged on to the system. We’ll look at how to do this in Chapter 19.

---

<sup>2</sup> See the discussion of administrative networks in Evan Marcus and Hal Stern’s *Blueprints for High Availability* (John Wiley and Sons, 2003).

## ***Installing Software to Monitor the Cluster Nodes***

You cannot wade through the log files on every cluster node every day. You need monitoring software capable of sending email messages, electronic pages, or text messages to the administrator when something goes wrong on one of the cluster nodes.

Many open source packages can accomplish this task, and Chapter 17 describes a method of doing this using the Simple Network Management Protocol (SNMP) and the Mon software package. SNMP and the Mon monitoring package together allow you to monitor the cluster nodes and send an alert when a threshold that you have specified is violated.

## ***Monitoring the Performance of Cluster Nodes***

In addition to monitoring the cluster nodes for problems, you will also want to be able to monitor the processing load on each node to see if they are balancing the workload properly. The Ganglia package is an excellent tool for doing this. Chapter 18 will describe how to use Ganglia and will discuss a few of the performance metrics (such as the system load average) that Ganglia collects from all of the cluster nodes.

Managers and operations staff can use web pages created with the Ganglia software package to watch the processing load on the cluster in real time. The day your cluster goes into production, this will be one of the most important tools you have to see what is going on inside the cluster.

## ***Updating Software on Cluster Nodes and Servers***

You will also need a method of automatically downloading and installing packages to fix security holes as they are found and plugged. The automated tool Yum, described in Appendix D, is one way of doing this. You'll want to learn how to use Yum or another of the many automated package update utilities before going into production to make sure you have built a system that can continue to evolve and adapt to change.

You'll also need to take into account subtle problems, like the fact that the SystemImager `updateclient` command may overwrite the package registration information or the list of software (RPM) packages stored on the cluster node's disk drive. (To resolve this problem, you may want to install software on one node—the SystemImager Golden Client—and then just use the `updateclient` command to upgrade the remaining cluster nodes.)

## ***Centralizing User Account Administration***

You will need to somehow centralize the administration of user accounts on your cluster. See Chapters 1 and 19 for discussions of the possible account-administration methods, such as NIS, LDAP, Webmin, OPIUM (part of the OSCAR package), or a cron job that copies information stored on a central

server to the local `/etc/passwd` file on each cluster node. (Whichever method you select, you will also want to decide if this is the right method for centralizing group and host information as well.)

### ***Installing a Printing System***

You will need to set up a printing system that allows you to have a single point of control for all print jobs from the cluster without creating a single point of failure. The use of the LPRng package for this purpose in a cluster environment will be briefly discussed in Chapter 19.

### ***Installing a Highly Available Batch Job-Scheduling System***

Building a highly available cluster will not improve the reliability of your system if your batch job scheduling system is not highly available. We'll look at how to build a batch job scheduling system with no single point of failure in Chapter 18.

### ***Purchasing the Cluster Nodes***

Clusters built to support scientific research can sometimes contain thousands of nodes and fill the data centers of large research institutions (see <http://clusters.top500.org>). For the applications that run on these clusters, there may be no practical limit to the amount of CPU cycles they can use—as more cycles become available, more work gets done. By contrast, an enterprise cluster has a practical upper limit on the amount of processing cycles that an application can use. An enterprise workload will have periods of peak demand, where processing needs may be double, triple, or more, compared to processing needs during periods of low system activity. However, at some point more processing power does not translate into more work getting done, because external factors (such as the number and ability of the people using the cluster) will determine this limitation.

A Linux Enterprise Cluster will therefore have an optimal number of nodes, determined by the requirements of the organization using it, and by the costs of building, maintaining, and supporting it. In this section, we'll look at two basic design considerations for finding the optimal number of nodes: cluster performance and the impact of a single node failure.

### ***Capacity Planning and Cluster Performance***

Applications spend most of their idle time in one of four states: waiting for user input, waiting for the CPU, waiting for filesystem I/O, or waiting for the network I/O. When you build a cluster, you greatly reduce only one of these—the amount of time applications spend waiting for the CPU. By spreading users out over several cluster nodes, you also reduce the likelihood of several CPU-bound processes competing for the same CPU at the same time.

Most organizations can easily afford to purchase enough cluster nodes to eliminate the CPU as the performance bottleneck, and building a cluster doesn't help you prevent the other three performance bottlenecks. Therefore, the second cluster design consideration (the impact of a single node failure) is likely to influence capacity planning in most organizations more than performance.

## OTHER PERFORMANCE AND DESIGN CONSIDERATIONS

In this greatly simplified discussion, we are leaving out two performance issues that are worth considering: the upper limit on the number of user sessions that can be placed on a single node (which is based on the profile<sup>1</sup> of the application or applications that will be running on each cluster node), and the potentially huge differences in the various profiles of the different applications that will be running on the cluster. In Chapters 16 and 18 this topic will be discussed further. As the processing ceiling all but disappears in a cluster environment, the performance of the cluster file system or the backend database takes on an increasingly important role. (Note that the backend database may also run on a cluster. See the section "Clustering Zope" in Chapter 20 for one example.)

<sup>1</sup> CPU and memory requirements, disk I/O, network bandwidth, and so on.

### Capacity Planning and the Impact of a Single Node Failure

The second and more significant design consideration for most organizations is the business impact of a node failure, and the ability to perform routine (planned) maintenance. When you are deciding how many nodes to purchase, the business impact of a single node failure on the enterprise or on the user community may be the single most important design consideration.

Purchasing more nodes than the number needed for peak CPU processing may make sense because the extra nodes will reduce the impact of the failure of a single node on the end-user community, and it will also make cluster maintenance easier. (The cluster administrator can remove a node from the cluster for maintenance, and the cluster will still have enough processing power to continue to get the job done.)

Assuming your budget will allow you to purchase more nodes than you need to adequately meet your CPU processing requirements, you'll need to examine how your workload will be distributed across the cluster nodes and determine the business impact of a node failure (for example, how many users would be affected or how many user sessions would be affected). You can then provide management with the total cost of ownership for each additional cluster node and explain the benefits of purchasing additional nodes.

## CONVERTING LEGACY APPLICATIONS FROM UNIX TO LINUX

If you are converting to a Linux Enterprise Cluster from a Unix operating system, there is an additional advantage of using a Linux cluster during the conversion process. As you test and develop the new Linux cluster, you can make the cluster nodes NFS clients of your existing Unix server (assuming, of course, your existing Unix server can become an NFS server). While this server is not likely to make a very good NFS server for production purposes, it will make an adequate server that is capable of providing access to production data files during the testing phase. The server will arbitrate lock access, making it possible to test the Linux cluster using live data, while still running on your old system.<sup>1</sup>

<sup>1</sup> Note that this will not work if your software applications have to contend with conversion *endian* issues. Sparc hardware, for example, uses big-endian byte ordering, while Intel hardware uses little-endian byte ordering. To learn more about endian conversion issues and Linux, see *Gulliver's Travels* or IBM's "Solaris-to-Linux porting guide" (<http://www-106.ibm.com/developerworks/linux/library/l-solar>).

## In Conclusion

Building a cluster allows you to remove the CPU performance bottleneck and improve the reliability and availability of your user applications. With careful testing and planning, you can build a Linux Enterprise Cluster that can run mission-critical applications with no single point of failure. And with the proper hardware capacity (that is, the right number of nodes) you can maintain your cluster during normal business hours without affecting the performance of end-user applications.

# 11

## **THE LINUX VIRTUAL SERVER: INTRODUCTION AND THEORY**



This chapter will introduce the cluster load-balancing software called IP Virtual Server (IPVS). The IPVS software is a collection of kernel patches that were merged into the stock version of the Linux kernel starting with version 2.4.23. When combined with the kernel's routing and packet-filtering capabilities (discussed in Chapter 2) the IPVS-enabled kernel lets you turn any computer running Linux into a cluster load balancer. Together, the IPVS-enabled cluster load balancer and the cluster nodes are called a Linux Virtual Server (LVS).

The LVS cluster load balancer accepts all incoming client computer requests for services and decides which cluster node should reply to each request. The load balancer is sometimes called an LVS Director or simply a Director. In this book the terms LVS Director, Director, and load balancer all refer to the same thing.

The nodes inside an LVS cluster are called *real servers*, and the computers that connect to the cluster to request its services are called *client computers*. The client computers, the Director, and the real servers communicate with each other using IP addresses the same way computers have always exchanged packets over a network; however, to make it easier to discuss this network communication, the LVS community has developed a naming convention to describe each type of IP address based on its role in the network conversation. So before we consider the different types of LVS clusters and the choices you have for distributing your workload across the cluster nodes (called *scheduling methods*), let's look at this naming convention and see how it helps describe the LVS cluster.

## LVS IP Address Name Conventions

In an LVS cluster, we cannot refer to network addresses as simply “IP addresses.” Instead, we must distinguish between different types of IP addresses based on the roles of the nodes inside the cluster. Here are four basic types of IP addresses used in a cluster:

### **Virtual IP (VIP) address**

The IP address the Director uses to offer services to client computers

### **Real IP (RIP) address**

The IP address used on the cluster nodes

### **Director's IP (DIP) address**

The IP address the Director uses to connect to the D/RIP network

### **Client computer's IP (CIP) address**

The IP address assigned to a client computer that it uses as a source IP address for requests sent to the cluster

## ***The Virtual IP (VIP)***

The IP address that client computers use to connect to the services offered by the cluster are called virtual IP addresses (VIPs). VIPs are IP aliases or secondary IP addresses on the NIC that connects the Director to the normal, public network.<sup>1</sup> The LVS VIP is important because it is the address that client computers will use when they connect to the cluster. Client computers send packets from their IP address to the VIP address to access cluster services. You tell the client computers the VIP address using a naming service (such as DNS, DDNS, WINS, LDAP, or NIS), and this is the only name or address that client computers ever need to know in order to use the services inside the cluster. (The remaining IP addresses inside the cluster are not known to the client computer.)

---

<sup>1</sup> As you'll see in Chapter 15, the VIPs in a high-availability configuration are under Heartbeat's control.

A single Director can have multiple VIPs offering different services to client computers, and the VIPs can be public IP addresses that can be routed on the Internet, though this is not required. What is required, however, is that the client computers be able to access the VIP or VIPs of the cluster. (As we'll see later, an LVS-NAT cluster can use a private intranet IP address for the nodes inside the cluster, even though the VIP on the Director is a public Internet IP address.)

### ***The Real IP (RIP)***

In LVS terms, a node offering services to the outside world is called a real server. (We will use the terms *cluster node* and *real server* interchangeably throughout this book.) The IP address used on the real server is therefore called a *real IP address* (RIP).

The RIP address is the IP address that is *permanently* assigned to the NIC that connects the real server to the same network as the Director. We'll call this network cluster network or the Director/real-server network (*D/RIP network*). The Director uses the RIP address for normal network communication with the real servers on the D/RIP network, but only the Director needs to know how to talk to this IP address.

### ***The Director's IP (DIP)***

The Director's IP (DIP) address is used on the NIC that connects the Director to the D/RIP network. As requests for cluster services are received on the Director's VIP, they are forwarded out the DIP to reach a cluster node. As is discussed in Chapter 15, the DIP and the VIP can be on the same NIC.

### ***The Client Computer's IP (CIP)***

The client computer's IP (CIP) address may be a local, private IP address on the same network as the VIP, or it may be a public IP address on the Internet.

### ***IP Addresses in an LVS Cluster***

A schematic of an LVS cluster containing one cluster node (one real server) and one Director is shown in Figure 11-1.

In this figure, the Director's public NIC, using the VIP address, is connected to the company network switch, or this could be an Internet router. The NIC connected to the D/RIP network has the DIP address on it. Figure 11-1 shows a D/RIP network hub or switch, though as we'll see, some cluster configurations can use the same network switch for both the public NIC and the cluster NIC (in LVS-DR clusters). Finally, the NIC that connects the real server to the D/RIP network is shown with a RIP address. Incoming packets destined for the real server arrive on the VIP, pass through the Director and out its DIP, and finally reach the real server's RIP.

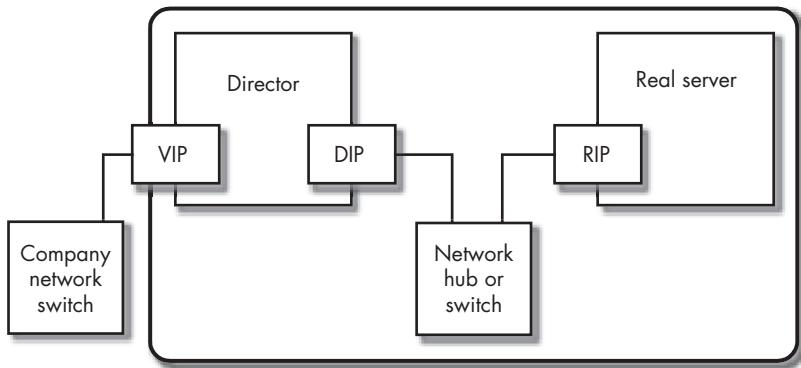


Figure 11-1: LVS cluster schematic

## Types of LVS Clusters

Now that we've looked at some of the IP address name conventions used to describe LVS clusters, let's examine the LVS packet-forwarding methods.

LVS clusters are usually described by the type of forwarding method the LVS Director uses to relay incoming requests to the nodes inside the cluster. Three methods are currently available:

- Network address translation (LVS-NAT)
- Direct routing (LVS-DR)
- IP tunneling (LVS-TUN)

Although more than one forwarding method can be used on a single Director (the forwarding method can be chosen on a per-node basis), I'll simplify this discussion and describe LVS clusters as if the Director is only capable of using one forwarding method at a time.

The best forwarding method to use with a Linux Enterprise Cluster is LVS-DR (and the reasons for this will be explained shortly), but an LVS-NAT cluster is the easiest to build. If you have never built an LVS cluster and want to use one to run your enterprise, you may want to start by building a small LVS-NAT cluster in a lab environment using the instructions in Chapter 12, and then learn how to convert this cluster into an LVS-DR cluster as described in Chapter 13. The LVS-TUN cluster is not generally used for mission-critical applications and is mentioned in this chapter only for the sake of completeness. It will not be described in detail.

### **Network Address Translation (LVS-NAT)**

In an LVS-NAT configuration, the Director uses the Linux kernel's ability (from the kernel's Netfilter code) to translate network IP addresses and ports as packets pass through the kernel. (This is called Network Address Translation (NAT), and it was introduced in Chapter 2).

**NOTE** We'll examine the LVS-NAT network communication in more detail in Chapter 12.

As shown in Figure 11-2, a request for a cluster service is received by the Director on its VIP, and the Director forwards this request to a cluster node on its RIP. The cluster node then replies to the request by sending the packet back through the Director so the Director can perform the translation that is necessary to convert the cluster node's RIP address into the VIP address that is owned by the Director. This makes it appear to client computers outside the cluster as if all packets are sent and received from a single IP address (the VIP).

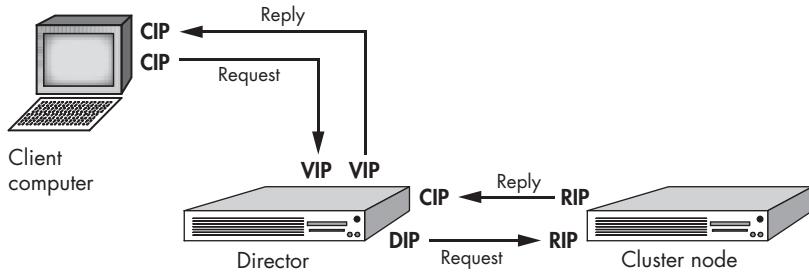


Figure 11-2: LVS-NAT network communication

### Basic Properties of LVS-NAT

The LVS-NAT forwarding method has several basic properties:

- The cluster nodes need to be on the same network (VLAN or subnet) as the Director.
- The RIP addresses of the cluster nodes normally conform to RFC 1918<sup>2</sup> (that is, they are private, non-routable IP addresses used only for intra-cluster communication).
- The Director intercepts all communication (network packets going in either direction) between the client computers and the real servers.
- The cluster nodes use the Director's DIP as their default gateway for reply packets to the client computers.
- The Director can remap network port numbers. That is, a request received on the Director's VIP on one port can be sent to a RIP inside the cluster on a different port.
- Any type of operating system can be used on the nodes inside the cluster.
- A single Director can become the bottleneck for the cluster.

At some point, the Director will become a bottleneck for network traffic as the number of nodes in the cluster increases, because all of the reply packets from the cluster nodes must pass through the Director. However, a

<sup>2</sup> RFC 1918 reserves the following IP address blocks for private intranets:

10.0.0.0 through 10.255.255

172.16.0.0 through 172.31.255.255

192.168.0.0 through 192.168.255.255

400 MHz processor can saturate a 100 Mbps connection, so the network is more likely to become the bottleneck than the LVS Director under normal circumstances.

The LVS-NAT cluster is more difficult to administer than an LVS-DR cluster because the cluster administrator sitting at a computer outside the cluster is blocked from direct access to the cluster nodes, just like all other clients. When attempting to administer the cluster from outside, the administrator must first log on to the Director before being able to telnet or ssh to a specific cluster node. If the cluster is connected to the Internet, and client computers use a web browser to connect to the cluster, having the administrator log on to the Director may be a desirable security feature of the cluster, because an administrative network can be used to allow only internal IP addresses shell access to the cluster nodes. However, in a Linux Enterprise Cluster that is protected behind a firewall, you can more easily administer cluster nodes when you can connect directly to them from outside the cluster. (As we'll see in Part IV of this book, the cluster node manager in an LVS-DR cluster can sit outside the cluster and use the Mon and Ganglia packages to gain diagnostic information about the cluster remotely.)

### ***Direct Routing (LVS-DR)***

In an LVS-DR configuration, the Director forwards all incoming requests to the nodes inside the cluster, but the nodes inside the cluster send their replies directly back to the client computers (the replies do not go back through the Director).<sup>3</sup> As shown in Figure 11-3, the request from the client computer or CIP is sent to the Director's VIP. The Director then forwards the request to a cluster node or real server using the same VIP destination IP address (we'll see how the Director does this in Chapter 13). The cluster node then sends a reply packet directly to the client computer, and this reply packet uses the VIP as its source IP address. The client computer is thus fooled into thinking it is talking to a single computer, when in reality it is sending request packets to one computer and receiving reply packets from another.

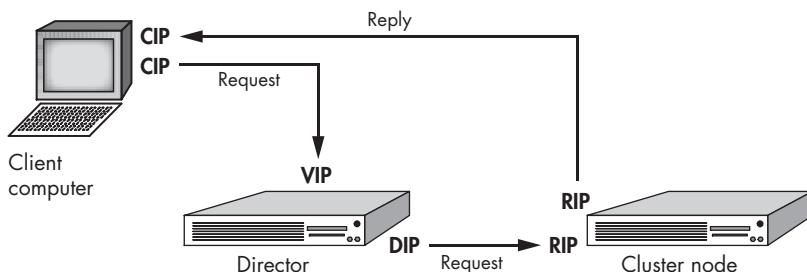


Figure 11-3: LVS-DR network communication

<sup>3</sup> Without the special LVS "martian" modification kernel patch applied to the Director, the normal LVS-DR Director will simply drop reply packets if they try to go back out through the Director.

## Basic Properties of LVS-DR

These are the basic properties of a cluster with a Director that uses the LVS-DR forwarding method:

- The cluster nodes must be on the same network segment as the Director.<sup>4</sup>
- The RIP addresses of the cluster nodes do not need to be private IP addresses (which means they do not need to conform to RFC 1918).
- The Director intercepts inbound (but not outbound) communication between the client and the real servers.
- The cluster nodes (normally) do not use the Director as their default gateway for reply packets to the client computers.
- The Director cannot remap network port numbers.
- Most operating systems can be used on the real servers inside the cluster.<sup>5</sup>
- An LVS-DR Director can handle more real servers than an LVS-NAT Director.

Although the LVS-DR Director can't remap network port numbers the way an LVS-NAT Director can, and only certain operating systems can be used on the real servers when LVS-DR is used as the forwarding method,<sup>6</sup> LVS-DR is the best forwarding method to use in a Linux Enterprise Cluster because it allows you to build cluster nodes that can be directly accessed from outside the cluster. Although this may represent a security concern in some environments (a concern that can be addressed with a proper VLAN configuration), it provides additional benefits that can improve the reliability of the cluster and that may not be obvious at first:

- If the Director fails, the cluster nodes become distributed servers, each with their own IP address. (Client computers on the internal network, in other words, can connect directly to the LVS-DR cluster node using their RIP addresses.) You would then tell users which cluster-node RIP address to use, or you could employ a simple round-robin DNS configuration to hand out the RIP addresses for each cluster node until the Director is operational again.<sup>7</sup> You are protected, in other words, from a catastrophic failure of the Director and even of the LVS technology itself.<sup>8</sup>

---

<sup>4</sup> The LVS-DR forwarding method requires this for normal operation. See Chapter 13 for more info on LVS-DR clusters.

<sup>5</sup> The operating system must be capable of configuring the network interface to avoid replying to ARP broadcasts. For more information, see “ARP Broadcasts and the LVS-DR Cluster” in Chapter 13.

<sup>6</sup> The real servers inside an LVS-DR cluster must be able to accept packets destined for the VIP without replying to ARP broadcasts for the VIP (see Chapter 13).

<sup>7</sup> See the “Load Sharing with Heartbeat—Round-Robin DNS” section in Chapter 8 for a discussion of round-robin DNS.

<sup>8</sup> This is unlikely to be a problem in a properly built and properly tested cluster configuration. We'll discuss how to build a highly available Director in Chapter 15.

- To test the health and measure the performance of each cluster node, monitoring tools can be used on a cluster node manager that sits outside the cluster (we'll discuss how to do this using the Mon and Ganglia packages in Part IV of this book).
- To quickly diagnose the health of a node, irrespective of the health of the LVS technology or the Director, you can telnet, ping, and ssh directly to any cluster node when a problem occurs.
- When troubleshooting what appear to be software application problems, you can tell end-users<sup>9</sup> how to connect to two different cluster nodes directly by IP (RIP) address. You can then have the end-user perform the same task on each node, and you'll know very quickly whether the problem is with the application program or one of the cluster nodes.

**NOTE** *In an LVS-DR cluster, packet filtering or firewall rules can be installed on each cluster node for added security. See the LVS-HOWTO at <http://www.linuxvirtualserver.org> for a discussion of security issues and LVS. In this book we assume that the Linux Enterprise Cluster is protected by a firewall and that only client computers on the trusted network can access the Director and the real servers.*

### **IP Tunneling (LVS-TUN)**

IP tunneling can be used to forward packets from one subnet or virtual LAN (VLAN) to another subnet or VLAN even when the packets must pass through another network or the Internet. Building on the IP tunneling capability that is part of the Linux kernel, the LVS-TUN forwarding method allows you to place cluster nodes on a cluster network that is not on the same network segment as the Director.

**NOTE** *We will not use the LVS-TUN forwarding method in any recipes in this book, and it is only included here for the sake of completeness.*

The LVS-TUN configuration enhances the capability of the LVS-DR method of packet forwarding by encapsulating inbound requests for cluster services from client computers so that they can be forwarded to cluster nodes that are not on the same physical network segment as the Director. For example, a packet is placed inside another packet so that it can be sent across the Internet (the inner packet becomes the data payload of the outer packet). Any server that knows how to separate these packets, no matter where it is on your intranet or the Internet, can be a node in the cluster, as shown in Figure 11-4.<sup>10</sup>

---

<sup>9</sup> Assuming the client computer's IP address, the VIP and the RIP are all private (RFC 1918) IP addresses.

<sup>10</sup> If your cluster needs to communicate over the Internet, you will likely need to encrypt packets before sending them. This can be accomplished with the IPSec protocol (see the FreeS/WAN project at <http://www.freeswan.org> for details). Building a cluster that uses IPSec is outside the scope of this book.

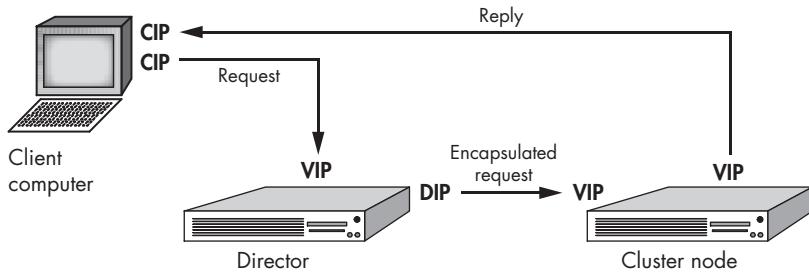


Figure 11-4: LVS-TUN network communication

The arrow connecting the Director and the cluster node in Figure 11-4 shows an encapsulated packet (one stored within another packet) as it passes from the Director to the cluster node. This packet can pass through any network, including the Internet, as it travels from the Director to the cluster node.

### Basic Properties of LVS-TUN

An LVS-TUN cluster has the following properties:

- The cluster nodes do not need to be on the same physical network segment as the Director.
- The RIP addresses must not be private IP addresses.
- The Director can normally only intercept inbound communication between the client and the cluster nodes.
- The return packets from the real server to the client must not go through the Director. (The default gateway can't be the DIP; it must be a router or another machine separate from the Director.)
- The Director cannot remap network port numbers.
- Only operating systems that support the IP tunneling protocol<sup>11</sup> can be servers inside the cluster. (See the comments in the `configure-lvs` script included with the LVS distribution to find out which operating systems are known to support this protocol.)

We won't use the LVS-TUN forwarding method in this book because we want to build a cluster that is reliable enough to run mission-critical applications, and separating the Director from the cluster nodes only increases the potential for a catastrophic failure of the cluster. Although using geographically dispersed cluster nodes might seem like a shortcut to building a disaster recovery data center, such a configuration doesn't improve the reliability of the cluster, because anything that breaks the connection between the Director and the cluster nodes will drop all client connections to the remote cluster nodes. A Linux Enterprise Cluster must be able to

<sup>11</sup> Search the Internet for the "Linux 2.4 Advanced Routing HOWTO" for more information about the IP tunneling protocol.

share data with all applications running on all cluster nodes (this is the subject of Chapter 16). Geographically dispersed cluster nodes only decrease the speed and reliability of data sharing.

## LVS Scheduling Methods

Having discussed three ways to forward packets to the nodes inside the cluster, let's look at how to distribute the workload across the cluster nodes. When the Director receives an incoming request from a client computer to access a cluster service on its VIP, it has to decide which cluster node should get the request. The *scheduling methods* the Director can use to make this decision fall into two basic categories: *fixed scheduling* and *dynamic scheduling*.

**NOTE** *When a node needs to be taken out of the cluster for maintenance, you can set its weight to 0 using either a fixed or a dynamic scheduling method. When a cluster node's weight is 0, no new connections will be assigned to it. Maintenance can be performed after all of the users log out normally at the end of the day. We'll discuss cluster maintenance in detail in Chapter 19.*

### **Fixed (or Non-dynamic) Scheduling Methods**

In the case of fixed, or non-dynamic, scheduling methods, the Director selects the cluster node to use for the inbound request without checking to see how many of the previously assigned connections are active. Here is the current list of fixed scheduling methods:

#### **Round-robin (RR)**

When a new request is received, the Director picks the next server on its list of servers, rotating through them in an endless loop.

#### **Weighted round-robin (WRR)**

You assign each cluster node a *weight* or *ranking*, based on how much processing load it can handle. This weight is then used, along with the round-robin technique, to select the next cluster node to be used when a new request is received, regardless of the number of connections that are still active. A server with a weight of 2 will receive twice the number of new connections as a server with a weight of 1. If you change the weight of a server to 0, no new connections will be allowed to the server (but currently active connections will not be dropped). We'll look at how LVS uses this weight to balance the incoming workload in the "Weighted Least-Connection (WLC)" section of this chapter.

#### **Destination hashing**

This method always sends requests for the same IP address to the same server in the cluster. Like the locality-based least-connection (LBLC) scheduling method (which will be discussed shortly), this method is useful when the servers inside the cluster are really cache or proxy servers.

## Source hashing

This method can be used when the Director needs to be sure the reply packets are sent back to the same router or firewall that the requests came from. This scheduling method is normally only used when the Director has more than one physical network connection, so that the Director knows which firewall or router to send the reply packet back through to reach the proper client computer.

## Dynamic Scheduling Methods

Dynamic scheduling methods give you more control over the incoming workload, with little or no penalty, since they only require a small amount of extra processing load on the Director. When dynamic scheduling methods are used, the Director keeps track of the number of *active* and *inactive* connections for each cluster node and uses this information to determine which cluster node to use when a new request arrives for a cluster service. An active connection is a TCP network session that remains open (in the ESTABLISHED state) while the client computer and cluster node are sending data to each other. In a Linux Enterprise Cluster, telnet or ssh sessions remain active as long as the user is logged on.<sup>12</sup>

An inactive connection, on the other hand, is any network connection that is not in the ESTABLISHED state. If a TCP inactivity timeout causes the connection to drop, or if the client computer sends a FIN packet to close the connection, LVS keeps the connection in the IPVS table for a brief period in case subsequent packets for the connection arrive to reestablish the TCP connection. This may happen, for example, when packets are resent due to transmission problems. The Director, in other words, attempts to protect the integrity of the connection between the client computer and the cluster node when there are minor network transmission problems.

**NOTE** *This discussion is more theoretical than practical when using telnet or ssh for user sessions in a Linux Enterprise Cluster. The profile of the user applications (the way the CPU, disk, and network are used by each application) varies over time, and user sessions last a long time (hours, if not days). Thus, balancing the incoming workload offers only limited effectiveness when balancing the total workload over time.*

As of this writing, the following dynamic scheduling methods are available:

- Least-connection (LC)
- Weighted least-connection (WLC)
- Shortest expected delay (SED)
- Never queue (NQ)

---

<sup>12</sup> Of course, it is possible that no data may pass between the cluster node and client computer during the telnet session for long periods of time (when a user runs a batch job, for example). See the discussion of the TCP session timeout value in the “LVS Persistence” section in Chapter 14.

- Locality-based least-connection (LBLC)
- Locality-based least-connection with replication scheduling (LBLCR)

### Least-Connection (LC)

With the least-connection scheduling method, when a new request for a service running on one of the cluster nodes arrives at the Director, the Director looks at the number of active and inactive connections to determine which cluster node should get the request.

The mathematical calculation performed by the Director to make this decision is as follows: For each node in the cluster, the Director multiplies the number of active connections the cluster node is currently servicing by 256, and then it adds the number of inactive connections (recently used connections) to arrive at an *overhead* value for each node. The node with the lowest overhead value wins and is assigned the new incoming request for service.<sup>13</sup>

If the mathematical calculation results in the same overhead value for all of the cluster nodes, the first node found in the IPVS table of cluster nodes is selected.<sup>14</sup>

### Weighted Least-Connection (WLC)

The weighted least-connection scheduling method combines the least-connection method and a specified weight or ranking for each server to select the cluster node. (This is the default selection method if you do not specify one.) This method was intended for use in clusters with nodes that have differing processing capabilities.

The Director determines which cluster node to assign to a new inbound request for a cluster service by first calculating the overhead value (as described earlier in the discussion of the LC scheduling method) for each cluster node and then dividing this value by the weight you have assigned to the cluster node to arrive at a WLC value for each cluster node. The cluster node with the lowest WLC value wins, and the incoming request is assigned to that node.<sup>15</sup>

If the WLC value for all of the nodes is the same, the first node found in the list of cluster nodes is selected. (We'll talk more about this list, which is called the IPVS table, in the next three chapters.)

The WLC scheduling method is a good choice for a Linux Enterprise Cluster because it does a good job of balancing the workload of a typical enterprise.

---

<sup>13</sup> See the 143 lines of source code in the `ip_vs_lc.c` file in the LVS source code distribution (version 1.0.10).

<sup>14</sup> The first cluster node that is capable of responding for the service (or port number) the client computer is requesting. We'll discuss the IP Virtual Server or IPVS table in more detail in the next three chapters.

<sup>15</sup> The code actually uses a mathematical trick to multiply instead of divide to find the node with the best overhead-to-weight value, because floating-point numbers are not allowed inside the kernel. See the `ip_vs_wlc.c` file included with the LVS source code for details.

## Shortest Expected Delay (SED)

SED is a recent addition to the LVS scheduling methods, and it may offer a slight improvement over the WLC method for services that use TCP and remain in an active state while the cluster node is processing each request (large batch jobs are a good example of this type of request).

The SED calculation is performed as follows: The overhead value for each cluster node is calculated by adding 1 to the number of active connections. The overhead value is then divided by the weight you assigned to each node to arrive at the SED value. The cluster node with the lowest SED value wins.

There are two things to notice about the SED scheduling method:

- It does not use the number of inactive connections when determining the overhead of each cluster node.
- It adds 1 to the number of active connections to anticipate what the overhead will look like after the new incoming connection has been assigned.

For example, let's say you have two cluster nodes and one is three times faster than the other (one has a 1 GHz processor and the other has a 3 GHz processor<sup>16</sup>), so you decide to assign the slower machine a weight of 1 and the faster machine a weight of 3. Suppose the cluster has been up and running for a while, and the slower node has 10 active connections and the faster node has 30 active connections. When the next new request arrives, the Director must decide which cluster node to assign. If this new request is not added to the number of active connections for each of the cluster nodes, the SED values would be calculated as follows:

<b>Slower node (1 GHz processor)</b>	10 active connections / weight 1 = 10
--------------------------------------	---------------------------------------

<b>Faster node (3 GHz processor)</b>	30 active connections / weight 3 = 10
--------------------------------------	---------------------------------------

Because the SED values are the same, the Director will pick whichever node happens to appear first in its table of cluster nodes. If the slower cluster node happens to appear first in the table of cluster nodes, it will be assigned the new request even though it is the *slower* node.

If the new connection is first added to the number of active connections, however, the calculations look like this:

<b>Slower node (1 GHz processor)</b>	11 active connections / weight 1 = 11
--------------------------------------	---------------------------------------

<b>Faster node (3 GHz processor)</b>	31 active connections / weight 3 = 10.34
--------------------------------------	--

The faster node now has the lower SED value, so it is properly assigned the new connection request.

---

<sup>16</sup> For the moment, we'll ignore everything else about the performance capabilities of these two nodes.

A side effect of adding 1 to the number of active connections is that a cluster node may sit idle even though multiple requests are assigned to another cluster node. For example, let's use our same two cluster nodes, but this time we'll assume the slower cluster node has no active connections and the faster node has one active connection. The SED calculation for each node looks like this (recall that 1 is added to the number of active connections):

<b>Slower node (1 GHz processor)</b>	1 active connection / weight 1 = 1
<b>Faster node (3 GHz processor)</b>	2 active connections / weight 3 = .67

So the new request gets assigned to the faster cluster node even though the slower cluster node is idle. This may or may not be desirable behavior, so another scheduling method was developed, called *never queue*.

### **Never Queue (NQ)**

This scheduling method enhances the SED scheduling method by adding one new feature: if a cluster node has no active connections, it is always assigned the new incoming request for service, regardless of the result of the calculated SED values for each cluster node.

### **Locality-Based Least-Connection (LBLC)**

Directors can also direct outbound traffic to a set of transparent proxy servers. In this configuration, the cluster nodes are transparent proxy or web cache servers that sit between client computers and the Internet.<sup>17</sup>

When the LBLC scheduling method is used, the Director attempts to send all requests destined for a particular IP address (a particular web server) to the same transparent proxy server (cluster node). In other words, the first time a request comes in for a web server on the Internet, the Director will pick one proxy server to service this destination IP address using a slightly modified version of the WLC scheduling method,<sup>18</sup> and all future requests for this same destination IP address will continue to go to the same proxy server. This method of load balancing, like the destination-hashing scheduling method described previously, is, therefore, a type of destination IP load balancing.

The Director will continue to send all requests for a particular destination IP address to the same cluster node (the same transparent proxy server) until it sees that another node in the cluster has a WLC value that is half of the WLC value of the assigned cluster node. When this happens, the

---

<sup>17</sup> A proxy server stores a copy of the web pages, or server responses, that have been requested recently so that future requests for the same information can be serviced without the need to ask the original Internet server for the same information again. See the "Transparent Proxy with Linux and Squid mini-HOWTO," available at <http://www.faqs.org/docs/Linux-mini/TransparentProxy.html> among other places.

<sup>18</sup> The modification is that the overhead value is calculated by multiplying the active connections by 50 instead of by 256.

Director will reassign the cluster node that is responsible for the destination IP address (usually an Internet web server) by selecting the least loaded cluster node using the modified WLC scheduling method.

In this method, the Director tries to associate only one proxy server to one destination IP address. To do this, the Director maintains a table of destination IP addresses and their associated proxy servers. This method of load balancing attempts to maximize the number of cache hits on the proxy servers, while at the same time reducing the amount of redundant, or replicated, information on these proxy servers.

### **Locality-Based Least-Connection with Replication Scheduling (LBLCR)**

The LBLCR scheduling method (which is also a form of destination IP load balancing) attempts to improve on the LBLC scheduling method by maintaining a set of proxy servers that can service each destination IP address. When a new connection request comes in, the Director will select the proxy server with the fewest number of active connections from this set of servers.

**NOTE** *See the file /proc/net/ip\_vs\_lblcr for the servers associated with each destination IP address on a Director that is using the LBLC scheduling method.*

At the time of a new connection request from a client computer, proxy servers are added to this set for a particular destination IP address when the Director notices a cluster node (proxy server) that has a WLC<sup>19</sup> value equal to half of the WLC value of the least loaded node in the set. When this happens, the cluster node with the lowest WLC value is added to the set and is assigned the new incoming request.

Proxy servers are removed from the set when the list of servers in the set has not been modified within the last six minutes (meaning that no new proxy servers have been added, and no proxy servers have been removed). If this happens, the Director will remove the server with the most active connections from the set. The proxy server (cluster node) will continue to service the existing active connections, but any new requests will be assigned to a different server still in the set.

**NOTE** *None of these scheduling methods take into account the processing load or disk or network I/O a cluster node is experiencing, nor do they anticipate how much load a new inbound request will generate when assigning the incoming request. You may see references to two projects that were designed to address this need, called feedbackd and LVS-KISS. As of this writing, however, these projects are not widely deployed, so you should carefully consider them before using them in production.*

---

<sup>19</sup> Again the overhead value used to calculate the WLC value is calculated by multiplying the number of active connections by 50 instead of 256.

## In Conclusion

This chapter introduced the Linux Virtual Server (LVS) cluster load balancer called the LVS Director and the forwarding and scheduling methods it uses to pass incoming requests for cluster services to the nodes inside the cluster.

The forwarding methods used by the Director are called Network Address Translation, direct routing, and tunneling, or LVS-NAT, LVS-DR, and LVS-TUN respectively. Although the Director can select a different forwarding method for each node inside the cluster, most load balancers use only one type of forwarding method for all of the nodes in the cluster to keep the setup simpler. As I've discussed briefly in this chapter, an LVS-DR cluster is the best type of cluster to use when building an enterprise-class cluster.

The LVS scheduling methods were also introduced in this chapter. The Director uses a scheduling method to evenly distribute the workload across the cluster nodes, and these methods fall into two categories: fixed and dynamic. Fixed scheduling methods differ from dynamic methods in that no information about the current number of active connections is used when selecting a cluster node using a fixed method.

The next two chapters take a closer look at the LVS-NAT and LVS-DR forwarding methods.

# 12

## THE LVS-NAT CLUSTER



Recall from the last chapter that the computers outside the cluster that access cluster services are called *client computers*, the cluster load balancer is called the *Director*, and nodes inside the cluster are called *real servers*.

In this chapter, I'll look at how client computers access services on real servers in a Linux Virtual Server Network Address Translation (LVS-NAT) cluster, and at how the Director uses the LVS-NAT forwarding method. I'll then give you a recipe for building an LVS-NAT web cluster.

### How Client Computers Access LVS-NAT Cluster Resources

To understand how client computers access cluster services, we'll use the example of a client that is connected to the Internet and is calling up a web page offered by the LVS-NAT cluster. (We'll look at how to build this cluster in the "Building an LVS-NAT Web Cluster" section, later in this chapter.) Figure 12-1 shows the beginning of this network conversation.<sup>1</sup>

---

<sup>1</sup> For the moment, we will ignore the lower-level TCP connection establishment packets (called SYN and ACK packets).

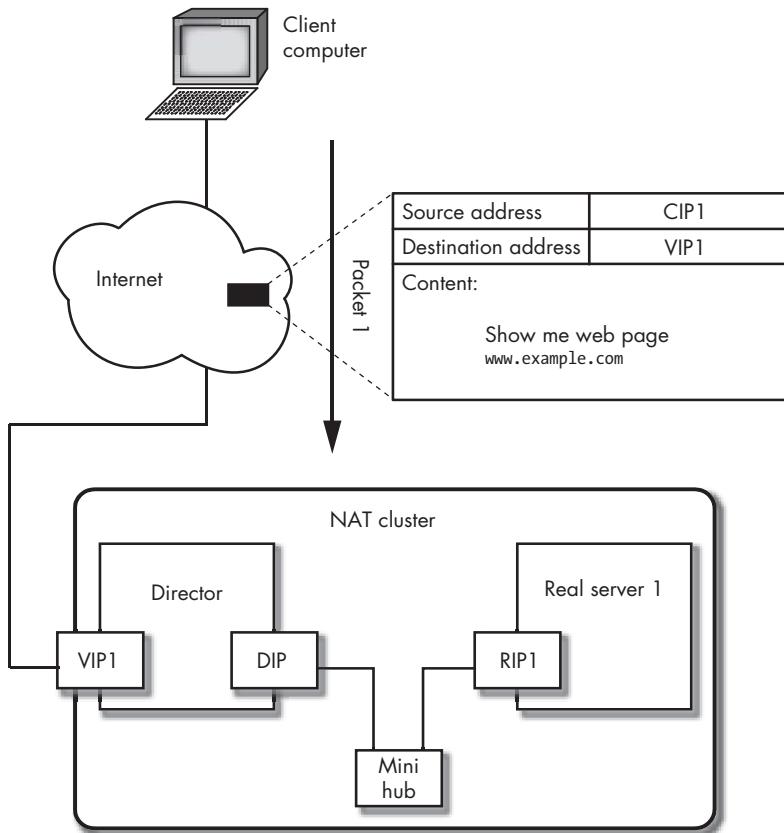


Figure 12-1: In packet 1 the client computer sends a request to the LVS-NAT cluster

As you can see in the figure, the client computer initiates the network conversation with the cluster by sending the first packet from a client IP (CIP1), through the Internet, to the VIP1 address on the Director. The source address of this packet is CIP1, and the destination address is VIP1 (which the client knew, thanks to a naming service like DNS). The packet's data payload is the HTTP request from the client computer requesting the contents of a web page (an HTTP GET request).

When the packet arrives at the Director, the LVS code in the Director uses one of the scheduling methods that were introduced in Chapter 11 to decide which real server should be assigned to this request. Because our cluster network has only one cluster node, the Director doesn't have much choice; it has to use real server 1, which has the address RIP1.

**NOTE** *The Director does not examine or modify the data payload of the packet.*

In Chapter 14 we'll discuss what goes on inside the kernel as the packet passes through the Director, but for now we only need to know that the Director has made no significant change to the packet—the Director simply changed the destination address of the packet (and possibly the destination port number of the packet).

**NOTE** LVS-NAT is the only forwarding method that allows you to remap network port numbers as packets pass through the Director.

This packet, with a new destination address and possibly a new port number, is sent from the Director to the real server, as shown in Figure 12-2, and now it's called packet 2. Notice that the source address in packet 2 is the client IP (CIP) address taken from packet 1, and the data payload (the HTTP request) remains unchanged. What has changed is the destination address—the Director changed the destination of the original packet to one of the real server RIP addresses inside the cluster (RIP1 in this example). Figure 12-2 shows the packet inside the LVS-NAT cluster on the network cable that connects the DIP to the cluster network.

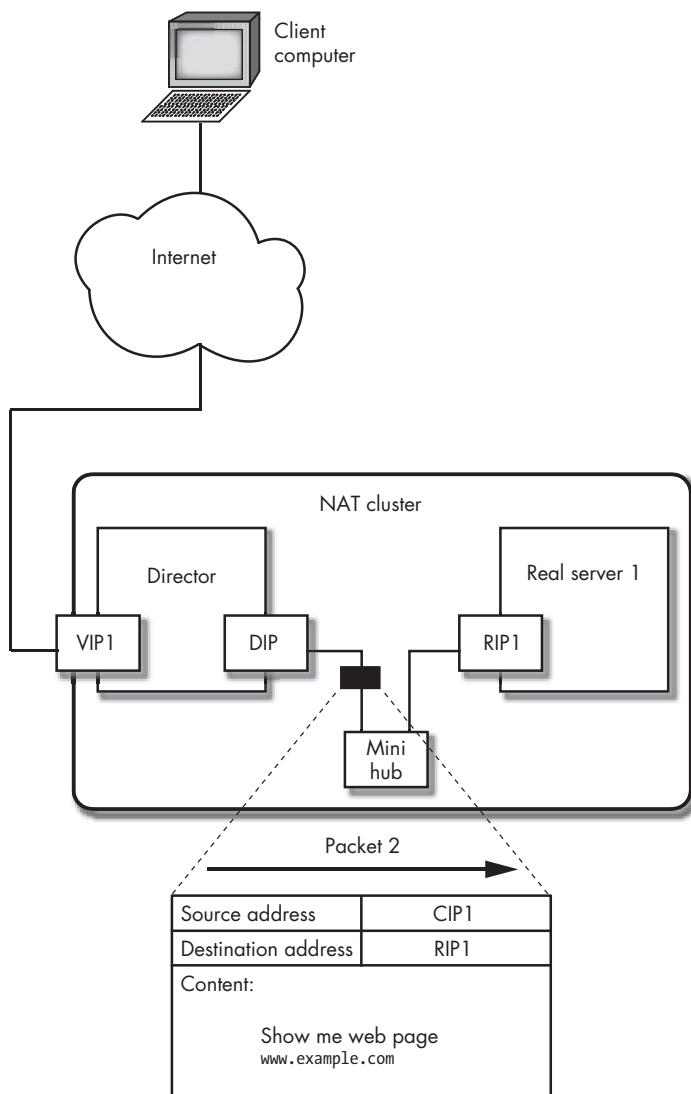


Figure 12-2: In packet 2 the Director forwards the client computer's request to a cluster node

When packet 2 arrives at real server 1, the HTTP server replies with the contents of the web page, as shown in Figure 12-3. Because the request was from CIP1, packet 3 (the return packet) will have a destination address of CIP1 and a source address of RIP1. In LVS-NAT, the default gateway for the real server is normally an IP address on the Director (the DIP), so the reply packet is routed through the director.<sup>2</sup>

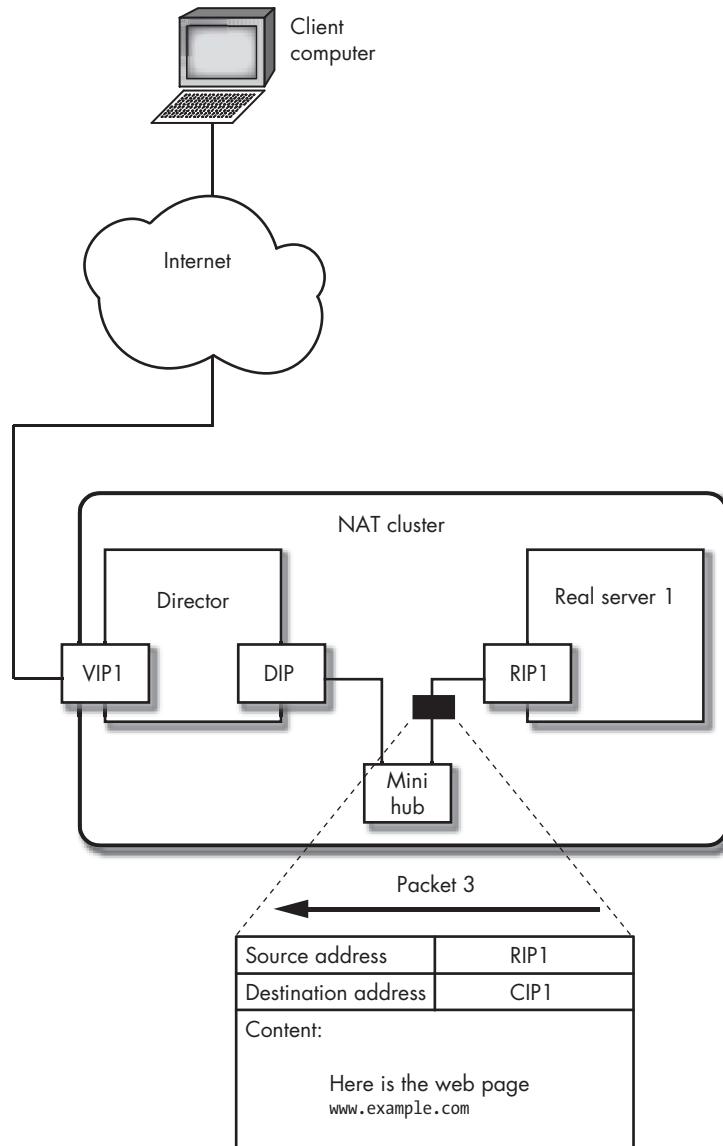


Figure 12-3: In packet 3 the cluster node sends a reply back through the Director

<sup>2</sup> The packets must pass back through the Director in an LVS-NAT cluster.

Packet 3 is sent through the cluster network to the Director, and its data payload is the web page that was requested by the client computer. When the Director receives packet 3, it passes the packet back through the kernel, and the LVS software rewrites the source address of the packet, changing it from RIP1 to VIP1. The data payload does not change as the packet passes through the Director—it still contains the HTTP reply from real server 1. The Director then forwards the packet (now called packet 4) back out to the Internet, as shown in Figure 12-4.

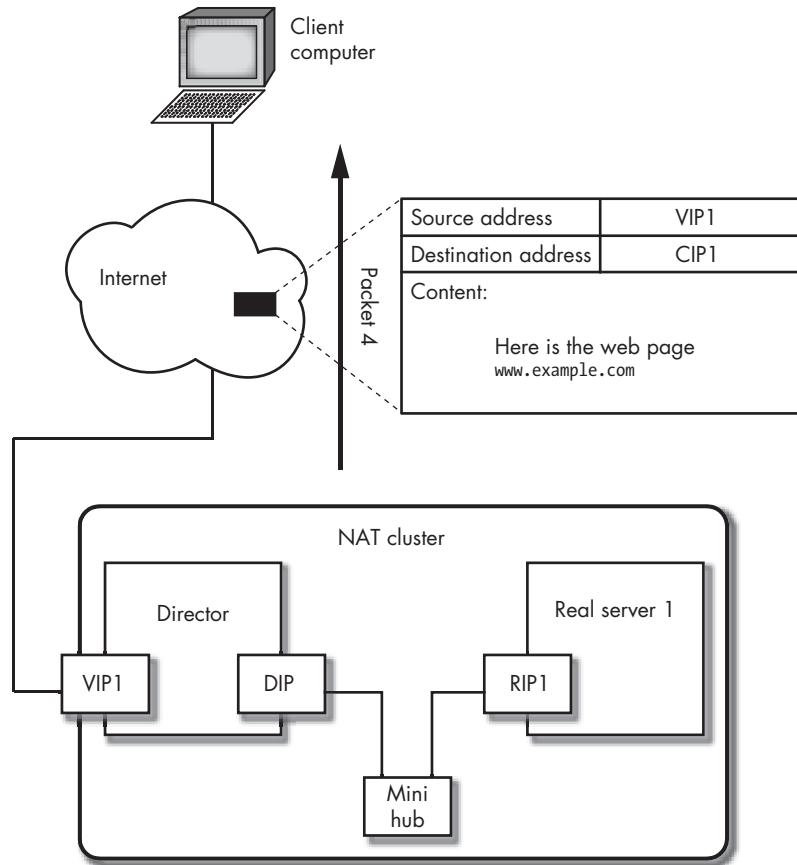


Figure 12-4: In packet 4 the Director forwards the reply packet to the client computer

Packet 4, shown in Figure 12-4, is the reply packet from the LVS-NAT cluster, and it contains a source address of VIP1 and a destination address of CIP1. The conversion of the source address from RIP1 to VIP1 is what gives LVS-NAT its name—it is the translation of the network address that lets the Director hide all of the cluster node's real IP addresses behind a single virtual IP address.

There are several things to notice about the process of this conversation:

- The Director is acting as a router for the cluster network.<sup>3</sup>
- The cluster nodes (real servers) use the DIP as their default gateway.<sup>4</sup>
- The Director must receive all of the inbound packets destined for the cluster.
- The Director must reply on behalf of the cluster nodes.
- Because the Director is masquerading the network addresses of the cluster nodes, the only way to test that the VIP addresses are properly replying to client requests is to use a client computer *outside* the cluster.

**NOTE** *The best method for testing an LVS cluster, regardless of the forwarding method you use, is to test the cluster's functionality from outside the cluster.*

### **Virtual IP Addresses on LVS-NAT Real Servers**

If you need to offer client computers access to different services on separate VIP addresses, cluster nodes in an LVS-NAT cluster can also use multiple RIP addresses (IP aliases or secondary IP addresses) to identify the VIP address the client computer used in the original request. As you can see in Figure 12-5, the Director can be configured to send packets to the cluster nodes (real servers) using multiple virtual RIP addresses. In this scenario, client computers will still only know the VIPs configured on the Director; the Director is responsible for sending the packets into the cluster through the proper virtual RIP address (using a load-balancing or scheduling method).

You might choose this scenario to load balance different types of users on different IP addresses. For example, accounting and reporting users might use one IP address, and customer service and simple transaction-based users would use another. You may also want to use IP-based virtual hosts for your web pages (see Appendix F for a detailed discussion of Apache virtual hosting).

Figure 12-5 shows the Director using three VIPs to receive network requests from client computers. The Director uses the LVS-NAT forwarding method to select one of the virtual RIP addresses on the real servers inside the cluster (any other LVS forwarding method could be used, too). In this configuration, the VIP addresses would normally be associated with the virtual RIP addresses, as shown in Figure 12-6.

---

<sup>3</sup> The Director can also act as a firewall for the cluster network, but this is outside the scope of this book. See the LVS HOWTO (<http://www.linuxvirtualserver.org>) for a discussion of the Antefacto patch.

<sup>4</sup> If the cluster nodes do not use the DIP as their default gateway, a routing table entry on the real server is required to send the client reply packets back out through the Director.

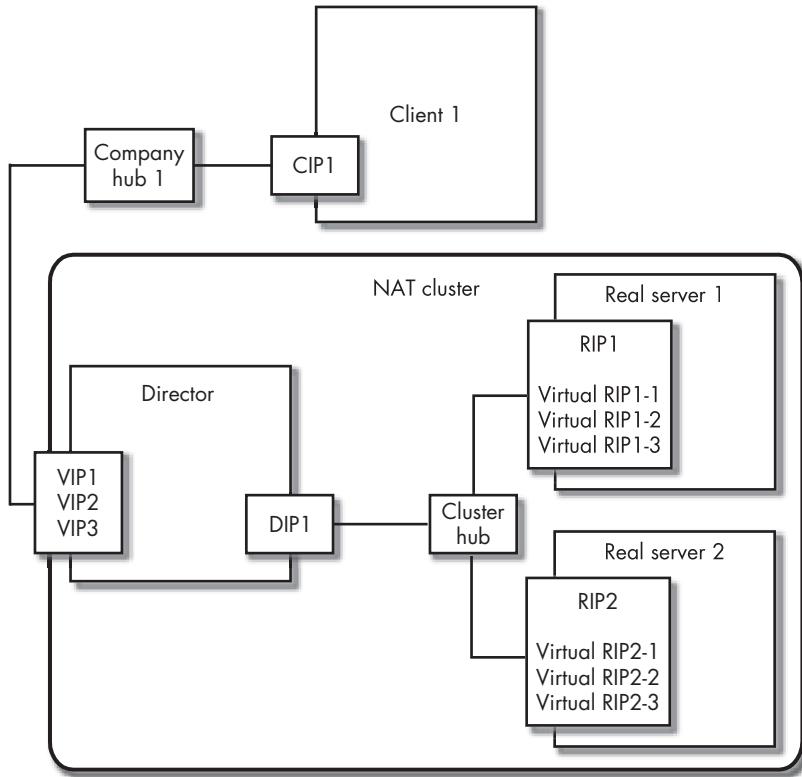


Figure 12-5: An LVS-NAT cluster with multiple VIPs

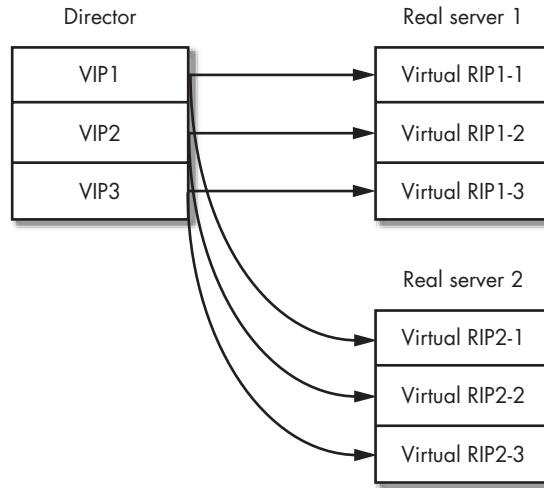


Figure 12-6: Multiple VIPs and their relationship with the multiple virtual RIs

Like the VIPs on the Director, the virtual RIP addresses can be created on the real servers using IP aliasing or secondary IP addresses. (IP aliasing and secondary IP addresses were introduced in Chapter 6.) A packet received on

the Director's VIP1 will be forwarded by the Director to either virtual RIP1-1 on real server 1 or virtual RIP2-1 on real server 2. The reply packet (from the real server) will be sent back through the Director and masqueraded so that the client computer will see the packet as coming from VIP1.

**NOTE** When building a cluster of web servers, you can use name-based virtual hosts instead of assigning multiple RIPv4 addresses to each real server. For a description of name-based virtual hosts, see Appendix F.

## Building an LVS-NAT Web Cluster

This recipe describes how to build an LVS-NAT web cluster consisting of a Director and a real server, using the Apache web server, as shown in Figure 12-7.

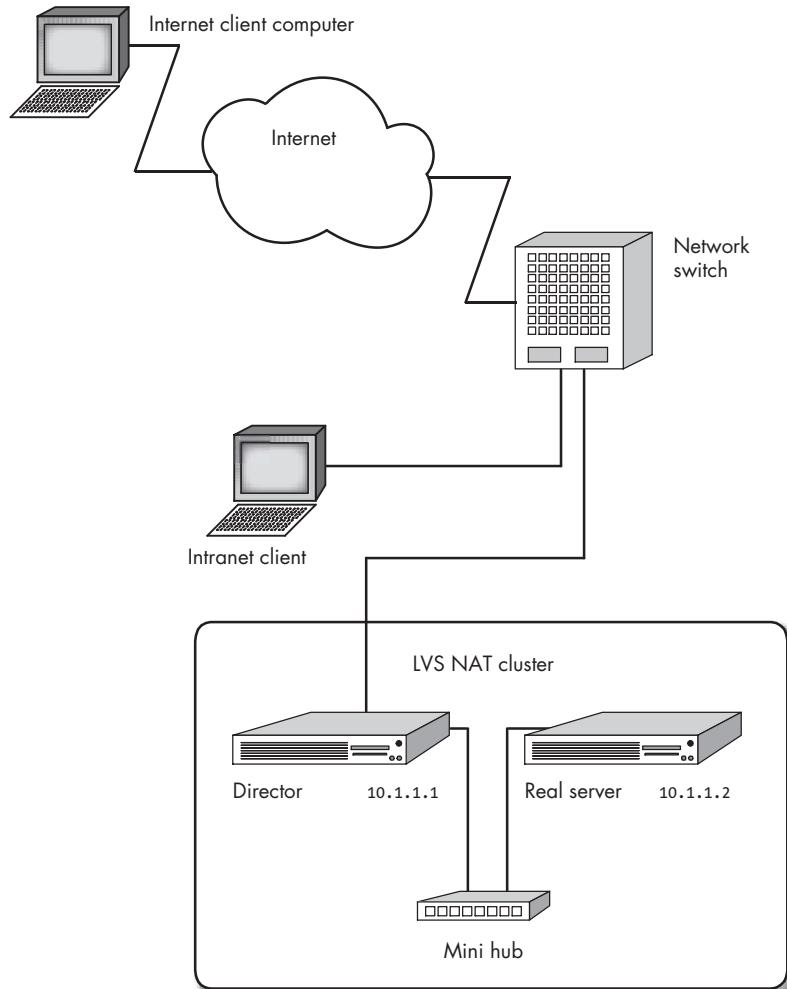


Figure 12-7: LVS-NAT web cluster

The LVS-NAT web cluster we'll build can be connected to the Internet, as shown in Figure 12-7. Client computers connected to the internal network (connected to the network switch shown in Figure 12-7) can also access the LVS-NAT cluster. Recall from our previous discussion, however, that the client computers must be *outside* the cluster (and must access the cluster services on the VIP).

**NOTE** *Figure 12-7 also shows a mini hub connecting the Director and the real server, but a network switch and a separate VLAN would work just as well.*

Before we build our first LVS-NAT cluster, we need to decide on an IP address range to use for our cluster network. Because these IP addresses do not need to be known by any computers outside the cluster network, the numbers you use are unimportant, but they should conform to RFC 1918.<sup>5</sup> We'll use this IP addressing scheme:

10.1.1.0	LVS-NAT cluster network (10.1.1.)
10.1.1.255	LVS-NAT cluster broadcast address
255.255.255.0	LVS-NAT cluster subnet mask

Assign the VIP address by picking a free IP address on your network. We'll use a fictitious VIP address of 209.100.100.100.

Let's continue with our recipe metaphor.

### ***Recipe for LVS-NAT***

List of ingredients:

2 servers running Linux<sup>6</sup>

1 client computer running an OS capable of supporting a web browser

3 network interface cards (2 for the Director and 1 for the real server)

1 mini hub with 2 twisted-pair cables (or 1 crossover cable)<sup>7</sup>

### ***Step 1: Install the Operating System***

When you install Linux on the two servers, be sure to configure the systems as web servers without any iptables (firewall or security) rules. The normal Red Hat installation process, for example, will automatically load Apache and create the /etc/httpd directory containing the Apache configuration files when you tell it that you would like your server to be a web server. Also, you do not need to load any X applications or a display manager for this recipe.

---

<sup>5</sup> RFC 1918 reserves the following IP address blocks for private intranets:

10.0.0.0 through 10.255.255

172.16.0.0 through 172.31.255.255

192.168.0.0 through 192.168.255.255

<sup>6</sup> Technically the real server (cluster node) can run any OS capable of displaying a web page.

<sup>7</sup> As mentioned previously, you can use a separate VLAN on your switch instead of a mini hub.

## Step 2: Configure and Start Apache on the Real Server

In this step you have to select which of the two servers will become the real server in your cluster. On the machine you select, make sure that the Apache daemon starts each time the system boots (see Chapter 1 for a description of the system boot process). You may have to use the `chkconfig` command to cause the `httpd` boot script to run at your default system runlevel. (See Chapter 1 for complete instructions.)

Next, you should modify the Apache configuration file on this system so it knows how to display the web content you will use to test your LVS-NAT cluster. (See Appendix F for a detailed discussion.)

After you've saved your changes to the `httpd.conf` file, make sure the error log and access log files you specified are available by creating empty files with these commands:

---

```
#touch /var/log/httpd/error_log
#touch /var/log/httpd/access_log
```

---

**NOTE** *The touch command creates an empty file.*

Also make sure your `DocumentRoot` exists and contains an `index.html` file—this is the default file the web server will display when you call up the URL. Use these commands:

---

```
#mkdir -p /www/htdocs
#echo "This is a test (from the real server)" > /www/htdocs/index.html
```

---

### Start Apache on the Real Server

You are now ready to start Apache using one of these commands:

---

```
#/etc/init.d/httpd start
```

---

or

---

```
#service httpd start
```

---

If the `HTTPD` daemon was already running, you will first need to enter one of these commands to stop it:

---

```
/etc/init.d/httpd stop
```

---

or

---

```
service httpd stop
```

---

The script should display this response: `OK`. If it doesn't, you probably have made an error in your configuration file (see Appendix F).

If it worked, confirm that Apache is running with this command:

---

```
#ps -elf | grep httpd
```

---

You should see several lines of output indicating that several HTTPd daemons are running. If not, check for errors with these commands:

---

```
#tail /var/log/messages  
#cat /var/log/httpd/*
```

---

If the HTTPd daemons are running, see if you can display your web page locally by using the Lynx command-line web browser:<sup>8</sup>

---

```
#lynx -dump 10.1.1.2
```

---

If everything is set up correctly, you should see the contents of your index.html file dumped to the screen by the Lynx web browser:

---

```
This is a test (from the real server)
```

---

### ***Step 3: Set the Default Route on the Real Server***

Real servers in an LVS-NAT cluster need to send all of their replies to client computers back through the Director. To accomplish this, we must set the default route for the real servers to the DIP. We can do so on Red Hat Linux by setting the GATEWAY variable in the /etc/sysconfig/network file.

Open the file in the vi text editor with this command:

---

```
#vi /etc/sysconfig/network
```

---

Add, or set, the GATEWAY variable to the Director's IP (DIP) address with an entry like this:

---

```
GATEWAY=10.1.1.1
```

---

We can enable this default route by rebooting (or re-running the /etc/init.d/network script) or by manually entering it into the route table with the following command:

---

```
#/sbin/route add default gw 10.1.1.1
```

---

This command might complain with the following error:

---

```
SIOCADDTR: File exists
```

---

---

<sup>8</sup> For this to work, you must have selected the option to install the text-based web browser when you loaded the operating system. If you didn't, you'll need to go get the Lynx program and install it.

If it does, this means the default route entry in the routing table has already been defined. You should reboot your system or remove the conflicting default gateway with the `route del` command.

### **Step 4: Install the LVS Software on the Director**

In this step you will make changes to the second server—the Director. (For the moment we are done making changes on the server you selected to be the real server.) The changes described here do not require you to reload your Linux distribution, but you will need to install a new kernel.

The Linux kernel is included on the CD-ROM that accompanies this book. This kernel contains the LVS software, but you'll need to compile and install this kernel with the LVS options enabled. You'll also need to install the ipvsadm utility to configure the Director. Older versions of the stock Linux kernel do not contain the LVS code, so you'll have to download the LVS patch and apply it to the kernel if you must use a kernel older than 2.4.23.

Copy the kernel source files included on the CD-ROM with this book (or download a kernel from <http://www.kernel.org>) and run the `make menuconfig` utility to enable the proper LVS and network options. Then use the instructions in Chapter 3 to compile and install this kernel on your system.

**NOTE** *You can avoid compiling the kernel by downloading a kernel that already contains the LVS patches from a distribution vendor. The Ultra Monkey project at <http://www.ultramonkey.org> also has patched versions of the Red Hat Linux kernel that can be downloaded as RPM files.*

Once you have rebooted your system on the new kernel,<sup>9</sup> you are ready to install the ipvsadm utility, which is also included on the CD-ROM. With the CD mounted, type these commands:

---

```
#cp /mnt/cdrom/chapter12/ipvsadm* /tmp
#cd /tmp
#tar xvf ipvsadm*
#cd ipvsadm-
#make
#make install
```

---

If this command completes without errors, you should now have the ipvsadm utility installed in the `/sbin` directory. If it does not work, make sure the `/usr/src/linux` directory (or symbolic link) contains the source code for the kernel with the LVS patches applied to it.

---

<sup>9</sup> As the system boots, watch for IPVS messages (if you compiled the IPVS schedulers into the kernel and not as modules) for each of the scheduling methods you compiled for your kernel.

## Step 5: Configure LVS on the Director

Now we need to tell the Director how to forward packets to the cluster node (the real servers) using the ipvsadm utility we compiled and installed in the previous step.

One way to do this is to use the configure script included with the LVS distribution. (See the LVS HOWTO at <http://www.linuxvirtualserver.org> for a description of how to use this method to configure an LVS cluster.) In this chapter, however, we will use our own custom script to create our LVS cluster so that we can learn more about the ipvsadm utility.

**NOTE** *We will abandon this method in Chapter 15 when we use the ldirectord daemon to create an LVS Director configuration—ldirectord will enter the ipvsadm commands to create the IPVS table automatically.*

Create an /etc/init.d/lvs script that looks like this (this script is included in the chapter12 subdirectory on the CD-ROM that accompanies this book):

---

```
#!/bin/bash
#
# LVS script
#
# chkconfig: 2345 99 90
# description: LVS sample script
#
case "$1" in
  start)
    # Bring up the VIP (Normally this should be under Heartbeat's
    # control.)
    /sbin/ifconfig eth0:1 209.100.100.3 netmask 255.255.255.0 up

  # Since this is the Director we must be
  # able to forward packets.10
  echo 1 > /proc/sys/net/ipv4/ip_forward

  # Clear all iptables rules.
  /sbin/iptables -F

  # Reset iptables counters.
  /sbin/iptables -Z

  # Clear all ipvsadm rules/services.
  /sbin/ipvsadm -C

  # Add an IP virtual service for VIP 209.100.100.3 port 80
```

---

<sup>10</sup> Many distributions also include the sysctl command for modifying the /proc pseudo filesystem. On Red Hat systems, for example, this kernel capability is controlled by sysctl from the /etc/rc.d/init.d/network script, as specified by a variable in the /etc/sysctl.conf file. This setting will override your custom LVS script if the network script runs after your custom LVS script. See Chapter 1 for more information about the boot order of init scripts.

```

/sbin/ipvsadm -A -t 209.100.100.3:80 -s rr

# Now direct packets for this VIP to
# to the real server IP (RIP) inside the cluster
/sbin/ipvsadm -a -t 209.100.100.3:80 -r 10.1.1.2 -m
;;

stop)
# Stop forwarding packets
echo 0 > /proc/sys/net/ipv4/ip_forward

# Reset ipvsadm
/sbin/ipvsadm -C

# Bring down the VIP interface
ifconfig eth0:1 down
;;
*)
echo "Usage: $0 {start|stop}"
;;
esac

```

---

**NOTE** *If you are running on a version of the kernel prior to version 2.4, you will also need to configure the masquerade for the reply packets that pass back through the Director. Do this with the ipchains utility by using a command such as the following:*

---

```
/sbin/ipchains -A forward -j MASQ -s 10.1.1.0/24 -d 0.0.0.0/0
```

---

*Starting with kernel 2.4, however, you do not need to enter this command because LVS does not use the kernel's NAT code. The 2.0 series kernels also needed to use the ipfwadm utility, which you may see mentioned on the LVS website, but this is no longer required to build an LVS cluster.*

The two most important lines in the preceding script are the lines that create the IP virtual server:

---

```
/sbin/ipvsadm -A -t 209.100.100.3:80 -s rr
/sbin/ipvsadm -a -t 209.100.100.3:80 -r 10.1.1.2 -m
```

---

The first line specifies the VIP address and the scheduling method (-s rr). The choices for scheduling methods (which were described in the previous chapter) are as follows:

ipvsadm Argument	Scheduling Method
-s rr	Round-robin
-s wrr	Weighted round-robin
-s lc	Least-connection
-s wlc	Weighted least-connection

ipvsadm Argument	Scheduling Method
-s lb1c	Locality-based least-connection
-s lb1cr	Locality-based least-connection with replication
-s dh	Destination hashing
-s sh	Source hashing
-s sed	Shortest expected delay
-s nq	Never queue

In this recipe, we will use the round-robin scheduling method. In production, however, you should use a weighted, dynamic scheduling method (see Chapter 11 for explanations of the various methods).

The second ipvsadm line listed above associates the real server's RIP (-r 10.1.1.2) with the VIP (or virtual server), and it specifies the forwarding method (-m). Each ipvsadm entry for the same virtual server can use a different forwarding method, but normally only one method is used. The choices are as follows:

ipvsadm Argument	Forwarding Method
-g	LVS-DR
-i	LVS-TUN
-m	LVS-NAT

In this recipe, we are building an LVS-NAT cluster, so we will use the -m option.<sup>11</sup>

When you have finished entering this script or modifying the copy on the CD-ROM, run it by typing this command:

---

```
#/etc/init.d/lvs start
```

---

If the VIP address has been added to the Director (which you can check by using the ifconfig command), log on to the real server and try to ping this VIP address. You should also be able to ping the VIP address from a client computer (so long as there are no intervening firewalls that block ICMP packets).

**NOTE** *ICMP ping packets sent to the VIP will be handled by the Director (these packets are not forwarded to a real server). However, the Director will try to forward ICMP packets relating to a connection to the relevant real server.*

To see the IP Virtual Server table (the table we have just created that tells the Director how to forward incoming requests for services to real servers inside the cluster), enter the following command on the server you have made into a Director:

---

```
#/sbin/ipvsadm -L -n
```

---

<sup>11</sup> Note that you can't simply change this option to -g to build an LVS-DR cluster.

This command should show you the IPVS table:

---

IP Virtual Server version 1.0.10 (size=4096)					
Prot	LocalAddress:Port	Scheduler	Flags	Forward	Weight
	-> RemoteAddress:Port			ActiveConn	InActConn
TCP	209.100.100.3:80	rr			
	-> 10.1.1.2:80		Masq	1	0
					0

---

This report wraps both the column headers and each line of output onto the next line. We have only one IP virtual server pointing to one real server, so our one (wrapped) line of output is:

---

TCP	209.100.100.3:80	rr			
	-> 10.1.1.2:80		Masq	1	0

---

This line says that our IP virtual server is using the TCP protocol for VIP address 209.100.100.3 on port 80. Packets are forwarded (->) to RIP address 10.1.1.2 on port 80, and our forwarding method is masquerading (Masq), which is another name for the Network Address Translation, or LVS-NAT. The LVS forwarding method reported in this field will be one of the following:

---

Report Output	LVS Forwarding Method
Masq	LVS-NAT
Route	LVS-DR
Tunnel	LVS-TUN

---

### **Step 6: Test the Cluster Configuration**

The next step is to test the cluster configuration. In this step, <DR> will precede the commands that should be entered on the Director, and <RS> will precede the commands that should be entered on the real server.

The first thing to do is make sure the network interface cards are configured properly and are receiving packets. Enter the following command (on the Director):

---

```
<DR>#ifconfig -a
```

---

The important thing to look for in the output of this command is whether or not the interface is **UP**, and how many packets have been transmitted (TX) and received (RX). In the following sample output I've marked this information in bold:

---

```
eth0      Link encap:Ethernet HWaddr 00:10:5A:16:99:8A
          inet addr:209.100.100.2 Bcast:209.100.100.255 Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
```

---

```
          RX packets:481 errors:0 dropped:0 overruns:0 frame:0
          TX packets:374 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          Interrupt:5 Base address:0x220

eth0:1      Link encap:Ethernet HWaddr 00:10:5A:16:99:8A
            inet addr:209.100.100.3 Bcast:209.100.100.255 Mask:255.255.255.0
            UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
            Interrupt:5 Base address:0x220

eth1      Link encap:Ethernet HWaddr 00:80:5F:0E:AB:AB
            inet addr:10.1.1.1 Bcast:10.1.1.255 Mask:255.255.255.0
            UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
            RX packets:210 errors:0 dropped:0 overruns:0 frame:0
            TX packets:208 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0 txqueuelen:100
            Interrupt:11 Base address:0x1400
```

---

If you do not see the word **UP** displayed in the output of your `ifconfig` command for all of your network interface cards and network interface aliases (or IP aliases), as shown in the bold sections in this example, then the software driver for the missing network interface card is not configured properly (or the card is not installed properly). See Appendix C for more information on working with network interface cards.

**NOTE** *If you use secondary IP addresses instead of IP aliases, use the `ip addr sh` command to check the status of the secondary IP addresses instead of the `ifconfig -a` command. You can then examine the output of this command for the same information (the report returned by the `ip addr sh` command is a slightly modified version of the sample output I've just given).*

If the interface is **UP** but no packets have been transmitted (TX) or received (RX) on the card, you may have network cabling problems.

Once you have a working network configuration, test the communication from the real server to the Director by pinging the Director's DIP and then VIP addresses from the real server. To continue the example, you would enter the following commands on the real server:

---

```
<RS>#ping 10.1.1.1
<RS>#ping 209.100.100.3
```

---

The first of these commands pings the Director's cluster IP address (DIP), and the second pings the Director's virtual IP address (VIP). Both of these commands should report that packets can successfully travel from the real server through the cluster network to the Director and that the Director successfully recognizes both its DIP and VIP addresses. Once you are sure this basic cluster network's IP communication is working properly, use a client computer to ping the VIP address from outside the cluster.

When you have confirmed that all of these tests work, you are ready to test the service from the real server. Use the Lynx program to send an HTTP request to the locally running Apache server on the real server:

---

```
<RS>#lynx -dump 10.1.1.2
```

---

If this does not return the test web page message, check to make sure HTTPd is running by issuing the following commands (on the real server):

---

```
<RS>#ps -elf | grep httpd
<RS>#netstat -apn | grep httpd
```

---

The first command should return several lines of output from the process table, indicating that several HTTPd daemons are running. The second command should show that these daemons are listening on the proper network ports (or at least on port 80). If either of these commands does not produce any output, you need to check your Apache configuration on the real server before continuing.

If the HTTPd daemons are running, you are ready to access the real server's HTTPd server from the Director. You can do so with the following command (on the Director):

---

```
<DR>#lynx -dump 10.1.1.2
```

---

This command, which specifies the real server's IP address (RIP), should display the test web page from the real server.

If all of these commands work properly, use the following command to watch your LVS connection table on the Director:

---

```
<DR>#watch ipvsadm -Ln
```

---

At first, this report should indicate that no active connections have been made through the Director to the real server, as shown here:

---

IP Virtual Server version 1.0.10 (size=4096)							
Prot	LocalAddress:Port	Scheduler	Flags	Forward	Weight	ActiveConn	InActConn
TCP	209.100.100.3:80	rr					
	-> 10.1.1.2:80			Masq	1	0	0

---

Leave this command running on the Director (the `watch` command will automatically update the output every two seconds), and from a client computer use a web browser to display the URL that is the Director's virtual IP address (VIP):

---

```
http://209.100.100.3/
```

---

If you see the test web page, you have successfully built your first LVS cluster.

If you look quickly back at the console on the Director, now, you should see an active connection (ActiveConn) in the LVS connection tracking table:

---

IP Virtual Server version 1.0.10 (size=4096)				
Prot	LocalAddress:Port	Scheduler	Flags	
-> RemoteAddress:Port Forward Weight ActiveConn InActConn				
TCP	209.100.100.3:80	rr		
	-> 10.1.1.2:80		Masq	1 1 0

---

In this report, the number 1 now appears in the ActiveConn column. If you wait a few seconds, and you don't make any further client connections to this VIP address, you should see the connection go from active status (ActiveConn) to inactive status (InActConn):

---

IP Virtual Server version 1.0.10 (size=4096)				
Prot	LocalAddress:Port	Scheduler	Flags	
-> RemoteAddress:Port Forward Weight ActiveConn InActConn				
TCP	209.100.100.3:80	rr		
	-> 10.1.1.2:80		Masq	1 0 1

---

In this report, the number 1 now appears in the InActConn column, and then it will finally drop out of the LVS connection tracking table altogether:

---

IP Virtual Server version 1.0.10 (size=4096)				
Prot	LocalAddress:Port	Scheduler	Flags	
-> RemoteAddress:Port Forward Weight ActiveConn InActConn				
TCP	209.100.100.3:80	rr		
	-> 10.1.1.2:80		Masq	1 0 0

---

The ActiveConn and InActConn columns are now both 0.

## LocalNode: Using the Director as a Real Server

When you build an enterprise-class cluster, you will need to decide whether you want to dedicate two Linux servers to be the primary and backup LVS Directors. For the moment, however, you can test to make sure the load-balancing function is working properly (regardless of what your final design configuration will be) by configuring and installing Apache on the Director (as described in the “Step 2: Configure and Start Apache on the Real Server” section of this chapter) and then configuring your Director to send client computer requests to this locally running copy of Apache as if the Director were also a node inside the cluster. This configuration is called *LocalNode* or *LocalNode mode*.

**NOTE** A limitation of using a *LocalNode* configuration is that the Director cannot remap port numbers—whatever port number the client computer used in its request to access the service must also be used on the Director to service the request. The LVS HOWTO describes a workaround for this current limitation in LVS that uses *iptables/ipchains* redirection.

Assuming you have Apache up and running on the Director, you can modify the */etc/init.d/lvs* script on the Director so two cluster nodes are available to client computers (the Director and the real server) as follows:

---

```
# Add an IP virtual service for VIP 209.100.100.3 port 80
/sbin/ipvsadm -A -t 209.100.100.3:80 -s rr

# Now direct packets for this VIP to
# the real server IP (RIP) inside the cluster
/sbin/ipvsadm -a -t 209.100.100.3:80 -r 10.1.1.2 -m
# And send requests to the locally running Apache
# server on the Director.
/sbin/ipvsadm -a -t 209.100.100.3:80 -r 127.0.0.1 -m
```

---

The last two uncommented lines cause the Director to send packets destined for the 209.100.100.3 VIP address to the RIP (10.1.1.2) and to the locally running daemons on the Director (127.0.0.1).

Use *ipvsadm* to display the newly modified IP virtual server table:

---

```
<DR>#ipvsadm -Ln

IP Virtual Server version 1.0.10 (size=4096)
Prot LocalAddress:Port Scheduler Flags
    -> RemoteAddress:Port           Forward Weight ActiveConn InActConn
TCP  209.100.100.3:80  rr
    -> 127.0.0.1:80                 Masq    1      0      0
    -> 10.1.1.2:80                 Masq    1      0      0
```

---

Now we have two servers capable of responding to requests received on 209.100.100.3—the cluster node and the Director.

Use the Lynx program to make sure you have created a default web page on your Director that is different from the default web page used on your real server. On the Director type this command:

---

```
<DR>#lynx -dump 127.0.0.1
```

---

This should produce output that shows the default web page you have created on your Director.

**NOTE** For testing purposes you may want to create test pages that display the name of the real server that is displaying the page. See Appendix F for more info on how to configure Apache and how to use virtual hosts.

Now you can watch the IP virtual server round-robin scheduling method in action. From a client computer (outside the cluster) call up the web page the cluster is offering by using the virtual IP address in the URL:

---

`http://209.100.100.3/`

---

This URL specifies the Director's virtual IP address (VIP). Watch for this connection on the Director and make sure it moves from the ActiveConn to the InActConn column and then expires out of the IP virtual server table. Then click the **Refresh** button<sup>12</sup> on the client computer's web browser, and the web page should change to the default web page used on the real server.

## In Conclusion

In this chapter we looked at how client computers communicate with real servers inside an LVS-NAT cluster, and we built a simple LVS-NAT web cluster. As I mentioned in Chapter 10, the LVS-NAT cluster will probably be the first type of LVS cluster that you build because it is the easiest to get up and working. If you've followed along in this chapter you now have a working two-node LVS-NAT cluster. But don't stop there. Continue on to the next chapter to learn how to improve the LVS-NAT configuration and build an LVS-DR cluster.

---

<sup>12</sup>You may need to hold down the SHIFT key while clicking the **Refresh** button to avoid accessing the web page from the cached copy the web browser has stored on the hard drive. Also, be aware of the danger of accessing the same web page stored in a cache proxy server (such as Squid) if you have a proxy server configured in your web browser.



# 13

## THE LVS-DR CLUSTER



The Linux Virtual Server Direct Routing (LVS-DR) cluster is made possible by configuring all nodes in the cluster and the Director with the same VIP address; despite having this common address, though, client computers will only send their packets to the Director.

The Director can, therefore, balance the incoming workload from the client computers by using one of the LVS scheduling methods we looked at in Chapter 12.

The LVS-DR cluster configuration that we use in this chapter assumes that the Director is a computer dedicated to this task. In Chapter 14 we'll take a closer look at what is going on inside this computer, and in Chapter 15 we'll see how the load-balancing resource<sup>1</sup> can be placed on a real server and made highly available using the Heartbeat package.

---

<sup>1</sup> Recall from our discussion of Heartbeat in Chapter 6 that a service, along with its associated IP address, is known as a *resource*. Thus, the virtual services offered by a Linux Virtual Server and their associated VIPs can also be called *resources* in high-availability terminology.

But before we build an enterprise-class, highly available LVS-DR cluster (in Chapter 15), let's examine how the LVS-DR forwarding method works in more detail.

## How Client Computers Access LVS-DR Cluster Services

Let's examine the TCP network communication that takes place between a client computer and the cluster. As with the LVS-NAT cluster network communication described in Chapter 12, the LVS-DR TCP communication starts when the client computer sends a request for a service running on the cluster, as shown in Figure 13-1.<sup>2</sup>

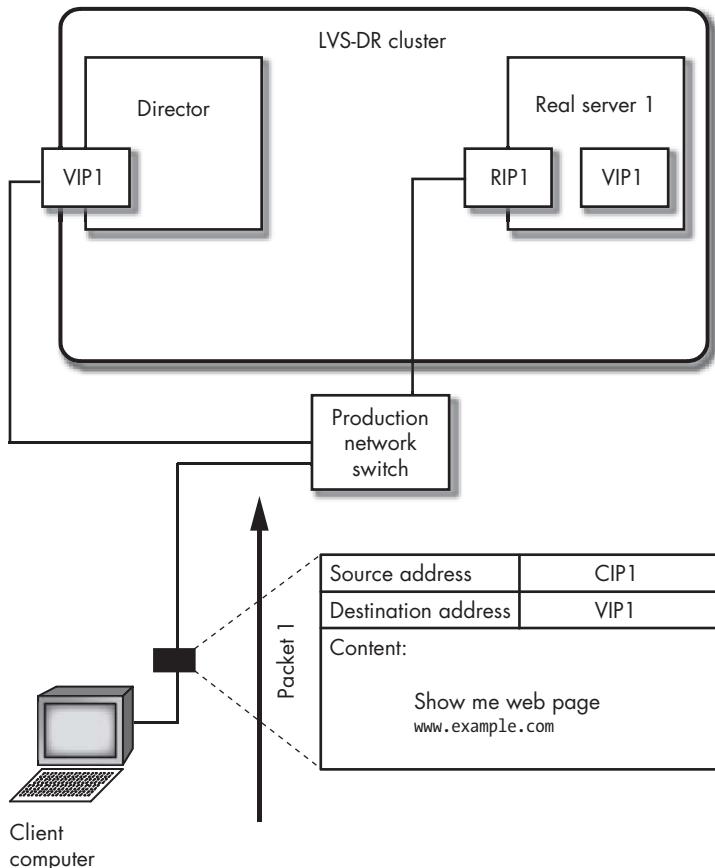


Figure 13-1: In packet 1 the client sends a request to the LVS-DR cluster

The first packet, shown in Figure 13-1, is sent from the client computer to the VIP address. Its data payload is an HTTP request for a web page.

<sup>2</sup> We are ignoring the lower-level TCP connection request (the TCP handshake) in this discussion for the sake of simplicity.

**NOTE** An LVS-DR cluster, like an LVS-NAT cluster, can use multiple virtual IP (VIP) addresses, so we'll number them for the sake of clarity.

Before we focus on the packet exchange, there are a couple of things to note about Figure 13-1. The first is that the network interface card (NIC) the Director uses for network communication (the box labeled VIP1 connected to the Director in Figure 13-1) is connected to the same physical network that is used by the cluster node and the client computer. The VIP the RIP and the CIP, in other words, are all on the same physical network (the same network segment or VLAN).

**NOTE** You can have multiple NICs on the Director to connect the Director to multiple VLANs.

The second is that VIP1 is shown in two places in Figure 13-1: it is in a box representing the NIC that connects the Director to the network, and it is in a box that is inside of real server 1. The box inside of real server 1 represents an IP address that has been placed on the loopback device on real server 1. (Recall that a *loopback device* is a logical network device used by all networked computers to deliver packets locally.) Network packets<sup>3</sup> that are routed inside the kernel on real server 1 with a destination address of VIP1 will be sent to the loopback device on real server 1—in other words, any packets found inside the kernel on real server 1 with a destination address of VIP1 will be delivered to the daemons running locally on real server 1. (We'll see how packets that have a destination address of VIP1 end up inside the kernel on real server 1 shortly.)

Now let's look at the packet depicted in Figure 13-1. This packet was created by the client computer and sent to the Director. A technical detail not shown in the figure is the lower-level destination MAC address inside this packet. It is set to the MAC address of the Director's NIC that has VIP1 associated with it, and the client computer discovered this MAC address using the Address Resolution Protocol (ARP).

An ARP broadcast from the client computer asked, "Who owns VIP1?" and the Director replied to the broadcast using its MAC address and said that it was the owner. The client computer then constructed the first packet of the network conversation and inserted the proper destination MAC address to send the packet to the Director. (We'll examine a broadcast ARP request and see how they can create problems in an LVS-DR cluster environment later in this chapter.)

**NOTE** When the cluster is connected to the Internet, and the client computer is connected to the cluster over the Internet, the client computer will not send an ARP broadcast to locate the MAC address of the VIP. Instead, when the client computer wants to connect to the

---

<sup>3</sup>When the kernel holds a packet in memory it places the kernel into an area of memory that is references with a pointer called a socket buffer or `sk_buff`, so, to be completely accurate in this discussion I should use the term `sk_buff` instead of packet every time I mention a packet inside the director.

*cluster, it sends packet 1 over the Internet, and when the packet arrives at the router that connects the cluster to the Internet, the router sends the ARP broadcast to find the correct MAC address to use.*

When packet 1 arrives at the Director, the Director forwards the packet to the real server, leaving the source and destination addresses unchanged, as shown in Figure 13-2. Only the MAC address is changed from the Director's MAC address to the real server's (RIP) MAC address.

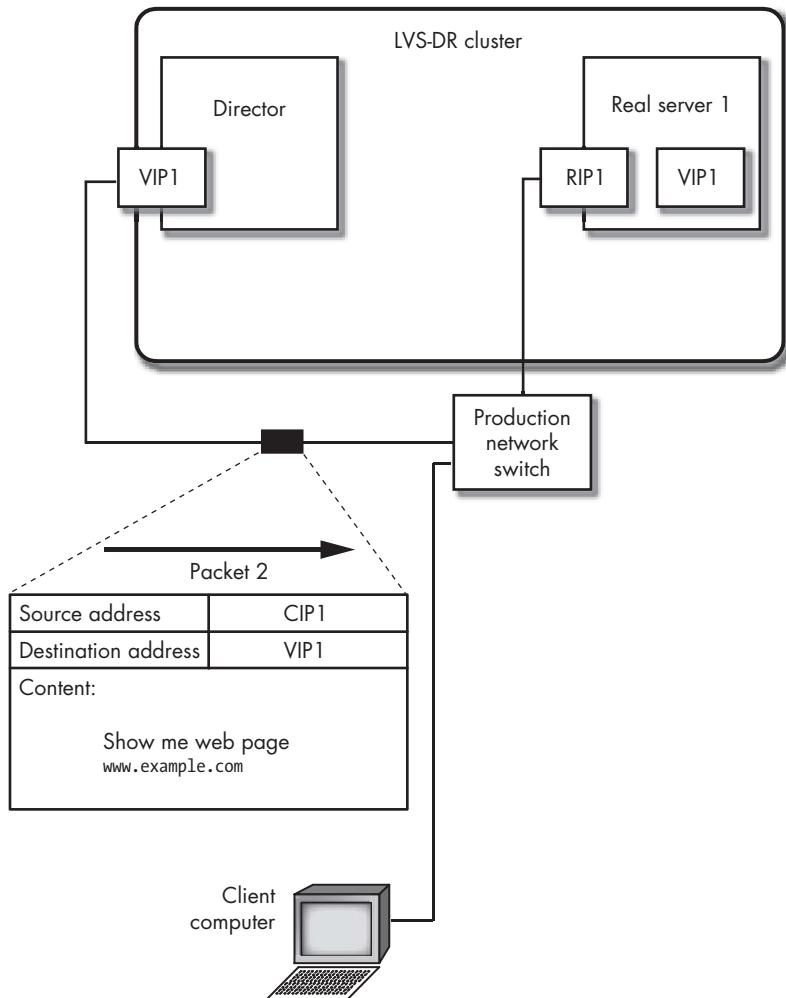


Figure 13-2: In packet 2 the Director forwards the client computer's request to a cluster node

Notice in Figure 13-2 that the source and destination IP address have not changed in packet 2: CIP1 is still the source address, and VIP1 is still the destination address. The Director, however, has changed the destination MAC address of the packet to that of the NIC on real server 1 in order to send the packet into the kernel on real server 1 (though the MAC addresses

aren't shown in the figure). When the packet reaches real server 1, the packet is routed to the loopback device, because that's where the routing table inside the kernel on real server 1 is configured to send it. (In Figure 13-2, the box inside of real server 1 with the VIP1 address in it depicts the VIP1 address on the loopback device.) The packet is then received by a daemon running locally on real server 1 listening on VIP1, and that daemon knows what to do with the packet—the daemon is the Apache HTTPd web server in this case.

The HTTPd daemon then prepares a reply packet and sends it back out through the RIP1 interface with the source address set to VIP1, as shown in Figure 13-3.

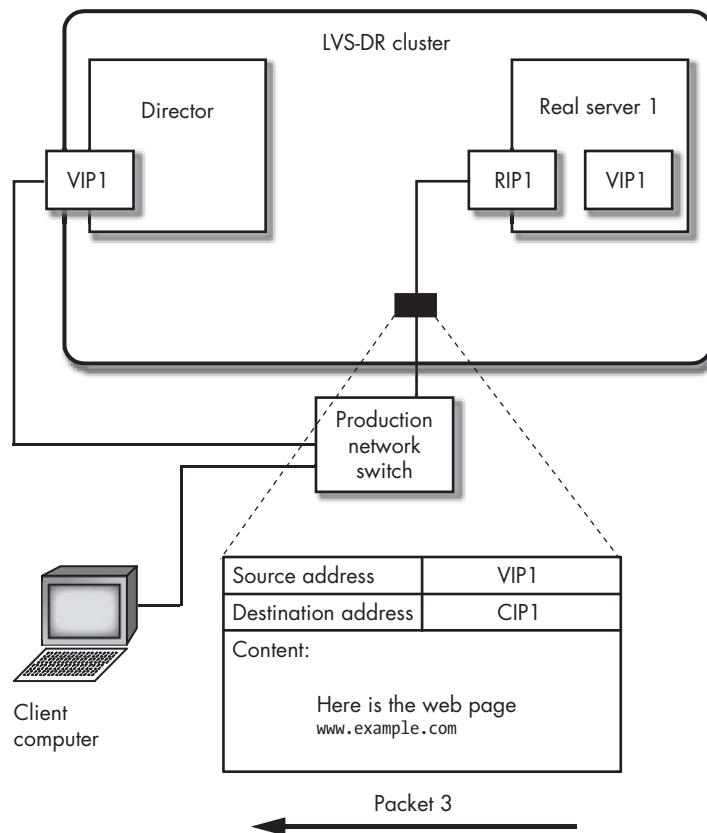


Figure 13-3: In packet 3 the cluster node sends a reply back through the Director

The packet shown in Figure 13-3 does not go back through the Director, because the real servers do not use the Director as their default gateway in an LVS-DR cluster. Packet 3 is sent directly back to the client computer (hence the name *direct routing*). Also notice that the source address is VIP1, which real server 1 took from the destination address it found in the inbound packet (packet 2).

Notice the following points about this exchange of packets:

- The Director must receive all the inbound packets destined for the cluster.
- The Director only receives inbound cluster communications (requests for services from client computers).
- Real servers, the Director, and client computers can all share the same network segment.
- The real servers use the router on the production network as their default gateway (unless you are using the LVS martian patch on your Director). If client computers will always be on the same network segment as the cluster nodes, you do not need to configure a default gateway for the real servers.<sup>4</sup>

## ARP Broadcasts and the LVS-DR Cluster

As we've just seen, placing VIP addresses on the loopback (lo) device on each cluster node allows the cluster nodes in an LVS-DR cluster to accept packets that are destined for the VIP address. However, this has one dangerous side effect: the real servers inside the cluster will try to reply to ARP broadcasts from client computers that are looking for the VIP. Unless special precautions are taken, the real servers will claim to own the VIP address, and client computers will send their packets directly to real servers, thus circumventing the cluster load-balancing method and destroying the integrity of network communication with the Director (where packets that use the VIP as their destination address are supposed to go).

To understand this problem (called "The ARP Problem" in the LVS-HOWTO), let's look at how a client computer uses the VIP address to find the correct MAC address by using ARP.

## Client Computers and ARP Broadcasts

Figure 13-4 shows a client computer sending an ARP broadcast to an LVS-DR cluster. Notice that because the Director and the cluster node (real server 1) are connected to the same network, they will both receive the ARP broadcast asking, "Who owns VIP1?"

In Figure 13-4, gray arrows represent the path taken by an ARP broadcast sent from the client computer. The ARP broadcast packet is sent to all nodes connected to the local network (the VLAN or physical network segment), so a gray arrow is shown on the physical wires that connect the Director and real server 1 to the network switch. This is normal network behavior.

However, we want real server 1 to ignore this ARP request and only the LVS-DR Director to respond to it, as shown in Figure 13-5. In the figure, a gray arrow depicts the path of the ARP reply. It should only come from the Director and not real server 1.

---

<sup>4</sup>This, however, would be an unusual configuration, because real servers will likely need to access both an email server and a DNS server residing on a different network segment.

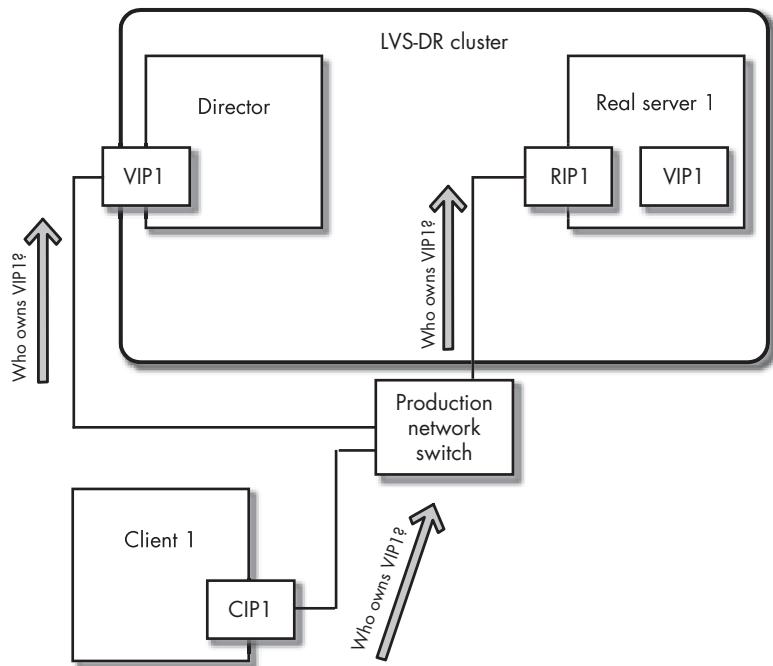


Figure 13-4: An ARP broadcast to an LVS-DR cluster

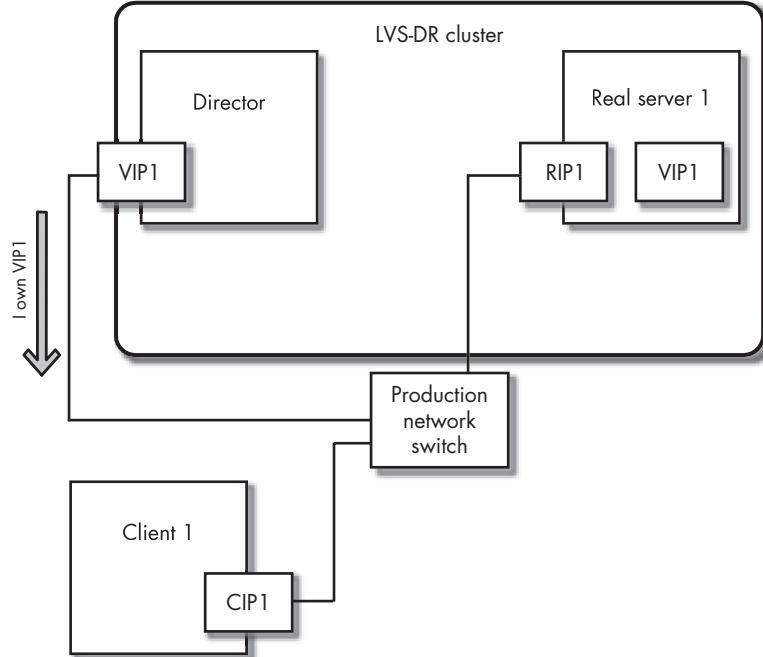


Figure 13-5: An ARP response from the LVS-DR Director

To prevent real servers from replying to ARP broadcasts for the LVS-DR cluster VIP, we need to hide the loopback interface on all of the real servers. Several techniques are available to accomplish this, and they are described in the LVS-HOWTO.

**NOTE** *Starting with Kernel version 2.4.26, the stock Linux kernel contains the code necessary to prevent real servers from replying to ARP broadcasts. This is discussed in Chapter 15.*

## In Conclusion

We've examined the LVS-DR forwarding method in detail in this chapter, and we examined a sample LVS-DR network conversation between a client computer and a cluster node. We've also briefly described a potential problem with ARP broadcasts when you build an LVS-DR cluster.

In Chapter 15, we will see how to build a high-availability, enterprise-class LVS-DR cluster. Before we do so, though, Chapter 14 will look inside the load balancer.

# 14

## THE LOAD BALANCER



In this chapter, we take a closer look at what happens when the Director receives a packet destined for a real server (a cluster node). This will lead us into a discussion of LVS persistence—the technique LVS uses to assign the same client computer to a particular real server. We'll also describe how packets can be selected with the iptables utility for processing by LVS using a technique called *packet marking*.

### LVS and Netfilter

In Figure 14-1, the five Netfilter hooks that were introduced in Chapter 2 are shown. Superimposed on top of these hooks is a series of small black boxes representing packets passing through the kernel. The kernel places each packet it receives into a memory structure called a *socket buffer*, or *sk\_buff* for short. Each of the little black boxes in Figure 14-1 is thus representing an *sk\_buff* inside the kernel, but in this discussion we'll continue to call them

*packets.* The gray arrows in the figure represent the path that all incoming LVS packets (packets from client computers) take as they pass through the Director on their way to a real server (cluster node).

Let's begin by looking at the five Netfilter hooks introduced in Chapter 2. Figure 14-1 shows these five hooks in the kernel.

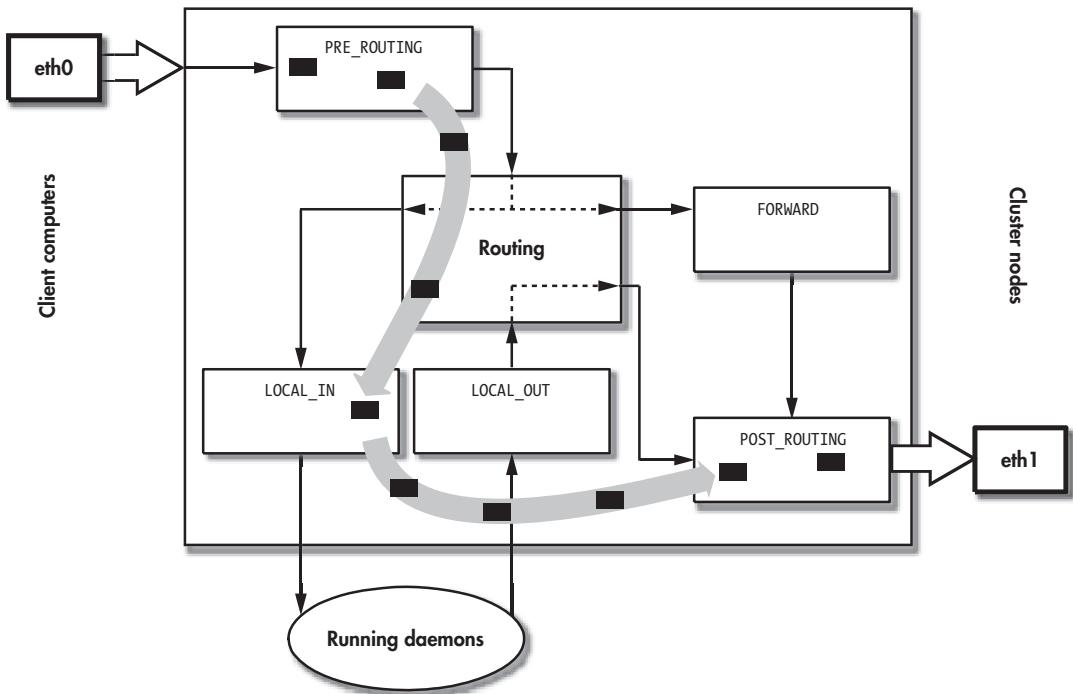


Figure 14-1: Incoming packets inside the Director

Notice in Figure 14-1 that incoming LVS packets hit only three of the five Netfilter hooks: `PRE_ROUTING`, `LOCAL_IN`, and `POST_ROUTING`.<sup>1</sup> Later in this chapter, we'll discuss the significance of these Netfilter hooks as they relate to your ability to control the fate of a packet on the Director. For the moment, we want to focus on the path that incoming packets take as they pass through the Director on their way to a cluster node, as represented by the two gray arrows in Figure 14-1.

The first gray arrow in Figure 14-1 represents a packet passing from the `PRE_ROUTING` hook to the `LOCAL_IN` hook inside the kernel. Every packet received by the Director that is destined for a cluster service regardless of the LVS forwarding method you've chosen<sup>2</sup> must pass from the `PRE_ROUTING` hook to the `LOCAL_IN` hook inside the kernel on the Director. The packet routing rules inside the kernel on the Director, in other words, must send all packets for cluster services to the `LOCAL_IN` hook. This is easy to accomplish when you

<sup>1</sup> LVS also uses the `IP_FORWARD` hook to recognize LVS-NAT reply packets (packets sent from the cluster nodes to client computers).

<sup>2</sup> LVS forwarding methods were introduced in Chapter 11.

build a Director because all packets for cluster services that arrive on the Director will have a destination address of the virtual IP (VIP) address. Recall from our discussion of the LVS forwarding methods in the last three chapters that the VIP is an IP alias or secondary IP address that is owned by the Director. Because the VIP is a local IP address owned by the Director, the routing table inside the kernel on the Director will always try to deliver packets locally. Packets received by the Director that have the VIP as a destination address will therefore always hit the `LOCAL_IN` hook.

The second gray arrow in Figure 14-1 represents the path taken by the incoming packets after the kernel has recognized that a packet is a request for a virtual service. When a packet hits the `LOCAL_IN` hook, the LVS software running inside the kernel knows the packet is a request for a cluster service (called a *virtual service* in LVS terminology) because the packet is destined for a VIP address. When you build your cluster, you use the `ipvsadm` utility to add virtual service VIP addresses to the kernel so LVS can recognize incoming packets sent to the VIP in the `LOCAL_IN` hook. If you had not already added the VIP to the LVS virtual service table (also known as the IPVS table), the packets destined for the VIP address would be delivered to the locally running daemons on the Director. But because LVS knows the VIP address,<sup>3</sup> it can check each packet as it hits the `LOCAL_IN` hook to decide if the packet is a request for a cluster service. LVS can then alter the fate of the packet before it reaches the locally running daemons on the Director. LVS does this by telling the kernel that it should not deliver a packet destined for the VIP address locally, but should send the packet to a cluster node instead. This causes the packet to be sent to the `POST_ROUTING` hook as depicted by the second gray arrow in Figure 14-1. The packets are sent out the NIC connected to the D/RIP network.<sup>4</sup>

The power of the LVS Director to alter the fate of packets is therefore made possible by the Netfilter hooks and the IPVS table you construct with the `ipvsadm` utility. The Director's ability to alter the fate of network packets makes it possible to distribute incoming requests for cluster services across multiple real servers (cluster nodes), but to do this, the Director must keep track of which real servers have been assigned to each client computer so the client computer will always talk to the same real server.<sup>5</sup> The Director does this by maintaining a table in memory called the *connection tracking table*.

**NOTE** *As we'll see later in this chapter, there is a difference between making sure all requests for services (all new connections) go to the same cluster node and making sure all packets for an established connection return to the same cluster node. The former is called persistence and the latter is handled by connection tracking.*

---

<sup>3</sup> Or VIP addresses.

<sup>4</sup> The network that connects the real servers to the Director.

<sup>5</sup> Throughout the IP network conversation or session for a particular cluster service.

## The Director's Connection Tracking Table

The Director's connection tracking table (also sometimes called the *IPVS connection tracking table*, or the *hash table*) contains 128 bytes for each new connection from a client computer in order to store just enough information to return packets to the same real server when the client computer sends in another network packet during the same network connection.

### **Hash Table Structure**

The tracking table consists of both rows and columns. Each row is called a *hash bucket*, and each column is a *connection tracking record*. Each record in the connection tracking table contains timer information; the protocol used by the packet; the client's IP address or CIP and the client's port number; the virtual IP address or VIP and the VIP port number; and some additional control information. Each row, or hash bucket, can contain an unlimited number of these records.

Because the Director looks for a match in this table every time it receives a packet, table lookups need to be as fast as possible. LVS speeds up table lookups by using a hashing technique to determine which row it should search first. Ideally, then, the table should have a very small number of connection records stored in each bucket (each row), because it takes less time for the hash function to locate the correct bucket than it would to run a sequential search through a large number of records within the bucket. The LVS programmers recommend 16 records per row and no more than 20.

### **Controlling the Hash Buckets**

You control the number of buckets (rows) in the LVS hash table with the kernel option "IP masquerading VS table size" in kernel 2.2, and with "IPVS connection table size" in 2.4 and later. The number two is raised to the power of the value you enter for this kernel parameter; by default, LVS will create  $2^{12}$  or 4,096 hash buckets in the LVS hash table. This is the number of hash buckets the Director will use to speed connection record lookups, not the maximum number of simultaneous connections your Director will support. (The number of connection records the table will hold is limited only by the amount of available memory on the Director.)

**NOTE** *One client may have multiple connection tracking table entries if it is accessing cluster resources on different ports (each connection is one connection tracking record in the connection tracking table).*

## **Viewing the Connection Tracking Table**

In the 2.4 and later series kernel, you can view the contents of the connection tracking table with the command:<sup>6</sup>

---

```
#ipvsadm -lcn
```

---

The size of the connection tracking table is displayed when you run the ipvsadm command:

---

```
#ipvsadm
IP Virtual Server version 0.8.2 (size=4096)
```

---

This first line of output from ipvsadm shows that the size of the connection tracking table is 4,096 bytes (the default).

## **Timeout Values for Connection Tracking Records**

When network communication between a client computer and a real server (cluster node) is no longer active, the timeout value for a connection tracking record is set to expire using a timeout value in the connection tracking record. In other words, for services like telnet that use the TCP protocol, the Director will hold the connection tracking record in memory as long as the TCP connection is in an ESTABLISHED state and packets are received from the client computer; when the TCP connection drops, the connection tracking record timeout value is set so the record will eventually be removed from the connection tracking table.

LVS also uses a larger connection tracking timeout value for *all* connection tracking records in the hash table so that it can remove connections that remain unused for a long period of time.

**NOTE** *The kernel also has TCP session timeout values, but they are much larger than the values imposed by LVS. For example, the ESTABLISHED TCP connection timeout value is five days in the 2.4 kernel. See the kernel source code file ip\_conntrack\_proto\_tcp.c for a complete list and the default values used by the kernel.*

LVS has three important timeout values for expiring connection tracking records:

- A timeout value for idle TCP sessions.
- A timeout value for TCP sessions after the client computer has closed the connection (a FIN packet was received from the client computer).<sup>7</sup>

---

<sup>6</sup> In the 2.2 series kernel, you can view the contents of this table with the command `#netstat -Mn`.

<sup>7</sup> For a discussion of TCP states and the FIN packet, see RFC 793.

- A timeout value for UDP packets. Because UDP is a *connectionless* protocol, the LVS Director expires UDP connection tracking records if another packet from the client computer is not received within an arbitrary timeout period.

**NOTE** *To see the default values for these timers on a 2.4 series kernel, look at the contents of the timeout\_\* files in the /proc/sys/net/ipv4/vs/ directory.<sup>8</sup> As of this writing, these values are not implemented in the 2.6 kernel, but they will be replaced with setsockopt controls under ipvsadm's control. See the latest version of the ipvsadm man page for details.*

These three timeout values can be modified by specifying the number of seconds to use for each timer using the --set argument of the ipvsadm command. All three values must be specified when you use the --set argument, so the command:

---

```
#ipvsadm --set 28800 30 36000
```

---

sets the connection tracking record timeout values to: 8 hours for established but idle TCP sessions, 30 seconds for TCP sessions after a FIN packet is received, and 10 minutes for each UDP packet.<sup>9</sup>

To implement these timeouts, LVS uses two tables: a connection timeout table called the *ip\_vs\_timeout\_table* and connection state table called *tcp\_states*. When you use ipvsadm to modify timeout values as shown in the previous example, the changes are applied to all current and future connections tracked by LVS inside the Director's kernel using these two tables.

**NOTE** *If you set /proc/sys/net/ipv4/vs/secure\_tcp to a nonzero value, LVS uses a different pair of tables to implement timeouts called vs\_timeout\_table\_dos and vs\_tcp\_states\_dos. You must, therefore, re-issue the ipvsadm command to set timeout values whenever you enable or disable secure\_tcp.*

We will discuss another timer that the Director uses to return a client computer's request for service to the same real server, called the *persistence timeout*, shortly. But first, let's look at how the Director handles packets going in the other direction: from the cluster node to the client computer.

## Return Packets and the Netfilter Hooks

So far in this chapter, we have only discussed incoming packets for virtual services and how these packets pass through the kernel on the Director. When you build an LVS-NAT cluster, however, packets will also pass back

---

<sup>8</sup> The values in this directory are only used on 2.4 kernels and only if the /proc/sys/net/ipv4/vs/secure\_tcp is nonzero. Additional timers and variables that you find in this directory are documented on the sysctl page at the LVS website (currently at <http://www.linuxvirtualserver.org/docs/sysctl.html>). These sysctl variables are normally only modified to improve security when building a public web cluster susceptible to a DoS attack.

<sup>9</sup> A value of 0 indicates that the default value should be used (it does not represent infinity).

through the Director in the opposite direction as real servers reply to the client computers. Recall from Chapter 12 that these reply packets must pass back through the Director in an LVS-NAT cluster because the Director needs to perform the Network Address Translation to convert the source IP address in the packets from the real server's RIP to the Director's VIP.

The source IP address Network Address Translation for reply packets from the real servers is made possible on the Director thanks to the fact that LVS is inserted into the FORWARD hook. As packets "walk" back through the Netfilter hooks on their way back through the Director to the client computer, LVS can look them up in its connection tracking table to find out which VIP address it should use to replace the RIP address inside the packet header (again, this is only for LVS-NAT).

Figure 14-2 shows the return path of packets as they pass back through the Director on their way to a client computer when LVS-NAT is used as the forwarding method.

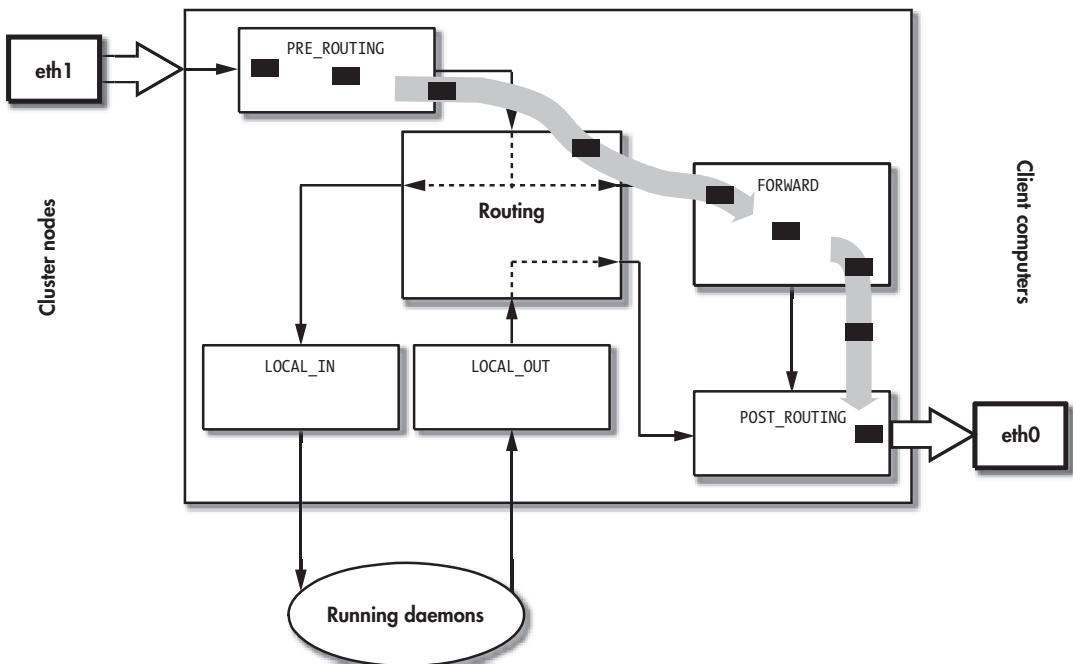


Figure 14-2: Outgoing LVS-NAT packets inside the Director

Notice in Figure 14-2 that the NIC depicted on the left side of the diagram is now the eth1 NIC that is connected to the DRIP network (the network the real servers and Director are connected to). This diagram, in other words, is showing packets going in a direction opposite to the direction of the packets depicted in Figure 14-1. The kernel follows the same rules for processing these inbound reply packets coming from the real servers on its eth1 NIC as it did when it processed the inbound packets from client computers on its eth0 NIC. However, in this case, the destination address of

the packet does not match any of the routing rules in the kernel that would cause the packet to be delivered locally (the destination address of the packet is the client computer's IP address or CIP). The kernel therefore knows that this packet should be sent out to the network, so the packet hits the FORWARD hook, shown by the first gray arrow in Figure 14-2.

This causes the LVS code<sup>10</sup> to de-masquerade the packet by replacing the source address in the packet (sk\_buff) header with the VIP address. The second gray arrow shows what happens next: the packet hits the POST\_ROUTING hook before it is finally sent out the eth0 NIC. (See Chapter 12 for more on LVS-NAT.)

Regardless of which forwarding method you use, the LVS hooks in the Netfilter code allow you to control how real servers are assigned to client computers when new requests come in, even when multiple requests for cluster services come from a single client computer. This brings us to the topic of LVS persistence. But first, let's have a look at LVS without persistence.

## LVS Without Persistence

LVS clusters are normally built without using persistence. The general rule for a Linux Enterprise Cluster is that each new connection to the cluster is sent to the least loaded real server (the cluster node with the fewest number of active connections) in order to achieve the best possible distribution of the workload.<sup>11</sup> To achieve this, you create an IP virtual server table and specify each port separately with commands like these (for LVS-NAT):

---

```
/sbin/ipvsadm -A -t 209.100.100.3:23 -s wrr
/sbin/ipvsadm -a -t 209.100.100.3:23 -r 10.1.1.2 -m
/sbin/ipvsadm -a -t 209.100.100.3:23 -r 10.1.1.3 -m
```

---

The first command creates the LVS virtual service for port 23 (the telnet port) using VIP 209.100.100.3. The next two lines specify which real servers are available to service requests that come in on port 23 for this VIP. (Recall from Chapter 12 that the `-m` option tells the Director to use the LVS-NAT forwarding method, though any LVS forwarding method would work.)

As the Director receives each new network connection request, the real server with the fewest connections is selected, using the weighted round robin or wrr scheduling method. As a result, if a client computer opens three telnet sessions to the cluster, it may be assigned to three different real servers.

This may not be desirable, however. If, for example, you are using a licensing scheme such as the Flexlm *floating license*<sup>12</sup> (which can cause a single user to use multiple licenses when they run several instances of the

---

<sup>10</sup> Called *ip\_vs\_out*.

<sup>11</sup> Here I am making a general correlation between number of connections and workload.

<sup>12</sup> Flexlm has other more cluster-friendly license schemes such as a per-user license scheme.

same application on multiple real servers), you may *want* all of the connections from a client computer to go to the same real server regardless of how that affects load balancing.

Keep in mind, however, that using LVS persistence can lead to load-balancing problems when many clients appear to come from a single IP address (they may all be behind a single NAT firewall), or if they are, in fact, all coming from a single IP address (all users may be logged on to one terminal server or one thin client server<sup>13</sup>). In such a case, all of the client computers will be assigned to a single real server and possibly overwhelm it even though other nodes in the cluster are available to handle the workload. For this reason, you should avoid using LVS persistence if you are using Thin Clients or a terminal server.

In the next few sections, we'll examine LVS persistence more closely and discuss when to consider using it in a Linux Enterprise Cluster.

## LVS Persistence

Regardless of the LVS forwarding method you choose, if you need to make sure all of the connections from a client return to the same real server, you need LVS persistence. For example, you may want to use LVS persistence to avoid wasting licenses if one user (one client computer) needs to run multiple instances of the same application.<sup>14</sup> Persistence is also often desirable with SSL because the key exchange process required to establish an SSL connection will only need to be done one time when you enable persistence.

### ***Persistent Connection Template***

When using LVS persistence, the Director is internally using a connection tracking record called a *persistent connection template* to ensure that all connections from the client computer are assigned to the same real server. As the client computer makes connection requests to the cluster, the director creates a normal connection tracking record for each connection, but it does so only after it looks at the persistent connection template record and decides which real server has already been assigned to this type of connection. What do I mean by type of connection? I'll discuss that in a moment, but first let me explain how the Director removes persistent connection templates based on a timeout value you specify.

### **Persistence Timeout**

Use the ipvsadm utility to specify a timeout value for the persistent connection template on the Director. I'll show you which ipvsadm commands to use to set the persistence timeout for each type of connection in a moment when I get to the discussion of the types of LVS persistence. The timeout value you

---

<sup>13</sup> See the Linux Terminal Server Project at <http://www.ltsp.org>.

<sup>14</sup> Again, this depends on your licensing scheme.

specify is used to set a timer for each persistent connection template as it is created. The timer will count down to zero whether or not the connection is active, and can be viewed<sup>15</sup> with `ipvsadm -L -c`.

If the counter reaches zero and the connection is still active (the client computer is still communicating with the real server), the counter will reset to a default value<sup>16</sup> of two minutes *regardless of the persistent timeout value you specified* and will then begin counting down to zero again. The counter is then reset to the default timeout value each time the counter reaches 0 as long as the connection remains active.

**NOTE** *You can also use the ipvsadm utility to specify a TCP session timeout value that may be larger than the persistent timeout value you specified. A large TCP session timeout value will therefore also increase the amount of time a connection template entry remains on the Director, which may be greater than the persistent timeout value you specify.*

### **Types of Persistent Connections**

Now that I've introduced you to the method LVS uses to expire unused persistent connection template records, let's examine the five types of persistent connections. They are:

1. *Persistent client connections* (PCC), which cause all services a client is accessing to persist. (Also called *zero port connections*.)
2. *Persistent port connections* (PPC), which cause a single service to persist.
3. *Persistent Netfilter Marked Packet persistence*, which causes packets that have been marked with the ipchains/iptables utility to persist.
4. *FTP connections* (FTP connections require careful handling due to the complex<sup>17</sup> nature of FTP connections).
5. *Expired persistence*, which is used internally by the Director to expire connection tracking entries when the persistent connection template expires.<sup>18</sup>

---

<sup>15</sup> The value in the “expire” column.

<sup>16</sup> Controlled by the `TIME_WAIT` timeout value in the IPVS code. If you are using secure tcp to prevent a DoS attack as discussed earlier in this chapter, the default `TIME_WAIT` timeout value is one minute.

<sup>17</sup> In normal FTP connections, the server connects back to the client to send or receive data. FTP can also switch to a passive mode so that the client can connect to the server to send or receive data.

<sup>18</sup> To avoid the time-consuming process of searching through the connection tracking table for all of the connections that are no longer valid and removing them (because a persistent connection template cannot be removed until after the connection tracking records it is associated with expire), LVS simply enters 65535 for the port numbers in the persistent connection template entry and allows the connection tracking records to expire normally (which will ultimately cause the persistent connection template entry to expire).

**NOTE** *If you are building a web cluster, you may need to set the persistence granularity that LVS should use to group CIPs. Normally, each CIP is treated as a unique address when LVS looks up records in the persistent connection template. You can, however, group CIPs using a network mask (see the `-M` option on the `ipvsadm` man page and the LVS HOWTO for details).*

We are most interested in PPC, PCC, and Netfilter Marked Packet persistence.

### **Persistent Client Connection (PCC)**

A persistent client connection (PCC) forces all connections from a client computer to a single real server. A PCC is simply a virtual service created with no port number (or port number 0) and with the `-p` flag set. You would use a persistence client connection when you want all of the connections from a client computer to go to the same real server. If a customer adds items to a shopping cart on your web cluster using the HTTP protocol and then clicks the checkout button to use the encrypted HTTPS protocol, you want to use a persistent client connection so that both port 80 (HTTP) and port 443 (HTTPS) will go to the same real server inside the cluster. For example:

---

```
/sbin/ipvsadm -A -t 209.100.100.3:0 -s rr -p
/sbin/ipvsadm -a -t 209.100.100.3:0 -r 10.1.1.2 -m
/sbin/ipvsadm -a -t 209.100.100.3:0 -r 10.1.1.3 -m
```

---

These three lines create a PCC virtual service on VIP address 209.100.100.3 using real servers 10.1.1.2 and 10.1.1.3.

The default timeout value of 360 seconds can be modified by supplying the `-p` option with the number of seconds that the persistent connection template should remain on the Director. For example, to create a one-hour PCC virtual service, use:

---

```
/sbin/ipvsadm -A -t 209.100.100.3:0 -s rr -p 3600
/sbin/ipvsadm -a -t 209.100.100.3:0 -r 10.1.1.2 -m
/sbin/ipvsadm -a -t 209.100.100.3:0 -r 10.1.1.3 -m
```

---

### **Persistent Port Connection (PPC)**

A persistent port connection (PPC) forces all connections from a client computer for a particular destination port number to the same real server. For example, let's say you want to allow a user to create multiple telnet sessions,<sup>19</sup> and you would like all of the telnet sessions to go to the same real server; however, when the user calls up a web page, you'd like to assign this

---

<sup>19</sup> From one client computer.

request to any node, regardless of which real server they are using for telnet. In this case, you could use persistent port connections for the telnet port (port 23) and the HTTP port (port 80), as follows:

---

```
/sbin/ipvsadm -A -t 209.100.100.3:80 -s rr -p 3600
/sbin/ipvsadm -a -t 209.100.100.3:80 -r 10.1.1.2 -m
/sbin/ipvsadm -a -t 209.100.100.3:80 -r 10.1.1.3 -m
/sbin/ipvsadm -A -t 209.100.100.3:23 -s rr -p 3600
/sbin/ipvsadm -a -t 209.100.100.3:23 -r 10.1.1.2 -m
/sbin/ipvsadm -a -t 209.100.100.3:23 -r 10.1.1.3 -m
```

---

## **Port Affinity**

The key difference between PCC and PPC persistence is sometimes called *port affinity*. With PCC persistence, all connections to the cluster from a single client computer end up on the same real server. Thus, a client computer talking on port 80 (using HTTP) will connect to the same real server when it needs to use port 443 (for HTTPS). When you use PCC, ports 80 and 443 are therefore said to have an affinity with each other.

On the other hand, when you create an IPVS table with multiple ports using PPC, the Director creates one connection tracking template record for each port the client uses to talk to the cluster so that one client computer may be assigned to multiple real servers (one for each port used). With PPC persistence, the ports do not have an affinity with each other.

In fact, when using PCC persistence, all ports have an affinity with each other. This may increase the chance of an imbalanced cluster load if client computers need to connect to several port numbers to use the cluster. To create port affinity that *only applies to a specific set of ports* (port 80 and port 443 for HTTP and HTTPS communication, for example), use Netfilter Marked Packets.

## **Netfilter Marked Packets**

Netfilter Marked Packets (formerly called *fwmarked packets*) have been marked by either the ipchains or the iptables utility on the Director. The Netfilter mark only affects the packet while it is on the Director; once the packet leaves the Director, it is no longer marked (real servers can't see the marks made on the Director).

When you specify the criteria for marking a packet (using iptables or ipchains), you assign a mark number. This number is then associated with an IP virtual service (using the ipvsadm utility), so that the marked packets will be sent to the proper real server.

The Netfilter mark number is placed in the socket buffer (called *sk\_buff*), not in the packet header, and is only associated with the packet while it is being processed by the kernel on the Director.

Your iptables rules cause the Netfilter mark number to be placed inside the packet `sk_buff` entry before the packet passes through the routing process in the `PRE_ROUTING` hook. Once the packet completes through the routing process and the kernel decides which packets should be delivered locally, it reaches the `LOCAL_IN` hook. At this point, the LVS code sees the packet and can check the Netfilter mark number to determine which IP virtual server (IPVS) to use, as shown in Figure 14-3.

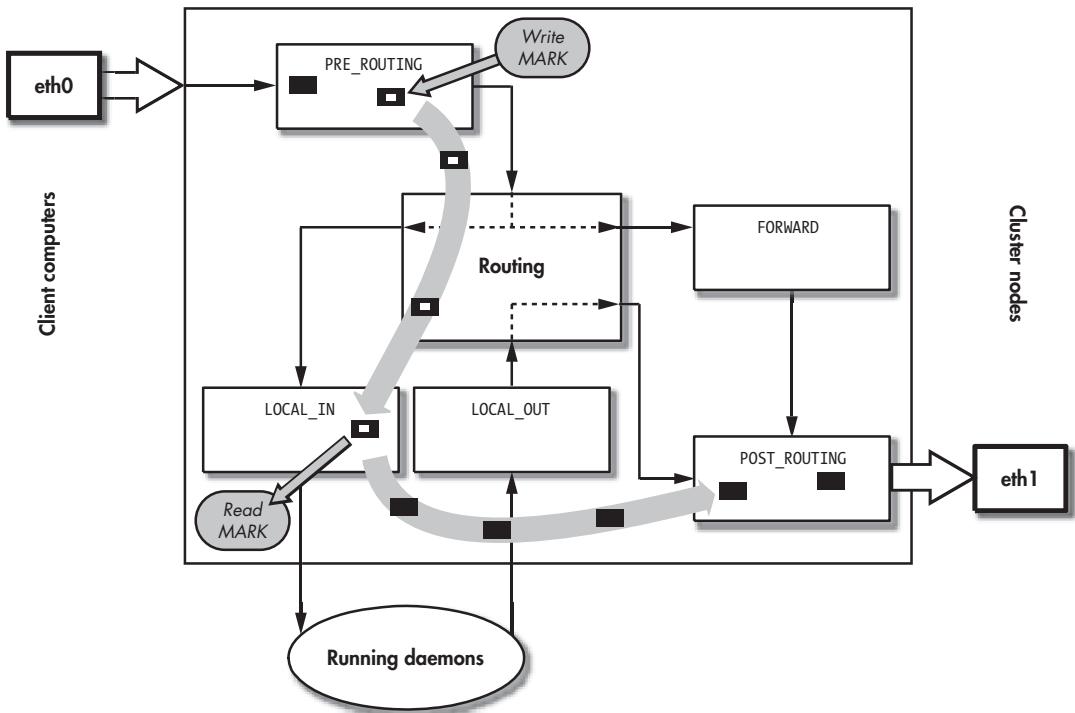


Figure 14-3: Netfilter Marked Packets and LVS

Notice in Figure 14-3 that the Netfilter mark is placed into the incoming packet header (shown as a white square inside of the black box representing packets) in the `PRE_ROUTING` hook. The Netfilter mark can then be read by the LVS code that searches for IPVS packets in the `LOCAL_IN` hook. Normally, only packets destined for a VIP address that has been configured as a virtual service by the `ipvsadm` utility are selected by the LVS code, but if you are using Netfilter to mark packets as shown in this diagram, you can build a virtual service that selects packets based on the mark number.<sup>20</sup> LVS can then send these packets back out a NIC (shown as an `eth1` in this example) for processing on a real server.

<sup>20</sup> Regardless of the destination address in the packet header.

## Marking Packets with iptables

For example, say we want to create one-hour persistent port affinity between ports 80 and 443 on VIP 209.100.100.3 for real servers 10.1.1.2 and 10.1.1.3. We'd use these `iptables` and `ipvsadm` commands:

---

```
/sbin/iptables -F -t mangle
/sbin/iptables -A PREROUTING -i eth0 -t mangle -p tcp \
    -d 209.100.100.3/24 --dport 80 -j MARK --set-mark 99
/sbin/iptables -A PREROUTING -i eth0 -t mangle -p tcp \
    -d 209.100.100.3/24 --dport 443 -j MARK --set-mark 99
/sbin/ipvsadm -A -f 99 -s rr -p 3600
/sbin/ipvsadm -a -f 99 -r 10.1.1.2 -m
/sbin/ipvsadm -a -f 99 -r 10.1.1.3 -m
```

---

**NOTE** *The \ character in the above commands means the command continues on the next line.*

The command in the first line flushes all of the rules out of the `iptables` mangle table. This is the table that allows you to insert your own rules into the `PRE_ROUTING` hook.

The commands on the second line contain the criteria we want to use to select packets for marking. We're selecting packets that are destined for our VIP address (209.100.100.3) with a netmask of 255.255.255.0. The /24 indicates the first 24 bits are the netmask. The balance of that line says to mark packets that are trying to reach port 80 and port 443 with the Netfilter mark number 99.

The commands on the last three lines use `ipvsadm` to send these packets using round robin scheduling and LVS-NAT forwarding to real servers 10.1.1.2 and 10.1.1.3.

To view the `iptables` we have created, you would enter:

---

```
#iptables -L -t mangle -n
Chain PREROUTING (policy ACCEPT)
target     prot opt source      destination
MARK       tcp  --  0.0.0.0/0  209.100.100.3      tcp dpt:80 MARK set 0x63
MARK       tcp  --  0.0.0.0/0  209.100.100.3      tcp dpt:443 MARK set 0x63

Chain OUTPUT (policy ACCEPT)
target     prot opt source      destination
```

---

Recall from Chapter 2 that `iptables` uses table names instead of Netfilter hooks to simplify the administration of the kernel netfilter rules. The chain called `PREROUTING` in this report refers to the netfilter `PRE_ROUTING` hook shown in Figure 14-3. So this report shows two `MARK` rules that will now be applied to packets as they hit the `PRE_ROUTING` hook based on destination address, protocol (tcp), and destination ports (dpt) 80 and 443. The report shows that packets that match this criteria will have their `MARK` number set to hexadecimal 63 (0x63), which is decimal 99.

Now, let's examine the LVS IP Virtual Server routing rules we've created with the following command:

---

```
#ipvsadm -L -n
IP Virtual Server version x.x.x (size=4096)
Prot LocalAddress:Port Scheduler Flags
  -> RemoteAddress:Port      Forward  Weight  ActiveConn  InActConn
FWM  99  rr  persistent 3600
    -> 10.1.1.2:0          Masq      1        0          0
    -> 10.1.1.3:0          Masq      1        0          0
```

---

The output of this command shows that packets with a Netfilter marked (fwmarked) FWM value of decimal 99 will be sent to real servers 10.1.1.2 and 10.1.1.3 for processing using the Masq forwarding method (recall from Chapter 11 that Masq refers to the LVS-NAT forwarding method).

Notice the flexibility and power we have when using iptables to mark packets that we can then forward to real servers with ipvsadm forwarding and scheduling rules. You can use any criteria available to the iptables utility to mark packets when they arrive on the Director, at which point these packets can be sent via LVS scheduling and routing methods to real servers inside the cluster for processing.

The iptables utility can select packets based on the source address, destination address, or port number and a variety of other criteria. See Chapter 2 for more examples of the iptables utility. (The same rules that match packets for acceptance into the system that were provided in Chapter 2 can also match packets for marking when you use the `-t mangle` option and the PREROUTING chain.) Also see the iptables man page for more information or search the Web for “Rusty’s Remarkably Unreliable Guides.”

## In Conclusion

You don't have to understand exactly how the LVS code hooks into the Linux kernel to build a cluster load balancer, but in this chapter I've provided you with a look behind the scenes at what is going on inside of the Director so you'll know at what points you can alter the fate of a packet as it passes through the Director. To balance the load of incoming requests across the cluster using an LVS Director, however, you will need to understand how LVS implements persistence and when to use it, and when not to use it.



# 15

## THE HIGH-AVAILABILITY CLUSTER



In this chapter, we'll combine the Heartbeat package with the LVS software to build a high-availability LVS-DR cluster with no single point of failure that will serve as the foundation for our enterprise-class cluster. (We'll add a filesystem, a method of scheduling batch jobs, and application monitoring software in later chapters.)

In order to build a high-availability cluster we need to use redundant LVS Directors, and we must be able to automatically remove real servers (cluster nodes) when they fail. Before we discuss all of the design goals of a high-availability cluster, we'll examine these two basic requirements.

**NOTE** *See the [Keepalived](#) project for another approach to building a high-availability cluster.*

### Redundant LVS Directors

A high-availability LVS cluster has two LVS Directors: a primary and a backup. When the two Directors first boot, the primary Director owns the

cluster load-balancing resource (the VIP and the LVS forwarding and scheduling rules). The secondary (backup) Director listens to the heartbeats coming from the primary Director. If the secondary Director does not hear the primary server's heartbeat, it initiates a failover and takes ownership of the cluster load-balancing resource by doing the following:

- Adding the VIP address to one of its NICs.
- Sending GARP broadcasts to let other machines on the network know that it now has the VIP.
- Creating an IPVS table to load balance incoming requests for (virtual) services.
- Possibly shutting off the power (Stonith) to the primary Director.

## High-Availability Cluster Design Goals

To be highly available with no single point of failure, our cluster must handle these situations in the following ways:

- If the Primary Director no longer responds to requests for cluster resources on its public network interface, the secondary Director should take over and shut off or reset the power to the Primary Director (to avoid a split-brain condition).
- If one of the real servers (cluster nodes) inside the cluster no longer responds to requests for its services, the real server should be removed from the cluster.
- If the Director is operating in LocalNode<sup>1</sup> mode and a locally running daemon stops responding, the Director should stop routing requests for cluster resources locally and forward all requests to real servers inside the cluster instead.<sup>2</sup>

A highly available cluster should also:

- Use separate physical switches for the primary and secondary Director in order to avoid a single point of failure with the network hardware.
- Use a highly available NAS server in order to avoid a single point of failure with the shared file system. (See Chapter 16 for a discussion of a shared storage technique that address lock arbitration problems.)<sup>3</sup>

**NOTE** *Many LVS clusters will not need to use NAS for shared storage. See Chapter 20 for a discussion of SQL database servers and a brief overview of the Zope project—two examples of sharing data that do not use NAS.*

---

<sup>1</sup> LocalNode was introduced in Chapter 12.

<sup>2</sup> Although this is normally a very unlikely scenario.

<sup>3</sup> Your NAS vendor will have a redundancy solution for this problem as well.

- Use a hardened Linux distribution such as EnGarde or LIDS (<http://www.lids.org>) in order to protect the real servers when the real servers are connected to the Internet.

**NOTE** See “LVS defense strategies against DoS attack” on the LVS website for ways to drop inbound requests in order to avoid overwhelming an LVS Director when it is under attack. Building a cluster that uses these techniques is outside the scope of this book.

## The High-Availability LVS-DR Cluster

In this chapter, we will describe how to build a Linux Enterprise Cluster that uses the LVS-DR forwarding method in conjunction with the Heartbeat package for high availability.

A diagram of a highly available LVS cluster with two Directors is shown in Figure 15-1. The Directors, real servers, and client computers are all connected to the same network segment. The short dotted arrow connecting the LVS Director and the backup LVS Director in this figure represents a separate physical connection for heartbeats.

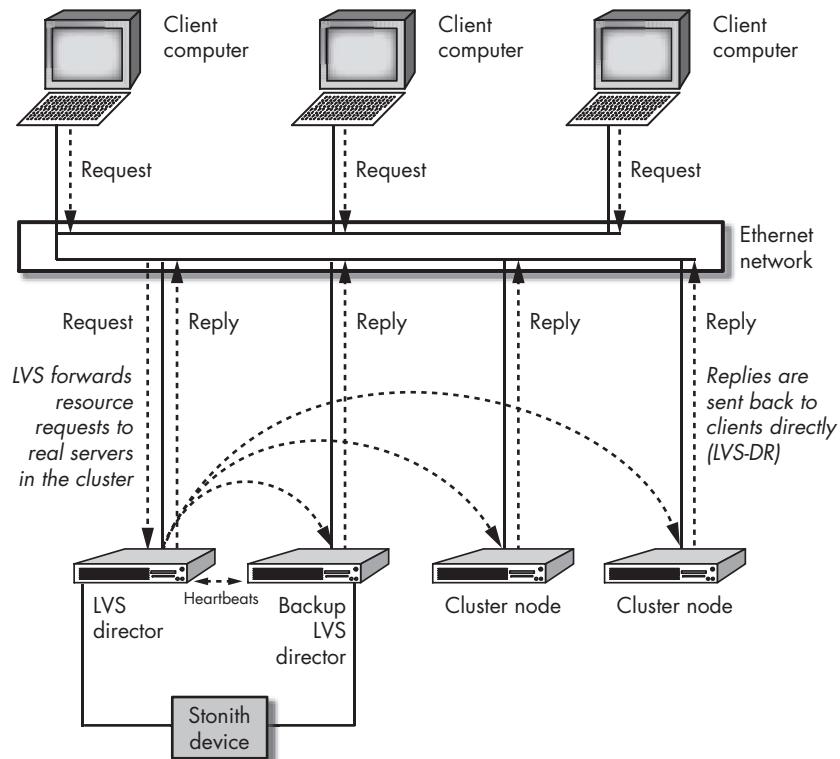


Figure 15-1: A highly available DR cluster

Using this configuration, the heartbeats can pass over the normal Ethernet network and a dedicated network or serial cable connection between the LVS Director and the backup LVS Director. Recall from Chapter 6 that heartbeats should be able to travel over two physically independent paths between the servers in a high-availability server pair.<sup>4</sup>

This diagram also shows a Stonith device (as described in Chapter 9) that provides true high availability. The backup node has a serial or network connection to the Stonith device so that Heartbeat can send power reset commands to it. The primary node has its power cable plugged into the Stonith device.<sup>5</sup>

The dashed gray arrows that originate from the LVS Director and point to each cluster node represent incoming packets from the client computers. Because this is an LVS-DR cluster, the reply packets are sent directly back out onto the Ethernet network by each of the cluster nodes and directly back to the client computers connected to this Ethernet network.

As processing needs grow, additional cluster nodes (LVS real servers) can easily be added to this configuration,<sup>6</sup> though two machines outside the cluster will always act as a primary and backup Director to balance the incoming workload.

## Introduction to `ldirectord`

To failover the LVS load-balancing resource from the primary Director to the backup Director and automatically remove nodes from the cluster, we need to use the `ldirectord` program. This program automatically builds the IPVS table when it first starts up and then it monitors the health of the cluster nodes and removes them from the IPVS table (on the Director) when they fail.

### ***How `ldirectord` Monitors Cluster Nodes (LVS Real Servers)***

The `ldirectord` daemon monitors the health of real servers by sending requests to access cluster resources on the real IP (RIP) of each real server. This is true for all LVS cluster types: LVS-DR, LVS-TUN, and LVS-NAT. Normally, one `ldirectord` daemon runs for each VIP address on the Director. When a real server does not reply to the `ldirectord` daemon running on the Director, the `ldirectord` daemon issues the correct `ipvsadm` command to remove it from the IPVS table for the VIP address. (Later, when the real server comes back online, `ldirectord` issues the correct `ipvsadm` command to add the real server back into the IPVS table.)

---

<sup>4</sup> Avoid the temptation to put heartbeats only on the real or public network (as a means of service monitoring), especially when using Stonith as described in this configuration. See Chapter 9 for more information.

<sup>5</sup> See Chapter 9 for a discussion of the limitations imposed on a high availability configuration that does not use two Stonith devices.

<sup>6</sup> Using the `SystemImager` package described in Chapter 5.

To monitor real servers inside a web cluster, for example, the `ldirectord` daemon uses the HTTP protocol to ask each real server for a specific web page. The Director knows what it should receive back from the real servers if they are healthy. If the reply string, or web page, takes too long to come back from the real server, or never comes back at all, or comes back changed in some way, `ldirectord` will know something is wrong, and it will remove the node from the IPVS table of valid real servers for a given VIP.

Figure 15-2 shows the `ldirectord` request to view a special health check web page (not normally viewed by client computers) on the first real server.

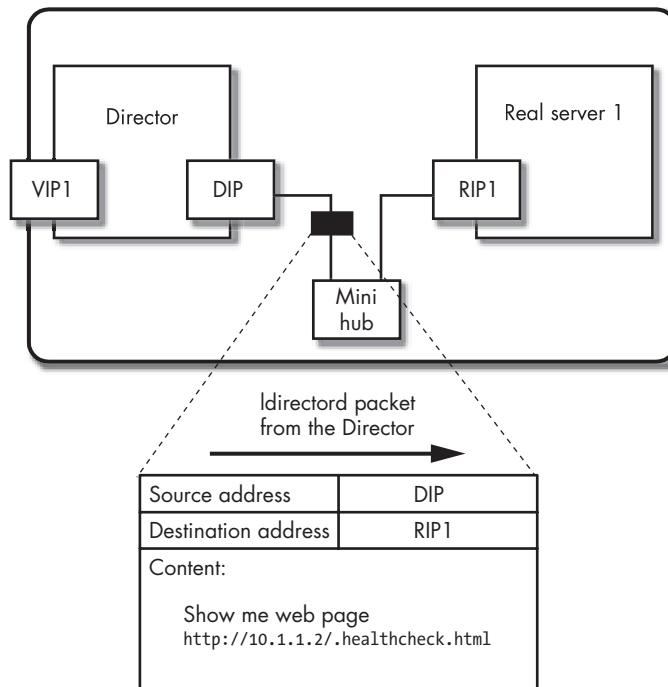


Figure 15-2: `ldirectord` requests a health check URL

The URL in this figure (`http://10.1.1.2/.healthcheck.html`) relies on a file named `.healthcheck.html` on real server 1 at RIP `10.1.1.2`. The Apache web server daemon running on real server 1 will use this file (which contains only the word `OKAY`) to reply to the `ldirectord` daemon as shown in Figure 15-3.

Notice in Figure 15-2 that the destination address of the packet is the RIP address of a real server (real server 1). Also notice that the return address of the packet is the Director's IP address (the DIP).

In Figure 15-3, the packet sent back from the real server to the Director's DIP is shown.

If anything other than the expected reply (the word `OKAY` as shown in this example) is received by the `ldirectord` daemon on the Director, the real server will be removed from the IPVS table.

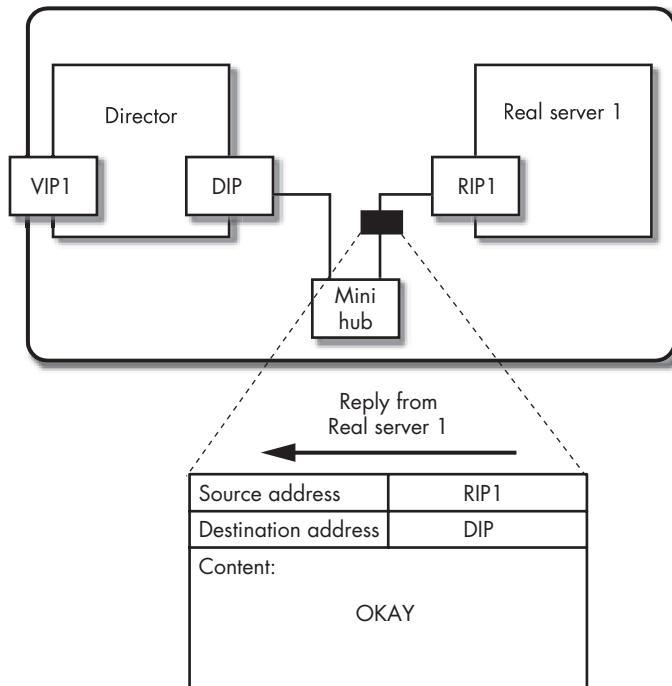


Figure 15-3: Real server 1 sends back the reply

Health checks for cluster resources must be made on the RIP addresses of the real servers inside the cluster *regardless of the LVS forwarding method used*. In the case of the HTTP protocol, the Apache web server running on the real servers must be listening for requests on the RIP address. (Appendix F contains instructions for configuring Apache in a cluster environment.) Apache must be configured to allow requests for the health check web page or health check CGI script on the RIP address.

### DISABLING SUPERFLUOUS APACHE MESSAGES

Apache normally writes a log entry each time the health check web page is accessed. This will quickly cause your access\_log and disk drive to fill up with useless information. To avoid this situation, create an httpd.conf entry like this:

---

```
SetEnvIf Request_URI \.healthcheck\.html$ dontlog
```

---

Place this line before the CustomLog entry for the access\_log in the httpd.conf file. Next, modify the CustomLog entry to ignore all requests that have been flagged with the dontlog environment variable by adding env=!dontlog so that the CustomLog line now looks like this:

---

```
CustomLog logs/access_log combined env=!dontlog
```

---

Then restart the Apache daemon.

**NOTE** *Normally, the Director sends all requests for cluster resources to VIP addresses on the LVS-DR real servers, but for cluster health check monitoring, the Director must use the RIP address of the LVS-DR real server and not the VIP address.*

Because ldirectord is using the RIP address to monitor the real server and not the VIP address, the cluster service might not be available on the VIP address even though the health check web page is available at the RIP address.

## LVS, Heartbeat, and ldirectord Recipe

In this recipe, we'll build the highly available LVS-DR cluster shown in Figure 15-1.

**NOTE** *If you haven't already built an LVS-DR according to the instructions in Chapter 13, you should do so before you attempt to follow this recipe.*

List of ingredients:

Kernel with LVS support and the ability to suppress ARP replies (2.4.26 or greater in the 2.4 series, 2.6.4 or greater in the 2.6 series)

Copy of the Heartbeat RPM package

Copy of the ldirectord Perl program

### ***Hide the Loopback Interface***

We need to tell the real servers to ignore ARP broadcasts from client computers that are searching for the owner (the MAC address) of the VIP (see the section "ARP Broadcasts and the LVS-DR Cluster" in Chapter 13 for an explanation of why this is required in an LVS-DR cluster).

There are two commonly used methods to accomplish this, and you can use either one: either create a script that is launched by the init program when the system boots (see Chapter 1), or modify the /etc/sysctrl.conf file and issue the sysctl -p command each time the system boots.<sup>7</sup> The chapter15 subdirectory on the CD-ROM contains a copy of the following sample init script that will let you use the first method. The script adds the VIP address to the hidden loopback device, adds a routing table entry<sup>8</sup> for the VIP, enables packet forwarding, and tells the kernel to hide the VIP.

**NOTE** *This script can be used on real servers and on the primary and backup Director because the Heartbeat program<sup>9</sup> knows how to remove a VIP address on the hidden loopback device when it conflicts with an IP address specified in the /etc/ha.d/haresources file. In other words, the Heartbeat program will remove the VIP from the*

---

<sup>7</sup> This method is also discussed on the Ultra Monkey website.

<sup>8</sup> See Chapter 2 for a discussion of the routing table.

<sup>9</sup> The IPAddr and IPAddr2 resource scripts.

hidden loopback device when it first starts up<sup>10</sup> on the primary Director, and it will remove the VIP from the hidden loopback device on the backup Director when the backup Director needs to acquire the VIP (because the primary Director has crashed).

```
#!/bin/bash
#
# lvsdrss init script to hide loopback interfaces on LVS-DR
# Real servers. Modify this script to suit
# your needs--You at least need to set the correct VIP address(es).
#
# Script to start LVS DR real server.
#
# chkconfig: 2345 20 80
# description: LVS DR real server
#
# You must set the VIP address to use here:
VIP=209.100.100.2
host=`/bin/hostname`
case "$1" in
start)
    # Start LVS-DR real server on this machine.
    /sbin/ifconfig lo down
    /sbin/ifconfig lo up
    echo 1 > /proc/sys/net/ipv4/conf/lo/arp_ignore
    echo 2 > /proc/sys/net/ipv4/conf/lo/arp_announce
    echo 1 > /proc/sys/net/ipv4/conf/all/arp_ignore
    echo 2 > /proc/sys/net/ipv4/conf/all/arp_announce

    /sbin/ifconfig lo:0 $VIP netmask 255.255.255.255 up11
    /sbin/route add -host $VIP dev lo:0
;;
stop)
    # Stop LVS-DR real server loopback device(s).
    /sbin/ifconfig lo:0 down
    echo 0 > /proc/sys/net/ipv4/conf/lo/arp_ignore
    echo 0 > /proc/sys/net/ipv4/conf/lo/arp_announce
    echo 0 > /proc/sys/net/ipv4/conf/all/arp_ignore
    echo 0 > /proc/sys/net/ipv4/conf/all/arp_announce

;;
status)
    # Status of LVS-DR real server.
    islothere=`/sbin/ifconfig lo:0 | grep $VIP`
    isrothere=`netstat -rn | grep "lo:0" | grep $VIP`
    if [ ! "$islothere" -o ! "$isrothere" ];then
        # Either the route or the lo:0 device
        # not found.
```

<sup>10</sup> This assumes that the Heartbeat program is started *after* this script runs.

<sup>11</sup> Use netmask 255.255.255.255 here no matter what netmask you normally use for your network.

```
        echo "LVS-DR real server Stopped."
    else
        echo "LVS-DR Running."
    fi
;;
*)
# Invalid entry.
echo "$0: Usage: $0 {start|status|stop}"
exit 1
;;
esac
```

---

Place this script in the `/etc/init.d` directory on all of the real servers and enable it using `chkconfig`, as described in Chapter 1, so that it will run each time the system boots.

**NOTE** *The Linux Virtual Server website (<http://www.linuxvirtualserver.org>) also contains an installation script that will build an init script similar to this for you automatically.*

### ***Install the Heartbeat on a Primary and a Backup Director***

We will use Heartbeat to start `ldirectord` and bring up the VIP IP alias or secondary IP address on the primary Director. The `ldirectord` program and the VIP will be one resource group (see Chapter 8) under Heartbeat's control. If the primary Director crashes, Heartbeat running on the backup Director will take over this resource group and client computers will continue to be able to access the cluster.

### ***Install `ldirectord` and Its Required Software Components***

`ldirectord` is a Perl program that relies on several prewritten Perl modules that help it to connect to the real servers using HTTP, POP, telnet, and so on. The following sections discuss how to install the necessary Perl modules.

#### **Install Perl Modules Located on the CD-ROM**

The `chapter15/ldirectord/dependencies` directory on the CD-ROM in this book includes the current Perl module dependencies for `ldirectord` from the Ultra Monkey website.<sup>12</sup> Use the following commands to extract and build each Perl module.

---

```
#mkdir -p /usr/local/src/perldeps
#mount /mnt/cdrom
#cp -r /mnt/cdrom/chapter15/ldirectord/dependencies /usr/local/src/perldeps
#cd /usr/local/src/perldeps
#tar xzvf libnet*
#cd libnet-<version>
```

---

<sup>12</sup> They originally came from CPAN.

```
#perl Makefile.PL  
#make  
#make install
```

---

## Download Perl Modules from CPAN

If you choose not to install from the CD-ROM, you can download the necessary Perl modules for free from the Comprehensive Perl Archive Network or CPAN (<http://www.cpan.org>). Here is a list of the protocols and services that Idirectord can monitor, as well as the associated CPAN Perl modules required to monitor them.

Service	PERL Module
HTTP	LWP::UserAgent, (libwww-perl)
HTTPS	Net::SSLeay, (Net_SSLeay)
FTP	Net::FTP (libnet)
LDAP	Net::LDAP (perl-ldap)
NNTP	IO::Socket/IO::Select (already part of Perl)
SMTP	Net::SMTP (libnet)
POP	Net::POP3 (libnet)
IMAP	Mail::IMAPClient (Mail-IMAPClient)

Start the CPAN program with the commands:<sup>13</sup>

---

```
#cd /  
#perl -MCPAN -e shell
```

---

**NOTE** *If you've never run this command before, it will ask if you want to manually configure CPAN. Say no if you want to have the program use its defaults.*

You should install the CPAN module MD5 to perform security checksums on the download files the first time you use this method of downloading CPAN modules. While optional, it gives Perl the ability to verify that it has correctly downloaded a CPAN module using the MD5 checksum process:

---

```
cpan> install MD5
```

---

The MD5 download and installation should finish with a message like this:

---

```
/usr/bin/make install -- OK
```

---

Now install the CPAN modules with the following case-sensitive commands. You will need to respond to questions about your network configuration.

---

<sup>13</sup> You can also use Webmin as a web interface or GUI to download and install CPAN modules.

**NOTE** You do not need to install all of these modules for *ldirectord* to work properly. However, installing all of them allows you to monitor these types of services later by simply making a configuration change to the *ldirectord* configuration file.

---

```
cpan> install LWP::UserAgent
cpan> install Net::SSLeay
cpan> install Mail::IMAPClient
cpan> install Net::LDAP
cpan> install Net::FTP
cpan> install Net::SMTP
cpan> install Net::POP3
```

---

Some of these packages will ask if you want to run tests to make sure that the package installs correctly. These tests will usually involve using the protocol or service you are installing to connect to a server. Although these tests are optional, if you choose not to run them, you may need to force the installation by adding the word **force** at the beginning of the *install* command. For example:

---

```
cpan> force install Mail::IMAPClient
```

---

Each command should complete with the following message to indicate a successful installation:

---

```
/usr/bin/make install -- OK
```

---

**NOTE** *ldirectord* does not need to monitor every service your cluster will offer. If you are using a service that is not supported by *ldirectord*, you can write a CGI program that will run on each real server to perform health or status monitoring and then connect to each of these CGI programs using HTTP.

If you want to use the LVS *configure* script (not described in this book) to configure your Directors, you should also install the following modules:

---

```
cpan> install Net::DNS
cpan> install strict
cpan> install Socket
```

---

When you are done with CPAN, enter the following command to return to your shell prompt:

---

```
cpan>quit
```

---

The Perl modules we just downloaded should have been installed under the `/usr/local/lib/perl<MAJOR-VERSION-NUMBER>/site_perl/<VERSION-NUMBER>` directory. Examine this directory to see where the LWP and Net directories are located, and then run the following command.

---

```
#perl -V
```

---

The last few lines of output from this command will show where Perl looks for modules on your system, in the `@INC` variable. If this variable does not point to the Net and SWP directories you downloaded from CPAN, you need to tell Perl where to locate these modules. (One easy way to update the directories Perl uses to locate modules is to simply upgrade to a newer version of Perl.)

**NOTE** *In rare cases, you may want to modify Perl's `@INC` variable. See the description of the `-I` option on the `perlrun` man page (type `man perlrun`), or you can also add the `-I` option to the `ldirectord` Perl script after you have installed it.*

### ***Install `ldirectord`***

You'll find `ldirectord` in the `chapter15` directory on the CD-ROM. To install this version enter:

---

```
#mount /mnt/cdrom
#cp /mnt/cdrom/chapter15/ldirectord* /etc/ha.d/resource.d/ldirectord
#chmod 755 /usr/sbin/ldirectord
```

---

You can also download and install `ldirectord` from the Linux HA CVS source tree (<http://cvs.linux-ha.org>), or the `ldirectord` website (<http://www.vergenet.net/linux/ldirectord>). Be sure to place `ldirectord` in the `/usr/sbin` directory and to give it execute permission.

**NOTE** *Remember to install the `ipvsadm` utility (located in the `chapter12` directory) before you attempt to run `ldirectord`.*

### ***Test Your `ldirectord` Installation***

To test the installation of `ldirectord`, ask it to display its help information with the `-h` switch.

---

```
#/usr/sbin/ldirectord -h
```

---

Your screen should show the `ldirectord` help page. If it does not, you will probably see a message about a problem finding the necessary modules.<sup>14</sup> If you receive this message, and you cannot or do not want to download the Perl modules using CPAN, you can edit the `ldirectord` program and

---

<sup>14</sup> Older versions of `ldirectord` would only display the help page if you ran the program as a non-root user. If you see a security error message when you run this command, you should upgrade to at least version 1.46.

comment out the Perl code that calls the methods of healthcheck monitoring that you are not interested in using.<sup>15</sup> You'll need at least one module, though, to monitor your real servers inside the cluster.

## **Create the ldirectord Configuration File**

ldirectord uses a configuration file to build the IPVS table. You can call this file any legal file name you wish, but you must place it in the `/etc/ha.d/conf` directory. For example, the configuration file for the IPVS on VIP `209.100.100.3` can be:

---

```
checktimeout=20
checkinterval=5
autoreload=yes
quiescent=no
logfile="info"
virtual=209.100.100.3:80
    real=127.0.0.1:80 gate 1 ".healthcheck.html", "OKAY"
    real=209.100.100.100:80 gate 1 ".healthcheck.html", "OKAY"
    service=http
    checkport=80
    protocol=tcp
    scheduler=wrr
    checktype=negotiate
    fallback=127.0.0.1
```

---

**NOTE** *You must indent the lines after the virtual line with at least four spaces or the tab character.*

The first four lines in this file are “global” settings; they apply to multiple virtual hosts. When used with Heartbeat, however, this file should normally contain `virtual=` sections for only one VIP address, as shown here. This is so because when you place each VIP in the `haresources` file on a separate line, you run one `ldirectord` daemon for each VIP and use a different configuration file for each `ldirectord` daemon. Each VIP and its associated IPVS table thus becomes one resource that Heartbeat can manage.

Let's examine each line in this configuration file.

---

```
checktimeout=20
```

---

---

<sup>15</sup> Making custom changes to get `ldirectord` to run in a test environment such as the one being described in this chapter is fine, but for production environments you are better off installing the CPAN modules and using the stock version of `ldirectord` to avoid problems in the future when you need to upgrade.

This sets the `checktimeout` value to the number of seconds `ldirectord` will wait for a health check (normally some type of network connection) to complete. If the check fails for any reason or does not complete within the `checktimeout` period, `ldirectord` will remove the real server from the IPVS table.<sup>16</sup>

---

`checkinterval=5`

---

This `checkinterval` is how long `ldirectord` sleeps between checks.

---

`autoreload=yes`

---

This enables the `autoreload` option, which causes the `ldirectord` program to periodically calculate an `md5sum` to check this configuration file for changes and automatically apply them when you change the file. This handy feature allows you to easily change your cluster configuration. A few seconds after you save changes to the configuration file, the running `ldirectord` daemon will see the change and apply the proper `ipvsadm` commands to implement the change, removing real servers from the pool of available servers or adding them in as needed.<sup>17</sup>

**NOTE** *You can also force `ldirectord` to reload its configuration file by sending the `ldirectord` daemon a `HUP` signal (with the `kill` command) or by running `ldirectord reload`.*

---

`quiescent=no`

---

A node is “quiesced” (its weight is set to 0) when it fails to respond within its `checktimeout` period. When you set this option, `ldirectord` will remove the real server from the IPVS table rather than “quiesce” it. Removing the node from the IPVS table breaks all of the existing client connections and causes LVS to drop all connection tracking records and persistent connection templates for the node. If you do not set this option to `no`, the cluster may appear to be down to some of the client computers when a node crashes, because they were assigned to the node before it crashed and the connection tracking record or persistent connection template still remains on the Director.

When using this option, you may also want to use a command<sup>18</sup> like this at boot time:

---

`echo 1 > /proc/sys/net/ipv4/vs/expire_nodest_conn`

---

<sup>16</sup> The `checktimeout` and `checkinterval` values for `ldirectord` have nothing to do with the `keepalive`, `deadtime`, and `initdead` variables used by Heartbeat.

<sup>17</sup> If you simply want to be able to remove real servers from the cluster for maintenance and you do not want to enable this feature, you can disconnect the real server from the network, and the real server will be removed from the IPVS table automatically when you have configured `ldirectord` properly.

<sup>18</sup> Or set this value in the `/etc/sysctl.conf` file on Red Hat systems.

Setting this kernel variable to 1 causes the connection tracking records to expire immediately if a client with a pre-existing connection tracking entry tries to talk to the same server again but the server is no longer available.<sup>19</sup>

**NOTE** *All of the sysctl variables, including the expire\_nodest\_conn variable, are documented at the LVS website (<http://www.linuxvirtualserver.org/docs/sysctl.html>).*

---

```
logfile="info"
```

---

This entry tells ldirectord to use the syslog facility for logging error messages. (See the file `/etc/syslog.conf` to find out where messages at the “info” level are written.) You can also enter the name of a directory and file to write log messages to (precede the entry with `/`). If no entry is provided, log messages are written to `/var/log/ldirectord.log`.

---

```
virtual=209.100.100.3:80
```

---

This line specifies the VIP addresses and port number that we want to install on the LVS Director. Recall that this is the IP address you will likely add to the DNS and advertise to client computers. In any case, this is the IP address client computers will use to connect to the cluster resource you are configuring.

You can also specify a Netfilter mark (or fwmark) value on this line instead of an IP address. For example, the following entry is also valid:

---

```
virtual=2
```

---

This entry indicates that you are using ipchains or iptables to mark packets as they arrive on the LVS Director<sup>20</sup> based on criteria that you specify with these utilities. All packets that contain this mark will be processed according to the rules on the indented lines that follow in this configuration file.

**NOTE** *Packets are normally marked to create port affinity (between ports 443 and port 80, for example). See Chapter 14 for a discussion of packet marking and port affinity.*

The next group of indented lines specifies which real servers inside the cluster will be able to offer the resource to client computers.

---

```
real=127.0.0.1:80 gate 1 ".healthcheck.html", "OKAY"
```

---

<sup>19</sup> The author has not used this kernel setting in production, but found that setting quiescent to “no” provided an adequate node removal mechanism.

<sup>20</sup> The machine running ldirectord will also be the LVS Director. Recall that packet marks only apply to packets while they are on the Linux machine that inserted the mark. The packet mark created by iptables or ipchains is forgotten as soon as the packet is sent out the network interface.

This line indicates that the Director itself (at loopback IP address 127.0.0.1), acting in LocalNode mode, will be able to respond to client requests destined for the VIP address 209.100.100.3.

**NOTE** *Do not use LocalNode in production unless you have thoroughly tested failing over the cluster load-balancing resource. Normally, you will improve the reliability of your cluster if you avoid using LocalNode mode.*

---

```
real=209.100.100.100:80 gate 1 ".healthcheck.html", "OKAY"
```

---

This line adds the first LVS-DR real server using the RIP address 209.100.100.100.

Each `real=` line in this configuration file uses the following syntax:

---

```
real=RIP:port gate|masq|ipip [weight] "Request URL", "Response Expected"
```

---

This syntax description tells us that the word `gate`, `masq`, or `ipip` must be present on each `real` line in the configuration file to indicate the type of forwarding method used for the real server. (Recall from Chapter 11 that the Director can use a different LVS forwarding method for each real server inside the cluster.) This configuration file is therefore using a slightly different terminology (based on the switch passed to the `ipvsadm` command) to indicate the three forwarding methods. See Table 15-1 for clarification.

**Table 15-1:** Different Terms for the LVS Forwarding Methods

<b>ldirectord Configuration Option</b>	<b>Switch Used When Running ipvsadm</b>	<b>Output of ipvsadm -L Command</b>	<b>LVS Forwarding Method</b>
gate	-g	Route	LVS-DR
ipip	-i	Tunnel	LVS-TUN
masq	-m	Masq	LVS-NAT

Following the forwarding method on each `real=` line is the weight assigned to the real servers in the cluster. This is used only with one of the weighted scheduling methods. The last two parameters on the line indicate which web page or URL ldirectord should ask for when checking the health of the real server, and what response ldirectord should expect to receive from the real server. Both of these parameters need to be in quotes as shown, separated by a comma.

---

```
service=http
```

---

This line indicates which service ldirectord should use when testing the health of the real server. You must have the proper CPAN Perl module loaded for the type of service you specify here.

---

```
checkport=80
```

---

This line indicates that the health check request for the http service should be performed on port 80.

---

```
protocol=tcp
```

---

This entry specifies the protocol that will be used by this virtual service. The protocol can be set to `tcp`, `udp`, or `fwm`. If you use `fwm` to indicate marked packets or fwmarked marked packets, then you must have already used the Netfilter mark number (or fwmark) instead of an IP address on the `virtual=` line that all of these indented lines are associated with.

---

```
scheduler=wrr
```

---

The `scheduler` line indicates that we want to use the weighted round-robin load-balancing technique (see the previous real lines for the weight assignment given to each real server in the cluster). See Chapter 11 for a description of the scheduling methods LVS supports. The `ldirectord` program does not check the validity of this entry; `ldirectord` just passes whatever you enter here on to the `ipvsadm` command to create the virtual service.

---

```
checktype=negotiate
```

---

This option describes which method the `ldirectord` daemon should use to monitor the real servers for this VIP. `checktype` can be set to one of the following:

#### **negotiate**

This method connects to the real server and sends the `request` string you specify. If the `reply` string you specify is not received back from the real server within the `checktimeout` period, the node is considered dead. You can specify the `request` and `reply` strings on a per-node basis as we have done in this example (see the discussion of the `real` lines earlier in this section), or you can specify one `request` and one `reply` string that should be used for all of the real servers by adding two new lines to your `ldirectord` configuration file like this:

---

```
request=".healthcheck.html"
receive="OKAY"
```

---

#### **connect**

This method simply connects to the real server at the specified `checkport` (or port specified for the real server) and assumes everything is okay on the real server if a basic TCP/IP connection is accepted by the real

server. This method is not as reliable as the negotiate method for detecting problems with the resource daemons (such as HTTP) running on the real server. Connect checks are useful when there is no negotiate check available for the service you want to monitor.

#### **A number**

If you enter a number here instead of the word negotiate or connect, the ldirectord daemon will perform the connection test the number of times specified, and then perform one negotiate test. This method reduces the processing power required of the real server when responding to health checks and reduces the amount of cluster network traffic required for health check monitoring of the services on the real servers.<sup>21</sup>

#### **off**

This disables ldirectord's health check monitoring of the real servers.

---

`fallback=127.0.0.1`

---

The fallback address is the IP address and port number that client connections should be redirected to when there are no real servers in the IPVS table that can satisfy their requests for a service. Normally, this will always be the loopback address 127.0.0.1, in order to force client connections to a daemon running locally that is at least capable of informing users that there is a problem, and perhaps letting them know whom to contact for additional help.

You can also specify a special port number for the fallback web page:

---

`fallback=127.0.0.1:9999`

---

**NOTE** *We did not use persistence to create our virtual service table. Persistence is enabled in the ldirectord.conf file using the persistent= entry to specify the number of seconds of persistence. See Chapter 14 for a discussion of LVS persistence.*

### ***Create the Health Check Web Page***

Now, on each real server and on the Director enter a command like this to create a simple health check web page:

---

`#echo "OKAY" > /www/htdocs/.healthcheck.html`

---

**NOTE** *The directory used here should be whatever is specified as the DocumentRoot in your httpd.conf file. Also note the period (.) before the file name in this example.<sup>22</sup>*

---

<sup>21</sup> How much it reduces network traffic is based on the type of service and the size of the response packet that the real server must send back to verify that it is operating normally.

<sup>22</sup> This is not required, but is a carryover from the Unix security concept that file names starting with a period (.) are not listed by the ls command (unless the -a option is used) and are therefore hidden from normal users.

See if the Healthcheck web page displays properly on each real server with the following command:

---

```
#lynx -dump 127.0.0.1/.healthcheck.html
```

---

## ***Start ldirectord Manually and Test Your Configuration***

In a moment we'll use Heartbeat to start our ldirectord monitoring daemon, but first you need to try starting the daemon on the Director to see if your IPVS table is created.

1. Enter this command:

---

```
#/etc/ha.d/resource.d/ldirectord -d ldirectord-209.100.100.3.conf start
```

---

You should see the ldirectord debug output indicating that the ipvsadm commands have been run to create the virtual server followed by commands such as the following:

---

```
DEBUG2: check_http: http://209.100.100.100/.healthcheck.html is down
```

---

2. Now if you bring the real server online at IP address 209.100.100.100, and it is running the Apache web server, the debug output message from ldirectord should change to:

---

```
DEBUG2: check_http: http://209.100.100.100/.healthcheck.html is up
```

---

3. Check to be sure that the virtual service was added to the IPVS table with the command:

---

```
#ipvsadm -L -n
```

---

4. Once you are satisfied with your ldirectord configuration and see that it is able to communicate properly with your health check URLs, press CTRL-C to break out of the debug display.
5. Restart ldirectord (in normal mode) with:

---

```
#/usr/sbin/ldirectord ldirectord-209.100.100.3.conf start
```

---

6. Then start ipvsadm with the `watch` command to automatically update the display:

---

```
#watch ipvsadm -L -n
```

---

You should see an ipvsadm table like this:

---

IP Virtual Server version x.x.x (size=4096)					
Prot	LocalAddress:Port	Scheduler	Flags	Forward	Weight
	-> RemoteAddress:Port			Forward	Weight
TCP	209.100.100.3:80	wrr			
	-> 209.100.100.100:80			Route	1
	-> 127.0.0.1:80			Local	1
					ActiveConn 0
					InActConn 0

---

7. Test ldirectord by shutting down Apache on the real server or disconnecting its network cable. Within 20 seconds, or the checktimeout value you specified, the real server's weight should be set to 0 so that no future connections will be sent to it.
8. Turn the real server back on or restart the Apache daemon on the real server, and you should see its weight return to 1.

### ***Add ldirectord to the Heartbeat Configuration***

Now it's time to tell Heartbeat about our ldirectord configuration so that it will start it automatically at boot time.

1. Follow the recipe in Chapter 7 to install the Heartbeat program if you have not done so already; then create the following entry in your /etc/ha.d/haresources file:

---

```
primary.yourdomain.com 209.100.100.3 ldirectord::ldirectord-209.100.100.3.conf
```

---

where `primary.yourdomain.com` is the host name (as returned by `uname -n`) of your primary Director.

This haresources entry tells Heartbeat that the machine named `primary.yourdomain.com` should normally own the ldirectord resource for VIP 209.100.100.3. The ldirectord resource, or script, is started by Heartbeat, and based on this entry, passed to the configuration file named `/etc/ha.d/conf/ldirectord-209.100.100.3.conf` in this example.

2. Stop ldirectord with the command:

---

```
#/etc/ha.d/resource.d/ldirectord ldirectord-209.100.100.3.conf stop
```

---

3. Make sure ldirectord is no longer running with:

---

```
#ps -elf | grep ldirectord
```

---

This command should not return any output if ldirectord is not running. If you see a running process here, be sure to kill it before going on with the next command.

4. Make sure ldirectord is not started as a normal boot script with the command:

---

```
#chkconfig --del ldirectord
```

---

5. Stop Heartbeat with the command:

---

```
#/etc/rc.d/init.d/heartbeat stop
```

---

6. Clean out the ipvsadm table so that it does not contain any test configuration information with:

---

```
#ipvsadm -C
```

---

7. Now start the heartbeat daemon with:

---

```
#/etc/rc.d/init.d/heartbeat start
```

---

and watch the output of the heartbeat daemon using the following command:

---

```
#tail -f /var/log/messages
```

---

**NOTE** *For the moment, we do not need to start Heartbeat on the backup Heartbeat server.*

The last line of output from Heartbeat should look like this:

---

```
heartbeat: info: Running /etc/ha.d/resource.d/ldirectord ldirectord-209.100.100.3.conf start
```

---

You should now see the IPVS table constructed again, according to your configuration in the /etc/ha.d/conf/ldirectord-209.100.100.3.conf file. (Use the ipvsadm -L -n command to view this table.)

To finish this recipe, install the ldirectord program on the backup LVS Director, and then configure the Heartbeat configuration files exactly the same way on both the primary LVS Director and the backup LVS Director. (You can do this manually or use the SystemImager package described in Chapter 5.)

### ***Stateful Failover of the IPVS Table***

As we've just seen, when the primary Director crashes and ldirectord needs to rebuild the IPVS table on the backup Director it can do so because you have placed all of the ipvsadm configuration rules into an ldirectord configuration file. However, the active client connections (the connection tracking records) are not re-created by ldirectord on the backup Director at failover time. All client computer connections to the cluster nodes are lost during a failover using the recipe we have just built.

These connections (entries in the connection tracking table) change rapidly on a heavily loaded cluster as client connections come and go, so we need a method of sending connection tracking records from a primary Director to a backup Director as they change. The LVS programmers developed a technique to do this (replicate the connection tracking table to the backup director) using multicast packets. This technique was originally called the server sync state daemon, and even though it was implemented inside the kernel (the server sync state daemon does not run in userland) the name stuck. To turn on the server sync state daemon, as it is called, inside the kernel run the following command on the primary Director:

---

```
/sbin/ipvsadm --start-daemon master
```

---

Then, on the backup Director, run the command:

---

```
/sbin/ipvsadm --start-daemon backup
```

---

The primary and backup Directors must be able to communicate with each other using multicast packets on multicast address 224.0.0.81 for the master server sync state daemon to announce changes to the connection tracking records, and for the backup server sync state daemon to hear about these changes and insert them into its idle connection tracking table. To find out if your cluster nodes support multicast, see the output of the `ifconfig` command and look for the word `MULTICAST`. It should be present for each interface that will be using multicast; if it isn't you'll need to recompile your kernel and provide support for multicast (see Chapter 3).<sup>23</sup>

**NOTE** *To stop the sync state daemon (either the master or the backup) you can issue the command `ipvsadm --stop-daemon`.*

Once you have issued these commands on the primary and backup Director (you'll need to add these commands to an init script so the system will issue the command each time it boots—see Chapter 1), your primary Director can crash, and then when `ldirectord` rebuilds IPVS table on the backup Director all active connections<sup>24</sup> to cluster nodes will survive the failure of the primary Director.

**NOTE** *The method just described will failover active connections from the primary Director to the backup Director, but it does not failback the active connections when the primary Director has been restored to normal operation. To failover and failback active IPVS connections you will need (as of this writing) to apply a patch to the kernel. See the LVS HOWTO (<http://www.linuxvirtualserver.org>) for details.*

---

<sup>23</sup> You may also be able to specify the interface that the server sync state daemon should use to send or listen for multicast packets using the `ipvsadm` command (see the `ipvsadm` man page); however, early reports of this feature indicate that a bug prevents you from explicitly specifying the interface when you issue these commands.

<sup>24</sup> All connections that have not been idle for longer than the connection tracking timeout period on the backup Director (default is three minutes, as specified in the LVS code).

## **Modifications to Allow Failover to a Real Server Inside the Cluster**

We have finished building our pair of highly available load balancers (with no single point of failure). However, before we leave the topic of failing over the Director as a resource under Heartbeat's control, let's look at how the Director failover works in conjunction with the LVS LocalNode mode.

Because a Linux Enterprise Cluster will have a relatively small number of incoming requests for cluster services (perhaps only one or two for each employee<sup>25</sup>), dedicating a server to the job of load balancing the cluster may be a waste of resources. We can fully utilize the CPU capacity on the Director by making the Director a node inside the cluster; however, *this may reduce the reliability of the cluster and should be avoided, if possible.*<sup>26</sup> When we tell one of the nodes inside the cluster to be the Director, we will also need to tell one of the nodes inside the cluster to be a backup Director if the primary Director fails.

Heartbeat running on the backup Director will start the `ldirector` daemon, move all of the VIP addresses, and re-create the IPVS table in order to properly route client requests for cluster resources if the primary Director fails, as shown in Figure 15-4.

Notice in Figure 15-4 that the backup Director is also replying to client computer requests for cluster services—before the failover the backup Director was just another real server inside the cluster that was running Heartbeat. (In fact, before the failover occurred the primary Director was also replying to its share of the client computer requests that it was processing locally using the LocalNode feature of LVS.)

Although no changes are required on the real servers to inform them of the failover, we will need to tell the backup LVS Director that it should no longer hide the VIP address on its loopback device. Fortunately, Heartbeat does this for us automatically, using the `IPAddr2` script<sup>27</sup> to see if there are hidden loopback devices configured on the backup Director with the same IP address as the VIP address being added. If so, Heartbeat will automatically remove the hidden, conflicting IP address on the loopback device and its associated route in the routing table. It will then add the VIP address on one of the real network interface cards so that the LVS-DR real server can become an LVS-DR Director and service incoming requests for cluster services on this VIP address.

**NOTE** *Heartbeat will remember that it has done this and will restore the loopback addresses to their original values when the primary server is operational again (the VIP address fails back to the primary Heartbeat server).*

---

<sup>25</sup> Compare this to the thousands of requests that are load balanced across large LVS web clusters.

<sup>26</sup> It is easier to reboot real servers than the Director. Real servers can be removed from the cluster one at a time without affecting service availability, but rebooting the Director *may* impact all client computer sessions.

<sup>27</sup> The `IPAddr` script, if you are using ip aliases.

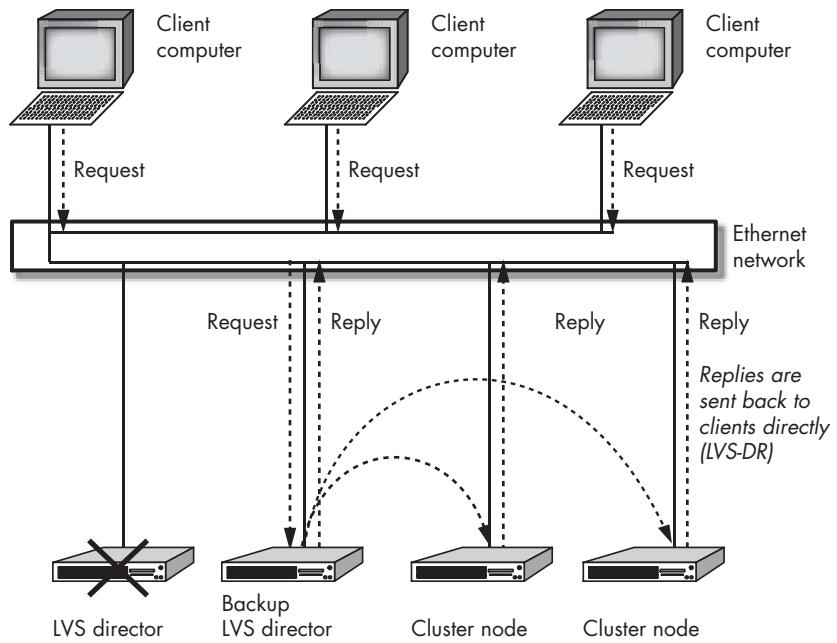


Figure 15-4: Failure of primary Director

## In Conclusion

An enterprise-class cluster must have no single point of failure. It should also know how to automatically detect and remove a failed cluster node. If a node crashes in a Linux Enterprise Cluster, users need only log back on to resume their session. If the Director crashes, a backup Director will take ownership of the load-balancing resource and VIP address that client computers use to connect to the cluster resources.

# 16

## THE NETWORK FILE SYSTEM



The Network File System (NFS) looks like a single, unified filesystem to users and application programs running on the cluster nodes. A key feature of NFS is its ability to share data with all of the cluster nodes while providing a locking mechanism to protect the integrity of the data. To build a cluster that does not require any changes to existing multiuser application programs, this locking mechanism must support the locking methods used by programs that are unaware of the fact that they are running on a cluster node. That is, NFS and its locking mechanism should be transparent to the existing multiuser applications. This transparency of the shared filesystem is one of the defining characteristics of the Linux Enterprise Cluster—applications see the cluster as a single, unified computing resource.

In this chapter, we'll examine what the term *lock* means, what locking methods are normally used on multiuser operating systems, and how NFS allows existing legacy, multiuser applications that are running on cluster nodes to share access to data. (If you do not need to run legacy Unix

applications on your Linux Enterprise Cluster, you can skip over most of the locking information in this chapter.) I'll also introduce you to a few performance issues you will face when using NFS and provide you with an example of the NFS client configuration options you can use on each node inside the cluster.

**NOTE** *In Chapter 20, we'll look at how a database system such as MySQL or Postgres can be used in a cluster environment.*

## Lock Arbitration

When an application program reads data from stable storage (such as a disk drive) and stores a copy of that data in a program variable or system memory, the copy of the data may very soon be rendered inaccurate. Why? Because a second program may, only microseconds later, change the data stored on the disk drive, leaving two different versions of the data: one on disk and one in the program's memory. Which one is the correct version?

To avoid this problem, well-behaved programs will first ask a lock arbitrator to grant them access to a file or a portion of a file before they touch it. They will then read the data into memory, make their changes, and write the data back to stable storage. Data accuracy and consistency are guaranteed by the fact that all programs agree to use the same lock arbitration method before accessing and modifying the data.

Because the operating system does not require a program to acquire a lock before writing data,<sup>1</sup> this method is sometimes called *cooperative*, *advisory*, or *discretionary* locking. These terms suggest that programs know that they need to acquire a lock before they can manipulate data. (We'll use the term *cooperative* in this chapter.)

The cooperative locking method used on a monolithic multiuser server should be the same as the one used by applications that store their data using NFS: the application must first acquire a lock on a file or a portion of a file before reading data and modifying it. The only difference is that this lock arbitration method must work on multiple systems that share access to the same data.

## The Lock Arbitrator

We can divide cooperative lock arbitration into three categories based on who or what is acting as the lock arbitrator:

### Kernel lock arbitration

The program asks the kernel for a lock on a particular file or a portion of a file. This is perhaps the most common method used by programmers when developing applications such as an order-entry system in a typical multiuser environment such as Linux or Unix.

---

<sup>1</sup> Except in the case of mandatory locks.

## File lock arbitration

The program creates a new file called a *lock file* or a *dotlock file* on stable storage to indicate that it would like exclusive access to a data file. Programs that access the same data must look for this file or examine its contents.<sup>2</sup> This method is typically used in order to avoid subtle differences in kernel lock arbitration implementations (in different versions of Unix for example) when programmers want to develop software that will run on a variety of different operating systems.

## External lock arbitration daemon

The program asks a daemon to track lock and unlock requests. This type of locking is normally used for shared storage. Examples of this type of lock arbitration include sophisticated database applications, distributed lock managers (that can run on more than one node at the same time and share lock information), and the Network Lock Manager (NLM) used in NFSv3, which will be discussed shortly.

**NOTE** Additional lock arbitration methods for cluster file systems are provided in Appendix E.

Our cluster environment should use a cooperative lock arbitration method that works in conjunction with shared storage while still honoring the classic Unix or Linux kernel lock arbitration requests (so we do not have to rewrite all of the user applications that will share data in the cluster). Fortunately, Linux supports a shared storage external lock arbitration daemon called `lockd`. But before we get into the details of NFS and `lockd`, let's more closely examine the kernel lock arbitration methods used by existing multiuser applications.

## The Existing Kernel Lock Arbitration Methods

The three most common cooperative kernel lock arbitration methods available on Linux are BSD `flock`, System V `lockf`, and Posix `fcntl`.

### BSD `flock`

This is considered an antiquated method of locking files because it only supports locking an entire file, not a range of bytes (called a *byte offset*) within a file. There are two types of flocks: *shared* and *exclusive*. As their names imply, many processes may hold a shared lock on one file, but only one process may hold an exclusive lock. (A file cannot be locked with both shared and exclusive locks at the same time.) Because this method only supports locking the entire file, your existing multiuser applications are probably not using this locking mechanism to arbitrate access to shared user data. We'll discuss this further in "The Network Lock Manager (NLM)."

### System V `lockf`

The System V locking method is called `lockf`. On a Linux system, `lockf` is really just an interface to the Posix `fcntl` method discussed next.

---

<sup>2</sup> The PID of the process that created the dotlock file is usually placed into the file.

## Posix fcntl

The Posix-compliant fcntl system call used on Linux does a variety of things, but for the moment we are only interested in the fact that it allows a program to lock a byte-range portion of a file.<sup>3</sup> Posix fcnlt locking supports the same two types of locks as the BSD flock method but uses the terms *read* and *write* instead of *shared* and *exclusive*. Multiple read locks are allowed at the same time, but when a program asks the kernel for a write lock, no other programs may hold either type of lock (read or write) *for the same range of bytes within the file*.

When using Posix fcnlt, the programmer decides what will happen when a lock is denied or blocked by the kernel by deciding if the program should wait for the call to fcntl to succeed (wait for the lock to be granted). If the choice is to wait, the fcntl call does not return and the kernel places the request<sup>4</sup> into a blocked queue. When the program that was holding the lock releases it and the reason for the blocked lock no longer exists, the kernel will reply to the fcntl request, waking up the program from a “sleep” state with the good news that its request for a lock has been granted.

For example, three processes may all hold a Posix fcntl read lock on the byte range 1,000–1,050 of the same file at the same time. But when a fourth process comes along and wants to acquire a Posix fcntl write lock to the same range of bytes, it will be placed into the blocked queue. As long as any one of the original three programs continues to hold its read lock, the fourth program’s request for a write lock will remain in the blocked queue. However, if a fifth program asks for a Posix fcntl read lock, it will be granted it immediately. Now the fourth program is still waiting for its write lock and it may have to wait forever if new processes keep coming along and acquiring new read locks. This complexity makes it more difficult to write a program that uses both Posix fcntl read and Posix fcntl write locks.

Also of note, when considering Posix fcntl locks:

1. If a process holds a Posix fcntl lock and forks a child process, the child process (running under a new process id or pid) will not inherit the lock from its parent.
2. A program that holds a lock may ask the kernel for this same lock a second time and the kernel will “grant” it again without considering this a conflict. (More about this in a moment when we discuss the Network Lock Manager and lockd.)
3. The Linux kernel currently does not check for collisions with Posix fcntl byte-range locks and BSD file flocks. The two locking methods do not know about each other.

---

<sup>3</sup> If the byte range covers the entire file, then this is equivalent to a whole file lock.

<sup>4</sup> Identified by the process id of the calling program as well as the file and byte range to be locked.

**NOTE**

*There are at least three additional kernel lock arbitration methods available under Linux: whole file leases, share modes (similar to Windows share modes), and mandatory locks. If your application relies on one of these methods for lock arbitration, you must use NFS version 4.*

Now that we've talked about the existing kernel lock arbitration methods that work on a single, monolithic server, let's examine a locking method that allows more than one server to share locking information: the Network Lock Manager.

## The Network Lock Manager (NLM)

The NFS was originally conceived and built to manage distributed access to a single data storage device, with separate programs added to provide a cooperative lock arbitration method. Because the filesystem and its locking method were implemented separately, the original NFS developers felt that they were providing a generic means of network lock arbitration that could be used in conjunction with any network file system. However, the network lock manager (NLM), as it became known, has only been widely used as a lock arbitration method by NFSv3 servers and NFS version 3 clients. The NFSv4 protocol does not use a separate daemon or protocol for locking, but many of the concepts are the same, so we'll introduce NFS locking by discussing the NFSv3 NLM.

The NLM consists of two daemons called `statd` and `lockd`. Both daemons are supposed to be running all the time on the NFS server and on all NFS clients to ensure that everyone agrees upon what is locked and which programs or processes own the locks. The `statd` daemon running on both the client and the server keeps a list of the hosts that are currently holding or granting locks (NFS clients, if the machine is acting as an NFS server, and NFS servers, if the machine is an NFS client<sup>5</sup>). The `lockd` daemon on the NFS client is in charge of making a lock request over the network by talking to the `lockd` daemon running on the NFS server.

Let's examine these two daemons more closely.

### `statd`

In a cluster, `statd`, also called `rpc.statd`, runs on each cluster node in case the node crashes while it holds a lock on the NFS server. If this happens, the `rpc.statd` program on the cluster node will notify the NFS server when the cluster node is rebooted that it is now operational again. `statd` on the NFS client knows that it should do this, because it writes the name of each NFS server to a local disk drive the first time a process running on the cluster node tries to lock a file on the NFS server. When the NFS server receives this notification after an NFS client reboots, it assumes all of the processes that were holding locks from this cluster node (or NFS client) are no longer running, and it releases them.

---

<sup>5</sup> Actually, it always keeps a list of both if the machine is both an NFS client and an NFS server.

**NOTE** *statd or rpc.statd is sometimes described as an implementation of a network status monitor (NSM).*

#### **lockd**

The Linux kernel knows when a Posix fcntl lock request is for a file or portion of a file stored on an NFS server, because the file is stored on an NFS-mounted filesystem. If the Linux machine is an NFS client, the kernel knows it needs to pass the lock request to the locally running lockd daemon.<sup>6</sup> The lockd daemon on the NFS client then contacts the lockd daemon running on the NFS server and requests the lock. The lockd daemon on the NFS server then asks its own kernel for the lock on behalf of the NFS client.

If the same program or process asks for the same lock<sup>7</sup> more than once, it will be granted the lock each time; the kernel doesn't consider this to be a violation of the lock. Before lockd became a part of the kernel, the ability for a single process to be granted the same lock more than once meant that the lockd daemon had to track lock requests that it had made on behalf of the NFS clients and deny any conflicts before asking the kernel for the lock. Were it not to do so, lockd could unknowingly ask for, and be granted, the same lock twice on behalf of two different NFS clients.

## **NLM and Kernel Lock Arbitration**

Let's look at how NLM works in conjunction with the kernel lock arbitration methods on the NFS clients (the cluster nodes). Remember that we want to support existing multiuser applications that use one or more of the existing kernel lock arbitration methods so that we do not have to rewrite user applications to run them on the cluster.

### **NLM and Kernel BSD flock**

The Linux kernel currently does not pass BSD flock requests for whole file locks to the NLM. As such, this method of file locking will not work on the Linux Enterprise Cluster when access to shared data is required across all cluster nodes.

Because BSD flocks can only lock whole files, your existing multiuser applications aren't likely to use them to share user data. BSD flocks are more commonly used by applications that fork child processes in order to prevent the child processes from doing things that would cause conflicts with each other. For example, the LPRng printing system creates child processes for sending print jobs, and the child processes create temporary control files in the `/var/spool/lpd` directory to ensure that print jobs are sent in the correct order.

---

<sup>6</sup> The lockd daemon actually runs *inside* the kernel as a kernel thread on Linux and is called `kllockd`, but for our purposes, we'll talk about it as if it were outside the kernel.

<sup>7</sup> At least this is true for Posix fcntl locks in NFS version 3.

In fact, most daemons that use BSD flocks create temporary files in a subdirectory underneath the /var directory, so when you build a Linux Enterprise Cluster you should not use a /var directory over NFS. By using a local /var directory on each cluster node, you can continue to run existing applications that use BSD flocks for temporary files.<sup>8</sup>

**NOTE** *In the Linux Enterprise Cluster, NFS is used to share user data, not operating system files.*

## **NLM and Kernel System V lockf**

The System V lockf method is a wrapper around the Posix fcntl method of lock arbitration. See the next section for details.

## **NLM and Kernel Posix fcntl**

When an application running on an NFS client issues an fcntl lock operation on data that is mounted over an NFS filesystem, several things happen:

1. Any file data or file attribute information stored in the local NFS client computer's cache is flushed.
2. The NFS server is contacted<sup>9</sup> to check the attributes of the file again, in part to make sure that the permissions and access mode settings on the file still allow the process to gain access to the file.
3. lockd on the NFS server is contacted to determine if the lock will be granted.
4. If this is the first lock made by the particular NFS client, the statd daemon on both the NFS client and the NFS server records the fact that this client has made a lock request.

Posix fcntl locks that use the NLM (for files stored using NFS) are very slow when compared to the ones granted by the kernel. Also notice that due to the way the NFS client cache works, the only way to be sure that the data inside your application program's memory is the same as the data stored on the NFS server is to lock the data before reading it. If your application reads the data without locking it, another program can modify the data stored on the NFS server and render your copy (which is stored in your program's memory) inaccurate.

We'll discuss the NLM performance issues more shortly.

---

<sup>8</sup> When you use the LPRng printing system, printer arbitration is done on a single print spool server (called the cluster node manager in a Linux Enterprise Cluster), so lock arbitration between child LPRng printing processes running on the cluster nodes is not needed. We'll discuss LPRng printing in more detail in Chapter 19.

<sup>9</sup> Using the GETATTR call.

## NFS and File Lock (dotlock) Arbitration

Recall from our discussion earlier in this chapter that applications that use the lock arbitration method (called *file lock arbitration* or *dotlocking*) check for a dotlock file<sup>10</sup> as a means of arbitrating access to a data file. Because NFS clients are normally configured to use locally cached data that may not be the same as the data stored on the NFS server, the dotlock file arbitration method must be implemented to work over NFS.

In order to use dotlock file arbitration, the programmer must use a method of creating dotlock files that ensures that the dotlock file has been stored on the NFS server and is not just sitting in the local NFS client's cache. This is accomplished (as described on the "open" man page) by creating the dotlock file, linking to it, and then checking to make sure that the link command worked or that the count of the number of links associated with the dotlock file has increased to 2.

Apache, sendmail, and LPRng all use dotlock file lock arbitration. However, none of these use dotlock arbitration to share *writable* data in a normal Linux Enterprise Cluster configuration.

**NOTE** A *CGI program running under Apache on a cluster node may share access to writable data with a CGI program running on another cluster node, but the CGI programs can implement their own locking method independent of the dotlock method used by Apache and thus avoid the complexity of implementing dotlock file locking over NFS. For an example, search CPAN<sup>11</sup> for "File::NFSLock."*

## Finding the Locks Held by the Linux Kernel

We will now leave the theoretical discussion of the NLM and its interaction with the kernel lock arbitration methods and discuss system administration and maintenance activities as they relate to kernel and NLM lock arbitration.

To see what Posix fcntl and BSD flocks are currently granted by the kernel, examine the contents of the /proc/locks file. Although you can do this on any Linux system, this technique is perhaps most useful for troubleshooting locking problems on cluster nodes that have NFS-mounted data.

To view the current locks, first cat the /proc/locks kernel file:

---

```
#cat /proc/locks
 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
1: POSIX ADVISORY WRITE 14555 21:06:69890 0 EOF cd4f87f0 c02d9fc8 cd4f8570 00000000 cd4f87fc
2: POSIX ADVISORY WRITE 14005 21:06:69889 0 EOF cd4f87f0 cd4f873c cd4f8ca0 00000000 cd4f87fc
3: POSIX ADVISORY WRITE 8508 21:07:56232 0 EOF cd4f8f7c cd4f873c cd4f8908 00000000 cd4f8f88
4: FLOCK ADVISORY WRITE 12435 22:01:2387398 0 EOF cd4f8904 cd4f8f80 cd4f8ca0 00000000 cd4f8910
5: FLOCK ADVISORY WRITE 12362 22:01:1831448 0 EOF cd4f8c9c cd4f8908 cd4f83a4 00000000 cd4f8ca8
```

---

<sup>10</sup> Or they check the contents of the dotlock file to see if a conflicting PID already owns the dotlock file.

<sup>11</sup> See Chapter 15 for how to search and download CPAN modules.

**NOTE** I've added column numbers to this output to help make the following explanation easier to understand.

Column 1 displays the lock number and column 2 displays the locking mechanism (either flock or Posix fcntl). Column 3 contains the word `advisory` or `mandatory`; column 4 is `read` or `write`; and column 5 is the PID (process ID) of the process that owns the lock. Columns 6 and 7 are the device major and minor number of the device that holds the lock,<sup>12</sup> while column 8 is the inode number of the file that is locked. Columns 9 and 10 are the starting and ending byte range of the lock, and the remaining columns are internal block addresses used by the kernel to identify the lock.

To learn more about a particular lock, use `lsof` to examine the process that holds it. For example, consider this entry from above:

---

```
1: POSIX ADVISORY WRITE 14555 21:06:69890 0 EOF cd4f87f0 c02d9fc8 cd4f8570 00000000 cd4f87fc
```

---

The PID is 14555, and the inode number is 69890. (See boldface text.) Using `lsof`, we can examine this PID further to see which process and file it represents:

---

```
#lsof -p 14555
```

---

The output of this command looks like this:

---

COMMAND	PID	USER	FD	TYPE	DEVICE	SIZE	NODE	NAME
lockdemo	14555	root	cwd	DIR	33,6	2048	32799	/root
lockdemo	14555	root	rtd	DIR	33,6	2048	2	/
lockdemo	14555	root	txt	REG	33,6	14644	69909	/tmp/lockdemo
lockdemo	14555	root	mem	REG	33,6	89547	22808	/lib/ld-2.2.5.so
lockdemo	14555	root	mem	REG	33,6	1401027	26741	/lib/i686/libc-2.2.5.so
lockdemo	14555	root	0u	CHR	136,3		5	/dev/pts/3
lockdemo	14555	root	1u	CHR	136,3		5	/dev/pts/3
lockdemo	14555	root	2u	CHR	136,3		5	/dev/pts/3
lockdemo	14555	root	3u	REG	33,6	1024	<b>69890</b>	/tmp/mydata

---

The last line in this `lsof` report tells us that the inode 69890 is called `/tmp/mydata` and that the name of the command or program that is locking this file is called `lockdemo`.<sup>13</sup>

To confirm that this process does, in fact, have the file open, enter:

---

```
#fuser /tmp/mydata
```

---

<sup>12</sup>You can see the device major and minor number of a filesystem by examining the device file associated with the device (`ls -l /dev/hda2`, for example).

<sup>13</sup>The source code for this program is currently available online at <http://www.ecst.csuchico.edu/~beej/guide/ipc/flock.html>.

The output gives us the file name followed by any PIDs that have the file open:

---

```
/tmp/mydata: 14555
```

---

To learn more about the locking methods used in NFSv3, see RFC 1813, and for more information about the locking methods used in NFSv4, see RFC 3530. (Also see the Connectathon lock test program on the “Connectathon Test Suites” link at <http://www.connectathon.org>.)

## Performance Issues with NFS—Bottlenecks and Perceptions

Now that we’ve examined the locking issues required to build a cluster filesystem that can support existing multiuser applications, we need to examine how well the NLM and NFS perform. The two basic areas of concern when analyzing the performance of a NFS are:

1. Where are the performance bottlenecks?
2. How will the end-user perceive the overall NFS performance?

Like an engineer building a suspension bridge, a cluster architect wants to know how NFS will bear the load of the traffic that it will carry. As such, we need to analyze the amount of traffic and know where the stress points or bottlenecks are likely to occur. We also need to know the difference between the raw numbers used to describe the speed of an NAS device and the end-user’s perception of how well the system as a whole performs.

We’ll first look at the size of the pipes used to send and receive data over the network. As shown in Figure 16-1, the size of the network “pipe” connecting the NAS server to the Ethernet backbone will normally support a gigabit per second (Gbps). Also, as shown in Figure 16-1, the cluster node has two network connections. The first is a 100 Mbps pipe to the network that connects the cluster node’s real IP address (RIP) to the Director’s IP (DIP), called the D/RIP network in LVS terms. The second network connection is used to connect the cluster node to a network that is dedicated to NFS traffic (labeled the “NFS Network” in Figure 16-1).

The Gigabit pipe on the NAS server and the switched Ethernet backbone (which can be hundreds of Gbps) for the NFS network are not likely to be the first stress points of performance. In fact, most NAS vendors support multiple Gigabit Ethernet pipes to the Ethernet backbone. Cluster nodes can also be built with multiple Gigabit pipes to the Ethernet backbone, though most multiuser applications running in a cluster environment will not saturate a single 100 Mbps pipe. (Remember that one of the benefits of using a cluster is that more nodes can always be added if too many instances of the same application or service on a single node are competing for access to the 100 Mbps pipe to the Ethernet backbone.)<sup>14</sup>

---

<sup>14</sup> The cost to add additional nodes may be less than the cost to build multiple Gigabit pipes to each cluster node.

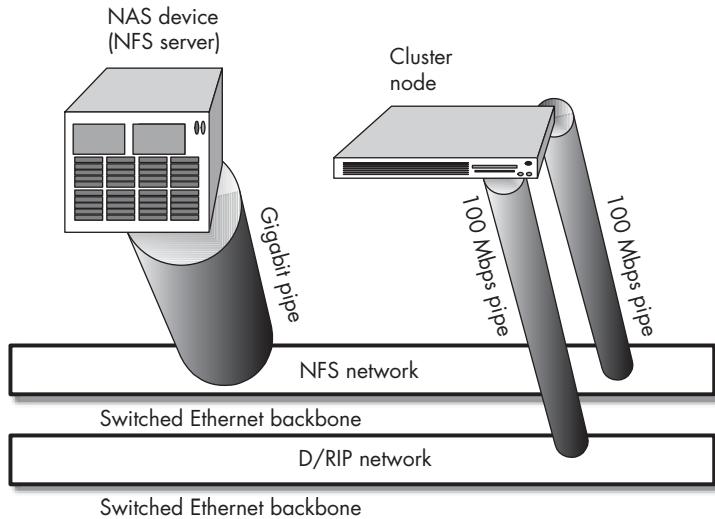


Figure 16-1: Ethernet performance bottlenecks

Where, then, is the likely stress point in a shared filesystem? To answer this question, we turn our attention to the end-user's perception of the overall NFS performance.

**NOTE** *In the following discussion, the term database refers to flat files or an indexed sequential access method (ISAM) database used by legacy applications. For a discussion of relational and object databases (MySQL, Postgres, ZODB), see Chapter 20.*

### **Single Transactions and User Perception of NFS Performance**

For example, consider users who enter orders into a system and who need to run reports against this data. Here's what happens: A customer calls, and a customer service agent pulls up the customer's record. The agent then starts a new order and types in information. Up to this point, the difference between clustered NFS disk performance at 100 Mbps and locally attached storage is negligible.

The agent searches for a desired item (product, airline seat, classified ad, and so forth). If the application can find the data the agent is looking for using a key such as a product number or a flight number, the performance hit created by the NFS overhead is negligible because the application does not need to bring a significant amount of information over the network to complete the search.

Next, the agent selects the item to modify. Now, a Posix fcntl lock operation ensures that another agent cannot select the same item or product. As with the above, the NFS locking overhead on a single byte range within a file is not likely to create a noticeable performance difference as far as the agent is concerned.

Once the agent has picked his or her item, the database quantity is updated to reflect the change in inventory, the lock is released, and perhaps a new record is added to another database for the line item in the order. This process is repeated until the order is complete and a final calculation is performed on the order (total amount, tax, and so on). Again, this calculation is likely to be faster on a cluster node than it would be on a monolithic server because more CPU time is available to make the calculation.

Given this scenario, or one like it, there is little chance that the extra overhead imposed by NFS will impact the customer service agent's perception of the cluster's performance.<sup>15</sup> But what about users running reports?

### ***Multiple Transactions and User Perception of NFS Performance***

The situation is different for the user running reports or multiple transactions in a batch. This user's perception of the cluster's performance is likely to depend upon how quickly the NAS server can perform I/O operations and how many lock and GETATTR operations are required for each read or write transaction.

**NOTE** *Historically, one drawback to using NAS servers (and perhaps a contributing factor for the emergence of storage area networks) was the CPU overhead associated with packetizing and de-packetizing filesystem operations so they could be transmitted over the network. However, as inexpensive CPUs push past the multi-GHz speed barrier, the packetizing and de-packetizing performance overhead fades in importance. And, in a cluster environment, you can simply add more nodes to increase available CPU cycles if more are needed to packetize and de-packetize Network File System operations (of course, this does not mean that adding additional nodes will increase the speed of NFS by itself).*

The speed of your NAS server (the number of I/O operations per second) will probably be determined by your budget. Discussing all of the issues involved with selecting the best NAS hardware is outside the scope of this book, but one of the key performance numbers you'll want to use when making your decision is how many NFS operations the NAS server can perform per second. The NAS server performance is likely to become the most expensive performance bottleneck to fix in your cluster.<sup>16</sup>

The second performance stress point—the quantity of lock and GETATTR operations—can be managed by careful system administration and good programming practices.

---

<sup>15</sup> Unless, of course, the NAS server is severely overloaded. See Chapter 18.

<sup>16</sup> At the time of this writing, a top-of-the-line NAS server can perform a little over 30,000 I/O operations per second.

## Managing Lock and GETATTR Operations in a Cluster Environment

You've seen lock operations; GETATTR operations are simply file operations that examine the attributes of a file (who owns it, what time it was last modified, and so forth<sup>17</sup>). Lock and GETATTR operations require the NFS client to communicate with the NAS server using the get attribute or GETATTR call of the NFS protocol. This brings us to the topic of attribute caching—a method you can use to improve the performance of GETATTR calls.

## Managing Attribute Caching

The file metadata information that NFS clients read using the NFS GETATTR call can be read from a cached copy of this information stored on the NFS client instead of from the NFS server.

**NOTE** *Some applications such as Oracle's 9iRAC technology require you to disable attribute caching, so the noac option discussed next may not be available to improve NFS client performance under all circumstances.*

To see the difference in performance between GETATTR operations that use the NFS server and GETATTR calls that use cached attribute information, mount the NFS filesystem with the `noac` option on the NFS client and then run a typical user application with the following command (also on the NFS client):

---

```
#watch nfsstat -c
```

---

Watch the value for “getattr” on this report increase as the application performs GETATTR operations to the NFS server. Now remove the `noac` option, and run the database report again. If the change in value for the `getattr` number is significantly lower this time, you should leave attribute caching turned on.

**NOTE** *Most applications with I/O-intensive workloads will perform approximately 40 percent more slowly when attribute caching is turned off.*

The performance of the attribute caching is also affected by the timeout values you specify for the attribute cache. See the options `acregmin`, `acregmax`, `acdirmmin`, `acdirmmax`, and `actimeo` on the `nfs` man page.

**NOTE** *Even with attribute caching turned on, the NFS client (cluster node) will remove attribute information from its cache each time the file is closed. This is called close-to-open or cto cache consistency. See the `nocto` option on the `nfs` man page.*

---

<sup>17</sup> All inode information about the file.

# Managing Interactive User Applications and Batch Jobs in a Cluster Environment

In the previous discussion, you saw two types of users: interactive users who perform many small I/O operations from the keyboard, and users who run batch jobs and database reports.<sup>18</sup> Each type of user places different demands on the cluster node CPUs and on the NAS server. The first type wants to use the CPU and the NAS server briefly and intermittently during normal business hours, while the second wants to grab as many CPU cycles and NAS I/O operations as possible for long periods of time. To satisfy both types, the cluster administrator can isolate each type of user application on two different cluster nodes.

**NOTE** *Historically, on a monolithic server, the Unix system administrator reduced the impact of batch jobs (CPU- and I/O-hungry programs) by running them at night or by starting them with the nice command. The nice command allows the system administrator to lower the priority of batch programs and database reports relative to other processes running on the system by reducing the amount of the CPU time slice they could grab. On a Linux Enterprise Cluster, the system administrator can more effectively accomplish this by allocating a portion of the cluster nodes to users running reports and allocating another portion of the cluster to interactive users (because this adds to the complexity of the cluster, it should be avoided if possible).*

## Run Batch Jobs Outside the Cluster

The NAS server, however, is still a single point of contention for these two types of applications, so you'll want to purchase the fastest NAS server you can afford. Another way to reduce the contention for the NAS server, if your budget doesn't allow you to purchase one that is fast enough to handle both types of users at the same time, is to use the rsync utility to make a nightly copy of the data stored on the NAS server and place this copy of the data onto local storage on an old server. You can then run batch jobs that do not require up-to-the-minute data on this server (month-end accounting reports can use the rsync snapshot of yesterday's data, for example).

## Use Multiple NAS Servers

Of course, you can also eliminate the NAS server as a single point of contention by using more than one NAS server and spreading your data across two or more NAS servers. For example, the central data center for a multi-warehouse facility could store the data for each warehouse on a separate NAS server.

However, if your budget allows, the easiest configuration for the system administrator to build and maintain is the one that lets users store all of their data in one place: on a single NAS server. And how fast does the NAS server

---

<sup>18</sup> Possibly batch jobs that run outside of the normal business hours.

need to be? In the next two sections, we will look at a few of the methods you can use to evaluate a NAS server when it will be used to support the cluster nodes running legacy applications.

## Measuring NFS Latency

To find out how long your application program is idle and waiting for the network and the NAS server, you can run a simulation<sup>19</sup> or a batch program preceded by the `time` command. For example, let's say you have a batch program called `mytest` in the `/usr/local/bin` directory that will perform read and write operations using data stored on the NAS server.<sup>20</sup> To test its performance, start it from a shell prompt with the command:

---

```
#time /usr/local/bin/mytest
```

---

When the `mytest` program finishes executing you will receive a report such as the following:

---

real	0m7.282s
user	0m1.080s
sys	0m0.060s

---

This report indicates the amount of wall clock time (real) the `mytest` program took to complete, followed by the amount of CPU user and CPU system time it used. Add the CPU user and CPU system time together, and then subtract them from the first number, the real time, to arrive at a rough estimate for the latency<sup>21</sup> of the network and the NAS server.

**NOTE** *This method will only work for programs (batch jobs) that perform the same or similar file-system operations throughout the length of their run (for example, a database update that requires updating the same field in all of the database records that is performed inside of a loop). The method I've just described is useful for arriving at the average I/O requirements of the entire job, not the peak I/O requirements of the job at a given point in time.*

You can use this measurement as one of the criteria for evaluating NAS servers, and also use it to study the feasibility of using a cluster to replace your monolithic Unix server. Keep in mind, however, that in a cluster, the speed and availability of the CPU may be much greater than on the monolithic server.<sup>22</sup> Thus, even if the latency of the NAS server is greater than locally attached storage, the cluster may still outperform the monolithic box.

---

<sup>19</sup> The Expect programming language is useful for simulating keyboard input for an application.

<sup>20</sup> You can use a Linux box as an NFS server for testing purposes. Alternatively, if your existing data resides on a monolithic Unix server, you can simply use your monolithic server as an NFS server for testing purposes. See the discussion of the `async` option later in this chapter.

<sup>21</sup> The time penalty for doing filesystem operations over the network to a shared storage device.

<sup>22</sup> Because processor speeds increase each year, I'm making a big assumption here that your monolithic server is a capital expenditure that your organization keeps in production for several years—by the end of its life cycle, a single processor on the monolithic server is likely to be much slower than the processor on a new and inexpensive cluster node.

## Measuring Total I/O Operations

The second measurement you'll need to make before you shop for a NAS solution is the total number of I/O operations per second that your applications will require at peak processing times. To find this number, use a Linux machine that is not doing anything else (preferably the hardware you will use for your cluster nodes) and run the `nfsstat -c` command. Sample output for this command looks like this:

---

```
Client rpc stats:
calls      retrans   authrefresh
667813156   38          0

Client nfs v2:
null      getattr  setattr   root      lookup      readlink
0        0% 0      0% 0      0% 0      0% 0      0% 0      0%
read      wrcache  write     create     remove     rename
0        0% 0      0% 0      0% 0      0% 0      0% 0      0%
link      symlink  mkdir     rmdir     readdir   fsstat
0        0% 0      0% 0      0% 0      0% 0      0% 0      0%

Client nfs v3:
null      getattr  setattr   lookup     access     readlink
0        0% 129765080 0% 523295 0% 10521226 1% 86507757 0% 46448 0%
read      write    create     mkdir     symlink   mknod
314127903 2% 23687043 4% 748925 0% 17      0% 225    0% 0      0%
remove    rmdir    rename    link     readdir   readdirplus
372452 0% 17      0% 390726 0% 4155 0% 563968 0% 0      0%
fsstat    fsinfo   pathconf  commit
2        0% 2      0% 0      0% 12997359 2%
```

---

Assuming you have configured this machine to use NFS version 3 (a sample `/etc/fstab` entry to do this is provided later in this chapter), you can add up all of the values listed under the `Client nfs v3:` section of this report and then run your sample application using the `time` command I just described. When the application finishes running, use the `nfsstat -c` command again, and add up the total number of NFS calls that were made. You can then simply subtract the first number from the second number to arrive at the total number of NFS operations your application performed. Divide this number by the total number of elapsed seconds your application ran (the first number returned by the `time` command) to determine the average NFS I/O operations per second your application uses. Now multiply this number by the total number of applications that will be running at peak system load (order processing deadline for example) to arrive at a very rough estimate of the total number of I/O operations you'll need on your NAS server. (For a more accurate picture of the total number of I/O operations per second you will require, perform the same analysis using a batch job, and add this number to your total as well.)

Now let's examine a few of the options you have for configuring your NFS environment to achieve the best possible performance and reliability.

## Achieving the Best NAS Performance Possible

Now that you know a few of the considerations you'll need to make before moving your legacy applications on to a cluster filesystem that uses the NFS protocol, note a few of the things you can do to make your cluster filesystem perform optimally:

- Dedicate a network to isolate NFS traffic.
- Use a high-quality NAS server from a NAS vendor (these NAS servers use nonvolatile RAM to commit NFS write operations as quickly as possible).
- Play with the read and write sizes used on the NFS client (see “wsize” and “rsize” on the mount man page). When using NFS over TCP (discussed in a moment), the recommended read and write size is 32K. (The read and write size should always be larger than the NFS client's page size, which is usually 4K.) *Sending larger read and write requests from an NFS client can significantly reduce the latency of NFS operations; however, you can't always control the size of NFS packets by simply changing these numbers (the application program that uses NFS may also need to be modified).*
- Use specialized networking techniques from your NAS vendor (such as trunking multiple network connections) to remove network performance bottlenecks.<sup>23</sup>

**NOTE** *In an effort to boost the NFS server's performance, the current default Linux NFS server configuration uses asynchronous NFS. This lets a Linux NFS server (acting as an NAS device) commit an NFS write operation without actually placing the data on the disk drive. This means if the Linux NFS server crashes while under heavy load, severe data corruption is possible. For the sake of data integrity, then, you should not use async NFS in production. An inexpensive Linux box using asynchronous NFS will, however, give you some feel for how well an NAS system performs.<sup>24</sup>*

Don't assume you need the added expense of a GigE network for all of your cluster nodes until you have tested your application using a high-quality NAS server.

In Chapter 18, we'll return to the topic of NFS performance when we discuss how to use the Ganglia monitoring package.

---

<sup>23</sup> Although, as we've already discussed the network is not likely to become a bottleneck for most applications.

<sup>24</sup> Though async NFS on a Linux server may even *outperform* a NAS device.

## NFS Client Configuration Options

Now let's examine the NFS mount options you can use on the cluster nodes.

- tcp** The Linux NFS client support for TCP helps to improve NFS performance when network load would otherwise force an NFS client using UDP to resend its network packets. Because UDP performance is better when network load is light, and because NFS is supposed to be a connectionless protocol, it is tempting to use UDP, but TCP is more efficient when you need it most—when the system is under heavy load. To specify `tcp` on the NFS client, use mount option `tcp`.
- vers** To make sure you are using NFS version 3 and not version 2, you should also specify this in your mount options on the NFS clients. This is mount option `vers=3`.
- hard** To continue to retry the NFS operation and cause the system to not return an error to the user application performing the I/O, use mount option `hard`.
- fg** To prevent the system from booting when it cannot mount the filesystem, use mount option `fg`.

### NFS DIRECT I/O

The kernel option that was introduced in kernel version 2.4.22, called `CONFIG_NFS_DIRECTIO`, gives the ability to read and write directly to the NFS server to applications that use the `O_DIRECT` `open()` flag. The NFS client, in other words, does not cache any data. Using this kernel option is useful in cases where applications implement their own cache coherency methods that would conflict with the normal NFS client's caching techniques, or when large data files (such as streaming audio files) would not benefit from the use of an NFS client's cache.

## Putting It All Together

The NFS mount options just described can be used in an `/etc/fstab` entry that looks like this:

---

```
nasserver:/clusterdata /clusterdata nfs rw, hard, nointr, tcp,  
vers=3, rsize=32k, wsize=32k, fg 0 0
```

---

This single line says to mount the filesystem called `/clusterdata` from the NAS server host named `nasserver` on the mount point called `/clusterdata`. The options that follow on this line specify: the type of filesystem is NFS (`nfs`), the type of access to the data is both read and write access (`rw`), the cluster node should retry failed NFS operations indefinitely (`hard`), the NFS operations cannot be interrupted (`nointr`), all NFS calls should use the TCP protocol instead of UDP (`tcp`), NFS version 3 should always be used (`vers=3`),

the read (`rsize`) and write (`wsize`) size of NFS operations are 32K to improve performance, the system will not boot when the filesystem cannot be mounted (`fg`), and dump program does not need to back up the filesystem (`0`) and the `fsck`<sup>25</sup> program does not need to check the file system at boot time (`0`).

## Developing NFS

The NFS protocol continues to evolve and develop with strong support from the industry. One example of this is the continuing effort that was started with NFSv4<sup>26</sup> file delegation to remove the NFS server performance bottleneck by distributing file I/O operations to NFS clients. With NFSv4, delegation of the applications running on an NFS client can lock different byte-range portions of the file without creating any additional network traffic to the NAS server. This has led to a development effort called NFS Extensions for Parallel Storage. The goal of this effort is to build a highly available filesystem with no single point of failure by allowing NFS clients to share file state information and file data without requiring communication with a single NAS device for each filesystem operation.

For more information about the NFS Extensions for Parallel Storage, see <http://www.citi.umich.edu/NEPS>.

## Additional Starting Points for Information on Linux and NFS

In this chapter, we have only scratched the surface of a complex topic: how to configure and use a NAS server. The focus has mostly been on lock information and basic performance issues to get you started thinking about the issues involved when you convert from a monolithic server to a Linux Enterprise Cluster that will run legacy multiuser applications. Here are additional resources to assist you:

- When building an NFS client on Linux, begin by reading the Network Appliance Technical Article “TR3183.”
- The home page for the open source version of NFS is <http://nfs.sourceforge.net>.
- Linux NFS performance tuning is discussed at <http://linux-perf.nl.linux.org>.
- Also, for Linux NFS performance tuning, see the “Linux NFS-HOWTO” at <http://nfs.sourceforge.net/nfs-howto/performance.html>.

---

<sup>25</sup> A filesystem sanity check can be performed by the `fsck` program on all locally attached storage at system boot time.

<sup>26</sup> NFSv4 (see RFC 3530) will provide a much better security model and allow for better interoperability with CIFS. NFSv4 will also provide better handling of client crashes and robust request ordering semantics as well as a facility for mandatory file locking.

**NOTE** See especially the online transaction processing (OLTP) benchmark, which was patterned after the widely used TCP-C benchmark.

- An excellent discussion of the methods for troubleshooting NFS problems is available in the book *Linux NFS and Automounter Administration* by Erez Zadok.

## In Conclusion

If you are deploying a new application, your important data—the data that is modified often by the end-users—will be stored in a database such as Postgres, MySQL, or Oracle. The cluster nodes will rely on the database server (outside the cluster) to arbitrate write access to the data, and the locking issues I've been discussing in this chapter do not apply to you.<sup>27</sup> However, if you need to run a legacy application that was originally written for a monolithic Unix server, you can use NFS to hide (from the application programs) the fact that the data is now being shared by multiple nodes inside the cluster. The legacy multiuser applications will acquire locks and write data to stable storage the way they have always done without even knowing that they are running inside of a cluster—thus the cluster, from the inside as well as from the outside, appears to be a single, unified computing resource.

---

<sup>27</sup> To build a highly available SQL server, you can use the Heartbeat package and a shared *storage* device rather than a shared filesystem. The shared storage device (a shared SCSI bus, or SAN storage) is only mounted on one server at a time—the server that owns the SQL “resource”—so you don’t need a shared filesystem.

# PART IV

## MAINTENANCE AND MONITORING





# 17

## THE SIMPLE NETWORK MANAGEMENT PROTOCOL AND MON



In this chapter, we'll see how the Simple Network Management Protocol (SNMP) can be used in conjunction with the Mon monitoring package to alert you of problems on the cluster nodes. The SNMP daemon, `snmpd`, runs on each cluster node, and the Mon monitoring daemon runs on the cluster node manager sitting outside of the cluster. When Mon finds a problem, it alerts you via an email message or a text message sent to your PDA device, so that you can take action before the problem affects users running mission-critical applications on the cluster.

### Mon

The humbly named Mon monitoring package uses a hierarchical relationship between events to decide whether or not to alert the system administrator. For example, if you are watching the `http` and `telnet` daemons on a cluster node, and you are also watching to see if the node is alive using the `ping` utility (and ICMP packets), you don't need three alerts: the first

reporting that http is not available, the second complaining that telnet doesn't work, and finally a third telling you the node is not available on the network.

## **Mon Alerts**

Mon can send alerts using a variety of methods, including:

- SNMP traps.
- Email messages.
- Short Message System (SMS) text messages to a cell phone or PDA device.
- A custom script or program.

Mon allows you to control how often an alert is sent if a service continues to be down for a specified time period and to control where and how often alerts are sent according to the day of the week or the time of day.

## **Mon Monitoring Scripts**

To monitor a service, Mon runs a monitoring script and passes arguments to it (an IP address to monitor, for example). Mon looks at three things returned by the script:

- The exit or return status of the script.
- The first line of output printed by the monitor script (the *status summary* in Mon-speak).
- Any remaining output from the monitor script (the *status detail*).

### **Mon and Monitoring Script Return Codes**

Mon looks at the return code of the monitoring script. If the return code is 0, Mon knows that nothing bad has happened. Any nonzero return value tells Mon that the test failed. If a script returns a nonzero status more than once, Mon will start comparing the status summary, which is the first line of text printed by the script, to see if it should issue another alert due to the change in the status summary.

## **Where to Run Mon**

In the Linux Enterprise Cluster we will run Mon on a cluster node manager, a machine outside the cluster (see Figure 17-1). The cluster node manager can also offer services to other cluster nodes such as a print spooler, an NIS or LDAP server, a fax server, an email server, and so forth.

**NOTE** The cluster node manager should be built in a high-availability configuration (a pair of servers running Heartbeat as described in Part II) in order to avoid a single point of failure. Examples of how to use Mon with Heartbeat are discussed later in this chapter.

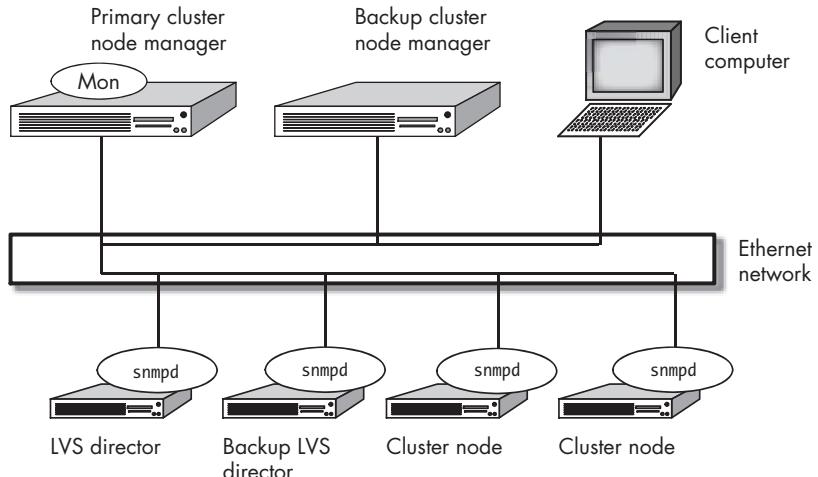


Figure 17-1: Mon and snmpd for cluster monitoring

Figure 17-1 shows Mon running on the primary cluster node manager and one `snmpd` daemon running on each cluster node. We'll see in the recipe that follows how to configure Mon to communicate with the `snmpd` daemon on each node.

## Basic Mon Recipe

This chapter has three Mon recipes that build on each other to provide a Mon configuration capable of monitoring the cluster nodes using SNMP. Each cluster node runs the `snmpd` daemon, which monitors system resources and writes information to a Management Information Base (MIB) stored locally on each cluster node. Mon, running on the cluster node manager, examines the health of each cluster node by reading each of these MIBs across the network, as shown in Figure 17-2.

Notice in Figure 17-2 that the MIB is stored locally on each cluster node. Mon can select the information it wants from each MIB on the cluster nodes.

List of ingredients:

- The Mon software (on the CD-ROM included with this book or from <http://www.kernel.org>)
- The fping package (on the CD-ROM or from <http://www.fping.com>)
- The SNMP software package

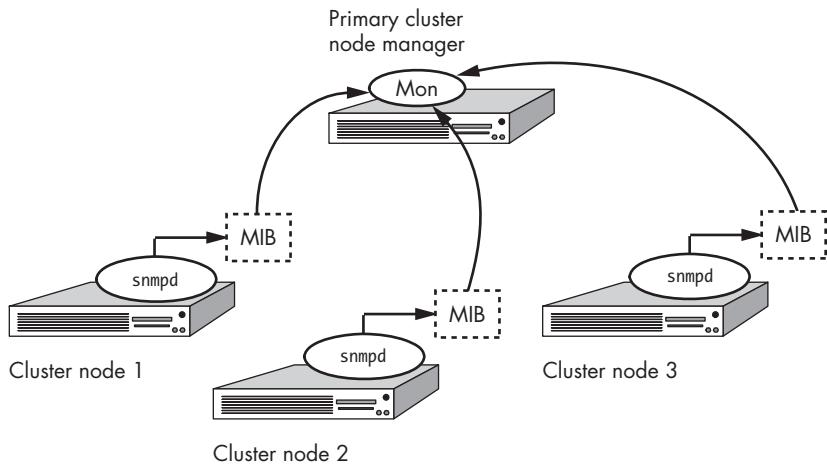


Figure 17-2: Mon examines the MIB of each cluster node

### Step 1: Compile and Install the fping Package

Your first step is to compile and install the fping package. You'll find the fping source code on the CD-ROM in the chapter17 directory. After untarring the fping package (see Appendix A), enter the following commands in the fping source directory to install the /usr/local/sbin/fping program:

---

```
#!/configure
#make
#make check
#make install
```

---

**NOTE** You can use fping in conjunction with your own script to monitor the status of the cluster nodes in the network, as discussed on the fping man page. However, in this recipe we will use the powerful features of the Mon package.

### Step 2: Install the SNMP Package

Next, you need to install the Perl SNMP package that Mon will use to monitor hosts using SNMP. Your distribution probably already ships with SNMP support, but it may not include the Perl SNMP package.

You can compile the Perl SNMP package from CPAN (called simply "SNMP"). If this compile fails, you'll need to rebuild the entire SNMP package and the Perl SNMP package from the source code on the CD-ROM included with this book. To do so, first untar the net-snmp-version.tar file from the CD-ROM on to your local hard drive, and then from inside the net-snmp-version source directory, enter the following.

---

```
#!/configure
#make
#umask 022
#make install
```

---

**NOTE** *You must run these commands as root.*

To build and install the Perl SNMP package, enter:

---

```
#cd perl
#perl Makefile.PL -NET-SNMP-CONFIG="..../net-snmp-config"
#make
#make install
```

---

**NOTE** *The perl directory is located underneath the net-snmp source code directory.*

Instead of compiling and installing the source from the CD-ROM included with this book, you may want to get the latest version by installing the net-snmp-devel package found on the download link at <http://net-snmp.sourceforge.net> and using CPAN to install SNMP with the commands:

---

```
#perl -MCPAN -e shell
cpan> install SNMP
```

---

These two commands download and install the latest Perl SNMP package from the CPAN site.

### ***Step 3: Install the Required CPAN Modules for Mon***

The chapter17/dependencies directory on the CD-ROM holds the Perl modules you need to get Mon up and running. To install these modules, copy them to your local hard drive, untar them, and then `cd` into each directory and enter the commands:

---

```
#perl Makefile.PL
#make
#make test
#make install
```

---

**NOTE** *If the SNMP installation fails, use the net-snmp source code included on the CD-ROM as described in the previous section.*

## DOWNLOADING MODULES FROM CPAN

You can also download the CPAN modules (see Chapter 16 for an example and more information about how to use CPAN) using the commands:

```
#perl -MCPAN -e shell
cpan>install Time::Period
cpan>install Time::HiRes
cpan>install Convert::BER
cpan>install Mon::Protocol
cpan>install Mon::SNMP
cpan>install Mon::Client
```

And, depending upon what you want to monitor, you may also wish to install the following optional modules:

```
cpan>install Filesys::DiskSpace
cpan>install Net::Telnet
cpan>install Net::LDAPapi
cpan>install Net::DNS
cpan>install SNMP
```

## Step 4: Install the Mon Software

Before installing Mon, create the directory where the Mon files will reside before downloading:

```
#mkdir -p /usr/lib/mon
```

Next, untar the Mon tar file located in the chapter17 directory on the CD-ROM into this directory (or download Mon from <http://www.kernel.org/software/mon>).

```
#mount /mnt/CD
#cd /mnt/CD/chapter17
#tar xvf mon-<version>.tar -C /usr/lib
#mv /usr/lib/mon-<version> /usr/lib/mon
```

You should now have a /usr/lib/mon directory with a list of files that looks like this:

alert.d	clients	CREDITS	INSTALL	mon.d	README	utils
cgi-bin	COPYING	doc	KNOWN-PROBLEMS	mon.lsm	state.d	VERSION
CHANGES	COPYRIGHT	etc	mon	muxpect	TODO	

Edit `/etc/services`, and add the following two lines to the end of the file:

---

mon	2583/tcp	# MON
mon	2583/udp	# MON traps

---

### **Step 5: Create the `/etc/mon/mon.cf` Configuration File**

To help make administration of Mon easier, create a symbolic link from the `/usr/lib/mon/etc` directory (just created by the software download) to the `/etc/mon` directory on the system with the following command:

---

```
#ln -s /usr/lib/mon/etc /etc/mon
```

---

The chapter17 directory on the CD-ROM contains the following sample configuration file.

**NOTE** *Do not enter the arrows and the numbers that follow them; they appear here for documentation purposes only.*

---

```
#  
# Simplified cluster "mon.cf" configuration file  
#  
alertdir      = /usr/lib/mon/alert.d ← 1  
mondir        = /usr/lib/mon/mon.d ← 2  
logdir        = /usr/lib/mon/logs ← 3  
histlength    = 500 ← 4  
dtlogging     = yes ← 5  
dtlogfile     = /usr/lib/mon/logs/dtlog ← 6  
  
hostgroup clusternodes clnode1 clnode2 clnode3 ← 7  
                           ← 8  
watch clusternodes ← 9  
    service cluster-ping-check ← 10  
    interval 5s ← 11  
    monitor fping.monitor ← 12  
    period wd {Sa-Su} ← 13  
        alert mail.alert alert@domain.com ← 14  
        upalert mail.alert alert@domain.com ← 15  
        alertevery 1h ← 16
```

---

- 1 → Path to the alert scripts.
- 2 → Path to the monitor scripts.
- 3 → Path to the log file.
- 4 → The maximum number of events to save in the log.
- 5 → Enable downtime logging.
- 6 → Log to store downtime events.
- 7 → List of cluster nodes assigned to a group.
- 8 → Required empty line after each hostgroup.

- 9 → Watch all of the nodes in the group.
- 10 → Call the service anything you want.
- 11 → How frequently the fping utility is run.
- 12 → Use the fping.monitor script.
- 13 → Type perldoc Time::Period for syntax.<sup>1</sup>
- 14 → When one of the nodes goes down.
- 15 → When one of the nodes comes up.
- 16 → Only send alert email once per hour.

**NOTE** *This configuration file is located on the CD-ROM in the chapter17 directory.*

Note the time period definition in this configuration file is period wd {Sa-Su}. If the host does not reply to a ping attempt when Mon is running outside this time period, no alert will be sent. This example uses Saturday thru Sunday as its period, which means the host can fail at any time and Mon will report it.

**NOTE** *We are using a very short interval of five seconds (5s) for testing purposes. You may want to reduce the amount of ping (ICMP) traffic on your network by increasing this time interval when you use Mon in production.*

## COMPLEX MON CONFIGURATION FILES AND THE M4 MACRO PROCESSOR

Complex Mon configuration files can be built using the GNU m4 macro processor. The full documentation for the m4 processor is available online at <http://www.gnu.org/software/m4/manual/index.html>. Why use the m4 macro processor to process your Mon configuration files? The m4 macro processor allows you to create sophisticated configuration files that contain, among other things, variables. You don't have to retype the same thing multiple times throughout the configuration file (one email address for all of your alerts, for example). See the /usr/lib/mon/etc/example.m4 configuration file that comes with the Mon distribution for an example. When you build a configuration file that relies on the m4 processor, you can manually convert it to a normal configuration file that Mon understands by passing it through the m4 macro processor each time you make a change, or you can start Mon with the -c option followed by the full path and file name of the configuration file and Mon will pass your configuration file through the m4 macro processor for you automatically.

### Step 6: Test by Running the fping.monitor and mail.alert Scripts Manually

To simulate what Mon will do, run our alert script manually:

---

```
#/usr/lib/mon/mon.d/fping.monitor clnode1
```

---

<sup>1</sup> wd is a code passed to the Time::Period Perl program to indicate the scale used to define the time period. The Perl period program tests to see if the time right now falls within the time period value you assign using the wd scale. The wd scale permits 1-7 or su, mo, tu, we, th, fr, sa to be used in the period definition. Thus, {Sa-Su} means the period is Saturday through Sunday, which is the same as saying all week or anytime. (A blank period would mean the same thing.)

Assuming the chnode1 computer is online and able to reply to ICMP packets, you should see output that looks like this:

---

```
start time: <current date>
end time  : <current date>
duration  : 0 seconds

-----
reachable hosts          rtt
-----
209.100.100.2           58.10 ms
```

---

This report indicates that cluster node1 at IP address 209.100.100.2 responded to the ping in 58.10 milliseconds.

If this script does not work properly, you may need to tell Perl where to find the fping utility. For example, edit the fping.monitor file, and change the line:

---

```
my $CMD = "fping -e -r $RETRIES -t $TIMEOUT";
```

---

so it looks like this:

---

```
my $CMD = "/usr/local/sbin/fping -e -r $RETRIES -t $TIMEOUT";
```

---

To test the email alert, enter:

---

```
#echo "Testing 123" | /usr/lib/mon/alert.d/mail.alert alert@domain.com
```

---

where alert@domain.com is the email address you would like to send email messages to.

### ***Step 7: Create the Mon Log Directory and Mon Log File***

Create the Mon log directory based on the logdir parameter you used in the mon.cf file.

---

```
#mkdir -p /usr/lib/mon/logs
```

---

### ***Step 8: Start the Mon Program in Debugging Mode and Test***

Before we run Mon as a daemon, we can run it from a shell prompt to check our configuration for syntax errors with the command:

---

```
#/usr/lib/mon/mon -d
```

---

If all of the cluster nodes are “pingable,” you will see (among other things) lines like the following scroll by at each interval (recall that we set the interval to five seconds for testing purposes):

---

```
PID 8211 (clusternodes/cluster-ping-check) exited with [0]
```

---

This line indicates that the cluster responded to the ping test. However, if you take down a cluster node, you’ll see the exit status of the script change from 0 to 1. You can also watch the `/var/log/messages` file for a report that the `mail.alert` script was run. When Mon sees the node drop off the network, you should get an email alert, and when it returns to the network, you should receive an email upalert.

Now let’s build on this basic Mon configuration by adding support for the SNMP protocol.

## Mon and SNMP “Proof of Concept” Recipe

SNMP is a protocol for network management, and a clustered environment of homogenous, or even semi-homogenous, hosts is the ideal way to take full advantage of it.

But before you can use SNMP, you’ll need to install the Net-SNMP (formerly UCD-SNMP) package on each cluster node.

### **Step 1: Install Net-SNMP Client Software on Each Cluster Node**

You can build and compile the `net-snmp` package using the source tar ball located in the `chapter17` directory on the CD-ROM included with this book, but your distribution probably already contains the SNMP software. Red Hat, for example, uses the following two RPM packages:

- `ucd-snmp`
- `ucd-snmp-utils`

Check to see whether you already have them with the command:

---

```
#rpm -qa | grep ucd-snmp
```

---

If you don’t, you’ll need to install them (see Appendix D).

**NOTE** *We built the Net-SNMP software from source code earlier in this recipe because Mon requires the Perl SNMP support. However, this only needs to be done on the cluster node manager; it is not required on each cluster node (though there’s no harm in compiling and installing SNMP from source code on all of your cluster nodes).*

## Step 2: Create the snmp.conf Configuration File

All cluster nodes use the same snmp.conf configuration file. You can create it by entering the following lines into the /etc/snmp/snmpd.conf file:

---

```
syslocation "Room 133" ← 1
syscontact alert@domain.com ← 2
rwcommunity private ← 3
rocommunity public ← 4
authtrapenable 1 ← 5
trapcommunity trapServ ← 6
trapsink localhost ← 7
trap2sink localhost ← 8
disk / 100000 ← 9
disk /var 100000 ← 9
```

---

- 1 → Quoted string.
- 2 → Who is responsible for this system?
- 3 → Community names are really passwords.
- 4 → Password for read-only information.
- 5 → Report SNMP authentication failures.
- 6 → Password to use when sending traps.
- 7 → Where to send SNMP traps.
- 8 → SNMP version 2 traps.
- 9 → Disk partition and minimum freespace.

**NOTE** *If you have an existing /etc/snmp/snmpd.conf file, you can make a backup copy and remove everything from the file except for the lines shown in this example.*

The community read-write and read-only names in this file (rwcommunity and rocommunity) are really the SNMP “passwords” used to access and write to the SNMP MIBS for this agent. Do not use these common values unless you are building your system behind a firewall.

The last two disk lines tell SNMP to raise an alert when there is less than 100 MB of free disk space on either of the disk partitions / or /var.

## Step 3: Start the SNMP Agent

The init script to start the SNMP agent should have been installed with the RPM package. Check for the existence of the file in the /etc/init.d directory, and then run the command:

---

```
#service snmpd start
```

---

or

---

```
#/etc/init.d/snmpd start
```

---

The `/var/log/messages` file should report:

---

```
snmpd: snmpd startup succeeded
```

---

**NOTE** Be sure `snmpd` starts each time the system boots on all of the cluster nodes. See “*Respawning Services with init*” in *Chapter 1* for instructions on how to automatically restart the `snmpd` daemon on each cluster node if it fails.

## Step 4: View the SNMP MIB Locally

Examine the MIB on a cluster node with the command:

---

```
#snmpwalk localhost -v 1 -c public
```

---

This command should produce a long report showing the MIB on the cluster node indicating that the local SNMP agent (`snmpd`) is responding to the query.

Now let’s look at something useful with the command:

---

```
#snmpwalk -v 1 localhost -c public .1.3.6.1.4.1.2021.9
```

---

**NOTE** In this command, we are using a numeric value instead of a symbolic name to locate information in the SNMP management information base, or MIB. To find the symbolic or textual name of this numeric object identifier (OID) number, use:

---

```
#snmptranslate .1.3.6.1.4.1.2021.9
```

---

If you set the disk values as shown in the configuration file example, you should see a disk report like this:

---

```
enterprises.ucdavis.dskTable.dskEntry.dskIndex.1 = 1
enterprises.ucdavis.dskTable.dskEntry.dskIndex.2 = 2
enterprises.ucdavis.dskTable.dskEntry.dskPath.1 = /
enterprises.ucdavis.dskTable.dskEntry.dskPath.2 = /var
enterprises.ucdavis.dskTable.dskEntry.dskDevice.1 = /dev/sdb2
enterprises.ucdavis.dskTable.dskEntry.dskDevice.2 = /dev/sda3
enterprises.ucdavis.dskTable.dskEntry.dskMinimum.1 = 100000
enterprises.ucdavis.dskTable.dskEntry.dskMinimum.2 = 100000
enterprises.ucdavis.dskTable.dskEntry.dskMinPercent.1 = -1
enterprises.ucdavis.dskTable.dskEntry.dskMinPercent.2 = -1
enterprises.ucdavis.dskTable.dskEntry.dskTotal.1 = 381121
enterprises.ucdavis.dskTable.dskEntry.dskTotal.2 = 253871
enterprises.ucdavis.dskTable.dskEntry.dskAvail.1 = 268888
enterprises.ucdavis.dskTable.dskEntry.dskAvail.2 = 162940
enterprises.ucdavis.dskTable.dskEntry.dskUsed.1 = 92554
enterprises.ucdavis.dskTable.dskEntry.dskUsed.2 = 77824
```

---

```
enterprises.ucdavis.dskTable.dskEntry.dskPercent.1 = 26
enterprises.ucdavis.dskTable.dskEntry.dskPercent.2 = 32
enterprises.ucdavis.dskTable.dskEntry.dskPercentNode.1 = 18
enterprises.ucdavis.dskTable.dskEntry.dskPercentNode.2 = 0
enterprises.ucdavis.dskTable.dskEntry.dskErrorFlag.1 = 0
enterprises.ucdavis.dskTable.dskEntry.dskErrorFlag.2 = 0
enterprises.ucdavis.dskTable.dskEntry.dskErrorMsg.1 =
enterprises.ucdavis.dskTable.dskEntry.dskErrorMsg.2 =
```

---

**NOTE** You see the *University of California at Davis* name in this disk space SNMP MIB because they developed it.

Notice especially the `dskEntry.dskErrorFlag` lines (shown in bold in this example). These lines indicate whether the disk is below (value 0) or above (value 1) the threshold you specified in the `snmpd.conf` file. Take a moment to modify the disk threshold value (in megabytes) to one that is less than the amount of free disk space on one of the partitions (use the command `df -m` to check). Then enter:

---

```
#service snmpd restart
```

---

or

---

```
#/etc/init.d/snmpd restart
```

---

**NOTE** A *kill HUP* to the *PID* of the `snmpd` daemon accomplishes the same thing.

Then enter the same `snmpwalk` command again:

---

```
#snmpwalk -v 1 localhost -c public .1.3.6.1.4.1.2021.9
```

---

You should now see that the error flag has been set to indicate that the disk partition is running out of disk space.

## **Step 5: Install the SNMP Monitoring Agents**

We're now ready to tell Mon to examine the SNMP MIB on the cluster node manager. In this test, we are using Mon to monitor the SNMP agent running on the local host (the cluster node manager itself) as an example. You'll be able to do the same for the entire cluster by simply launching SNMP daemons on all of your cluster nodes and adding them to the Mon configuration file on the cluster node manager.

**NOTE** You must have installed the Perl SNMP package as described earlier in this chapter so that the SNMP monitoring scripts that come with the Mon package can run properly on the cluster node manager.

In this example, we will use the SNMP disk monitoring program called `netsnmp-freespace.monitor` located on the CD-ROM in the `chapter17` directory and online at <http://ftp.kernel.org/pub/software/admin/mon/contrib/index.html>. To install, enter:

---

```
#cp /mnt/CDROM/chapter17/netsnmp-freespace.monitor /usr/lib/mon/mon.d/
#chmod 755 /usr/lib/mon/mon.d/netsnmp-freespace.monitor
#/usr/lib/mon/mon.d/netsnmp-freespace.monitor localhost
```

---

If you downloaded this monitoring utility and you see the error message `localhost returned an SNMP error: Unknown user name`, modify the `netsnmp-freespace.monitor` script to force it to use Version 2 of the SNMP protocol. (See the `netsnmp-freespace.monitor` script included on the CD-ROM for this one-line change.)

If you need to use a community name other than `public` to be able to read the SNMP MIB (because you specified something other than `public` in the `snmpd.conf` file), use the `-c` option to the `netsnmp-freespace.monitor` script.

### ***Examine the Output of `netsnmp-freespace.monitor`***

This script returns nothing if a disk space threshold has not been violated. Try modifying the freespace threshold in `/etc/snmp/snmpd.conf`, and then restart the `snmpd` service (`service snmpd restart` or `/etc/init.d/snmpd restart`) and run this command again. When a disk is violating a freespace threshold, you'll see a report like this:

---

```
#/usr/lib/mon/mon.d/netsnmp-freespace.monitor localhost
localhost: /dev/sdb2 total=372 used=91(26%) free=261 err=/: less than 300000
free (= 267767)
```

---

### ***Install Mon on All Cluster Nodes***

You are now ready to put Mon to work and ask it to watch the disk space on all of your cluster nodes. To do so, install the `/etc/snmp/snmpd.conf` file you created on all of your cluster nodes, and start the `snmpd` daemon on each node.

You can use this sample `mon.cf` configuration file on the cluster node manager to add `netsnmp-freespace.monitor`.

---

```
alertdir      = /usr/lib/mon/alert.d
mondir        = /usr/lib/mon/mon.d
logdir        = /usr/lib/mon/logs
histlength    = 500
dtlogging     = yes
dtlogfile     = /usr/lib/mon/logs/dtlog
```

---

```
hostgroup clusternodes localhost

watch clusternodes
  service cluster-ping-check
    interval 30s
    monitor fping.monitor
    period wd {Su-Sa}
      alert mail.alert alert@domain.com
      upalert mail.alert alert@domain.com
      alertevery 1h
  service disk-space-check
    interval 15m
    monitor netsnmp-freespace.monitor
    period wd {Su-Sa}
      alert mail.alert alert@domain.com
      upalert mail.alert alert@domain.com
```

---

For the moment, the `clusternodes` group contains only `localhost` (which should point to the loopback IP address `127.0.0.1` in the `/etc/hosts` file). Start Mon again, and you can generate alerts based on disk thresholds on the local system by modifying the `/etc/snmp/snmpd.conf` file and restarting the `snmpd` daemon.<sup>2</sup>

Now that we have a proof of concept for an SNMP configuration using Mon and SNMP, let's build a configuration that can be used to monitor important system functions on each cluster node.

## Mon and SNMP “Real-World” Recipe

Sophisticated resource monitoring is easy with SNMP. By simply adding the correct line to your `snmpd.conf` file, you can add useful information to the MIB that can then be accessed by the Mon daemon from the cluster node manager.

### **Step 1: Create the `snmpd.conf` on All Cluster Nodes**

On the CD-ROM in the `chapter17` directory, you'll find a file called `snmpd.conf.example` that contains the following configuration to monitor both disk space and processes:

---

```
# GENERAL SNMP INFORMATION
syslocation "Room 133"
syscontact alert@domain.com
rwcommunity private
rocommunity public
authtrapenable 1
```

---

<sup>2</sup>If you used the technique described in Chapter 1 to start `snmpd` from an entry in the `/etc/inittab` file, you need only enter the following command to restart `snmpd`:  
`#killall snmpd`

```
trapcommunity trapServ
trapsink localhost
trap2sink localhost

# DISK SPACE MONITORING
# Disk partitions should have at least 100 MB of
# freespace.
disk / 100000
disk /var 100000
disk /home 10000
disk /usr 10000

# PROCESS MONITORING
# System init
proc init

# Job Scheduling and system time.
proc atd
proc ntpd

# System logging
proc syslogd

# Start services based on in-bound connections
proc xinetd

# Remote machine login
proc sshd

# Printing
proc lpd

# Web services (may be used in cluster environment
# for monitoring servers inside the cluster).
proc httpd

# NFS Client software
proc portmap
proc lockd
proc rpc.statd
# Optional Automounter daemon
# proc automount

# NIS Client software
proc ypbnd
```

---

**NOTE** *The crond daemon is not included here because we will only run the cron daemon on one node. (See Chapter 19.)*

Copy this file into place by naming it `/etc/snmp/snmpd.conf`, and then restart the `snmpd` daemon and check the `/var/log/messages` file to see if any errors are reported when the `snmpd` daemon first starts.

## **Step 2: Install `netsnmp-proc.monitor` and the New `mon.cf` File on the Cluster Node Manager**

Mon only needs to be run on the cluster node manager. The `netsnmp-proc.monitor` script is also included on the CD-ROM (modified to support establishing the SNMP session using version 2 of the protocol).<sup>3</sup> Copy it to the `/usr/lib/mon/mon.d` directory and make sure to set the execute permissions on the file:

---

```
#cp /mnt/CDROM/chapter17/netsnmp-proc.monitor /usr/lib/mon/mon.d/
#chmod 755 /usr/lib/mon/mon.d/netsnmp-proc.monitor
```

---

The CD-ROM also includes another `mon.cf` configuration file named `mon.cf.2`:

```
alertdir      = /usr/lib/mon/alert.d
mondir        = /usr/lib/mon/mon.d
logdir        = /usr/lib/mon/logs
histlength    = 500
dtlogging     = yes
dtlogfile     = /usr/lib/mon/logs/dtlog

hostgroup clusternodes localhost clnode1 clnode2 clnode3

watch clusternodes
  service cluster-ping-check
    interval 30s
    monitor fping.monitor
    depend clusternodes:cluster-ping-check
    period wd {Su-Sa}
      alert mail.alert alert@domain.com
      upalert mail.alert alert@domain.com
      alertevery 1h
  service disk-space-check
    interval 15m
    monitor netsnmp-freespace.monitor -t 10000000
    depend clusternodes:cluster-ping-check
    period wd {Su-Sa}
      alert mail.alert alert@domain.com
      upalert mail.alert alert@domain.com
  service process-check
    interval 30s
```

---

<sup>3</sup>You can also download the script from <http://ftp.kernel.org/pub/software/admin/mon/contrib/monitors/netsnmp/>.

```
monitor netsnmp-proc.monitor -t 10000000
  depend clusternodes:cluster-ping-check
period wd {Su-Sa}
  alert mail.alert alert@domain.com
  upalert mail.alert alert@domain.com
service telnet-check
  description telnet to servers in cluster
  interval 2m
  monitor telnet.monitor
  depend clusternodes:cluster-ping-check
  period wd {Sun-Sat}
  alertevery 1h
```

---

You may want to add additional (hostgroup) entries to this file in order to monitor the LDAP server,<sup>4</sup> the DNS server, and the mail server using additional monitor scripts available from the download site.

**NOTE** *The SNMP monitors are being passed the -t 10000000 argument, which means that the SNMP response can take up to ten seconds.*

This configuration file introduces the syntax necessary to create a hierarchical relationship between the things being monitored. The line:

---

```
depend clusternodes:cluster-ping-check
```

---

when inserted in each service paragraph causes Mon to suppress alerts if a cluster node does not respond to a ping.

**NOTE** *See the Mon man page for the complete description of the depend syntax. (You can use any Perl expression that evaluates as true or false to create complex dependencies.)*

### **Step 3: Install the Mon init Script**

The chapter17 directory on the CD-ROM contains a script called `mon` that can be placed in the `/etc/init.d` directory and then added to the normal boot process using the `chkconfig` command (on Red Hat systems).

---

```
#cp /mnt/CDROM/chapter17/mon /etc/init.d/mon
#chmod 755 /etc/init.d/mon
#chkconfig --add mon
```

---

---

<sup>4</sup> Requires the Net::LDAP package (which requires the Convert::ASN1 package) from CPAN. Here is a sample `mon` entry to use the `ldap.monitor` to look up the account `admin` in the LDAP database for `basedn yourdomain.com`:

```
monitor ldap.monitor -basedn "dc=yourdomain,dc=com" -filter="uid=admin" -attribute=uid -value=admin
```

## Step 4: Run SNMP and Mon

With the new configuration files in place, restart the SNMP service on each cluster node and the Mon service on the cluster node manager.<sup>5</sup> On each cluster node, enter:

---

```
#service snmpd restart
```

---

On the cluster node manager, start Mon in debug mode to make sure your configuration file is working properly (use `/usr/lib/mon/mon -d`). Once you are sure that Mon works, start it using the init script with the command:

---

```
#service mon start
```

---

or

---

```
#/etc/init.d/mon start
```

---

**NOTE** *If you receive an error message that the fping monitor could not be located, see the instructions earlier in this chapter for specifying the path to the fping binary in the `/usr/lib/mon/mon.d/fping.monitor` script.*

## Email Alerts from Mon

When Mon sends an email alert about a failed process, it looks like this:

Subject: ALERT clusternodes/process-check: localhost:ntpd,ypbind

Summary output : localhost:ntpd,ypbind

Group : clusternodes  
Service : process-check  
Time noticed : <Time and Date>  
Secs until next alert :  
Members : localhost

Detailed text (if any) follows:

-----  
localhost:ntpd Count=0 Min=0 Max=0  
localhost:ypbind Count=0 Min=0 Max=0

The status summary is placed in the subject line of the email, and the status detail is included in the body of the message. This example shows Mon alerting you that ypbnd and ntpd are not running on the localhost (the machine Mon is running on).

---

<sup>5</sup> In Chapter 19, we will discuss how to run a command on all cluster nodes.

## Creating Your Own SNMP Script

The SNMP software package allows you to create your own resource monitoring script, which is run (usually not more than every 30 seconds) to check the health of system resources and report errors into an SNMP MIB that can be monitored by the Mon program running on a central monitoring server. This not only allows you to customize what you are monitoring by writing a simple script, but with a slightly more complex program or script (one that stores values in a temporary file), you can also monitor the system for sudden changes (like a sudden increase in swap space usage). By default, SNMP runs your script and places the exit status of the program after each execution into MIB 1.3.6.1.4.1.2021.8.1.100 and the first line of output (on stdout) from the script into MIB 1.3.6.1.4.1.2021.8.1.101. (Any additional output returned by the script is ignored.)

The syntax in the `/etc/snmp/snmpd.conf` file to specify the name of the script, or program, to run looks like this:

---

```
sh NAME PROG [ARGS]
```

---

where:

*NAME* is an arbitrary name to describe your monitoring script.

*PROG* is the full path and file name of your script, or program.

*ARGS* an optional argument or list of arguments to pass to the script.

A sample entry in the `/etc/snmp/snmpd.conf` configuration file looks like this:

---

```
sh check-dns /usr/local/check-dns
```

---

Here, the program `/usr/local/check-dns` will run (without arguments) when the MIB is queried, or every 30 seconds if the MIB is queried more often than that.

### ***A Sample Custom SNMP Script***

Now, let's create the `/usr/local/check-dns` script with the following simple bash script:

---

```
#!/bin/bash
host=`uname -n`

hostnotfound=`/usr/bin/host www.google.com | grep "not found"`

if [ "$hostnotfound" ];then
echo "DNS lookup not working properly on $host"
exit 1
fi
```

---

If the DNS query of host `www.google.com` fails, the script returns one line of error and exits with a return value of 1. Save this script to the file `/usr/local/check-dns` (be sure to set the execute bits with the command `chmod 755 /usr/local/check-dns`) and run it first with a known good host like `www.google.com`; then modify the script and try a bogus host name (one that does not resolve). Then add the line shown above to your `/etc/snmp/snmpd.conf` file and start, or restart, the `snmpd` daemon.

When using a bogus host entry in the script file, you should see the following output when you examine the MIB entry:

---

```
#snmpwalk -v 1 localhost -c public 1.3.6.1.4.1.2021.8.1.101
UCD-SNMP-MIB::extOutput.1 = STRING: DNS services not working properly on
primary.yourdomain.com
```

---

And when you change the script back to a valid host name for the DNS query, you should see nothing in the MIB when you run the same `snmpwalk` command.

## Monitoring Your SNMP Script with Mon

Now let's monitor this SNMP MIB with another SNMP monitoring script called `netsnmp-exec.monitor`. Copy this script from the `chapter17` directory on the CD-ROM into the `/usr/lib/mon/mon.d` directory, and set the permissions of the file so Mon can execute it:

---

```
#cp /mnt/CDROM/chapter17/netsnmp-exec.monitor /usr/lib/mon/mon.d
#chmod 755 /usr/lib/mon/mon.d/netsnmp-exec.monitor
```

---

Create a `mon.cf` service entry (underneath the existing `watch clusternodes` entry) that looks like this:

---

```
service custom-snmp-monitor-script
description Checks the output of the custom SNMP script
interval 5s
monitor netsnmp-exec.monitor
depend clusternodes:ping
period wd {Su-Sa}
alert mail.alert alert@yourdomain.com
upalert mail.alert alert@yourdomain.com
```

---

(Of course, you would normally install this script on all of the cluster nodes and modify the `/etc/snmp/snmpd.conf` file on all of the cluster nodes.)

You can test this Mon configuration (once you've restarted Mon) by modifying the `check-dns` script again and changing the host to a bogus host. This should cause Mon to raise the alert you have specified (Mon will send an email to `alert@yourdomain.com` in this example).

You can have multiple custom monitoring scripts. Each additional script can be monitored individually with the `netsnmp-exec.monitor` script by passing it the `-n` argument followed by a number to represent which script to use from the `snmpd.conf` file. The default, if the `-n` argument is not used, is to check all scripts.

**NOTE** *You may also want to use the `-t 10000` argument to specify a timeout value for the `netsnmp-exec.monitor` script in your `mon.cf` entry.*

## Things to Monitor with SNMP Monitoring Scripts

Using this method, you will only be able to send one line of an error message;<sup>6</sup> however, a single script can return a *different* line of output each time it runs, and Mon will report it each time (because Mon will consider each change in output a new alert).

Here are a few things that you might want to monitor with your custom monitoring scripts:

- A sudden increase in page faults. (A bash script to do this called `page-faults.sh` is included in the `chapter17` directory on the CD-ROM.)<sup>7</sup>
- A sudden increase or decrease in free swap space, or if free swap space is getting too close to the maximum. (See the script `swap.sh` also included on the CD-ROM in the `chapter17` directory.)
- The number of defunct<sup>8</sup> processes on the system.
- Logins with very high idle (inactivity) times.
- Processes consuming too much of the CPU.
- Assuming you are using NFS clients, you may also want to monitor the number of IP reassembly failures (failed attempts by the kernel to reassemble network packets containing fragmented data) by examining the SNMP variable `IP-MIB::ipReasmFails`. Here is a sample command that will do this:

---

```
#snmpwalk localhost -c public IP-MIB::ipReasmFails.0
```

---

**NOTE** *You can write a script that calls another script, but just be sure to pass the exit status of the child script back as the exit status of the main script or SNMPD will not see it.*

---

<sup>6</sup> See the SNMP traps and sending SNMP trap alerts sections in the SNMP and Mon documentation if you need to send more than one-line error messages in your SNMP messages.

<sup>7</sup> The first time you run this script you'll see an error that you can ignore because the historical page faults information doesn't exist.

<sup>8</sup> Defunct processes are shown on a `ps` report and usually cannot be removed from the system without rebooting.

## Forcing a Stonith Event with Mon

We can use Mon on the cluster node manager to access cluster services from outside the cluster. Mon can test the health of the cluster from the perspective of a client computer and use custom scripts to take corrective action to return the services to normal operation. For example, if the cluster services are not available on the VIP address, Mon can check to see if the services are still available on the cluster node RIP addresses. If the services are working normally on the RIP addresses but not the VIP address, Mon knows the Director is malfunctioning and corrective action is required.

Here is a sample mon.cf configuration file to accomplish this:

---

```
alertdir      = /usr/lib/mon/alert.d
mondir        = /usr/lib/mon/mon.d
logdir        = /usr/lib/mon/logs
histlength    = 500
dtlogging     = yes
dtlogfile     = /usr/lib/mon/logs/dtlog

# The list of cluster node host names.
hostgroup clusternodes clnode1 clnode2 clnode3 clnode4

# 209.100.100.2 is the LVS VIP address
hostgroup clusterservices 209.100.100.2

watch clusternodes
  service RIP
    interval 30s
    monitor telnet.monitor
    period wd {Su-Sa}
      alert mail.alert alert@domain.com
      upalert mail.alert alert@domain.com
      alertevery 1h

watch clusterservices
  service VIP
    interval 1m
    monitor telnet.monitor
    depend clusternodes:RIP
    period wd {Su-Sa}
      alterafters 5
    alert mail.alert alert@domain.com
    alert initiate.stonith.event
```

---

**NOTE** *The telnet.monitor script is included with the Mon distribution.*

This configuration file tells Mon to watch the telnet service on the cluster nodes, and if the cluster nodes are available when telnet is not available on the cluster VIP (from the LVS Director), the Mon server will

reset the power to the Director (the standby Director should take over the cluster load-balancing service and resume offering the telnet service to client computers).

The `initiate.stonith.event` script should contain the Stonith commands to power reset the LVS Director. To create this script, we first need to build a Stonith configuration file—we'll call ours `/etc/stonith.cfg`. In this file, place the correct syntax for your Stonith device (see Chapter 9). Here is the syntax for a Western Telematic Remote Power Switch `rps10` device:

---

```
#vi /etc/stonith.cfg
/dev/ttys0    lvsdirector    0
```

---

This single line in the `/etc/stonith.cfg` file tells the `stonith` command (we'll use it in a moment) that the `RPS10` device is connected to the first serial port on the cluster node manager, and that it is controlling power to the host named `lvsdirector` using the first port on the RPS device.

Now create the `initiate.stonith.event` script containing:

---

```
#!/bin/bash
#
# initiate.stonith.event
#
/usr/sbin/stonith -F /etc/stonith.cfg lvsdirector
```

---

Place this file in the `/usr/lib/mon/alert.d` directory, and make sure it has the execute bit set:

---

```
#chmod 600 /usr/lib/mon/alert.d/initiate.stonith.event
```

---

**NOTE** *You will probably also want to create a `mon.cf` entry to monitor the VIP address that does not depend on the RIP addresses that will notify you via email or pager when the VIP address isn't accessible. Regardless of what has gone wrong, you'll want to know when resources cannot be accessed on the VIP.*

## Forcing a Heartbeat Failover with Mon

Another method to accomplish a failover is to run Mon on the LVS Director and force the LVS Director to release all of its resources using the `hb_standby` program when Mon detects that something has gone wrong. (The Heartbeat package comes with the `hb_standby` program.)

By running Heartbeat and Mon on the LVS Director, you can initiate a Heartbeat failover with a Mon alert script that executes the `hb_standby` program.

**NOTE** *Heartbeat will not stop resources in a resource group (recall that a resource group is a list of resources on a single line in Heartbeat's /etc/ha.d/haresources file) when the hb\_standby program is used to initiate a failover. To avoid a possible split brain condition, you must use Stonith.*

### SOPHISTICATED SNMP MONITORING

A more sophisticated SNMP monitoring technique than the method described in this chapter is available to you when you use the Mon monitoring script called `snmpvar.monitor`. This script is available for download from the Mon contrib page of the Mon website at <http://www.kernel.org/software/mon>.

The `snmpvar.monitor` script allows you to specify SNMP variables along with their associated minimum and maximum values in a configuration file (`snmpvar.cf`). The `snmpvar.monitor` script can then check the SNMP variables on a remote host and tell Mon to raise an alert if a value falls outside of the range you specify.

If you need help with installing the `snmpvar.monitor` monitoring script (or if you have any questions about Mon), you can post your question to the Mon mailing list at <http://www.kernel.org/software/mon/list.html>.

## In Conclusion

An enterprise-class cluster needs to be monitored so you can take action before a problem affects the services that are offered to the client computers. One method of doing this is to use the humbly named Mon package.

Mon is a powerful monitoring system that can enforce complex, hierarchical relationships between conditions and threshold violations when deciding what events should cause alert notifications. Alert notifications can take any form that can be programmed or scripted.



# 18

## GANGLIA



The Ganglia software package provides a sophisticated mechanism for collecting and storing cluster node performance data, and it includes tools for displaying this data in a web browser.

Once your cluster goes into production, you'll use Ganglia's data visualization tools to *watch* the cluster manage and distribute its workload. This will help you to determine if there are any trends that indicate a load-balancing problem, a runaway process, or hardware under capacity. Historical graphs allow you to watch the cluster and see the daily, weekly, and even yearly demand for cluster services and hardware resources.

**NOTE** *Ganglia requires very little CPU, memory, and network resources in order to run—even for clusters with more than 100 nodes.*

## Introduction to Ganglia

The Ganglia project was started at the University of California, Berkeley to support scientific research that depends on computers for data analysis. The Ganglia web page says the project “strives to advance science by creating a ubiquitous, continuous, and pervasive national computational infrastructure: the Grid.” For the moment, however, we are only concerned with Ganglia’s ability to monitor and display information about the cluster we have built.

**NOTE** *Like the term cluster, the term grid has been used to describe a variety of computational systems. The most common use of the term evolved from the scientific community where it was used to describe multiple independently managed geographically disparate computing clusters. Using this definition, a grid can contain a cluster, but a cluster does not contain a grid. Both a grid and cluster, however, are computational environments where workloads run in parallel.*

The Ganglia package consists of several command-line utilities, which we’ll see shortly, and daemons that run on the cluster nodes. Let’s briefly examine two of the Ganglia daemons before I describe how to install Ganglia on the cluster.

### **gmond**

`gmond` is the Ganglia monitoring daemon. `gmond`’s job is to gather performance metrics on the machine it is running on and to keep track of the status of every other `gmond` daemon running in the cluster. If one of the `gmond` daemons dies (due to a failure of the cluster node, for example), all surviving `gmond` daemons find out about it.

The `gstat` utility can report the information collected by `gmond` in XML format. Starting with version 2.7.0, `gmond` can compress this XML data before it is sent over the network.

### **gmetad**

The `gmond` daemon is required on every cluster node, but the `gmetad` daemon only needs to run on the cluster node manager.<sup>1</sup> The `gmetad` daemon polls the `gmond` daemons for their performance metric information every 15 seconds and then stores this information in a round-robin database using the `RRDtool`. (In a round-robin database, the newest data overwrites the oldest data, so the database never fills up.)

`gmetad` displays the information it collects using the Apache web server. Its ability to display web pages is made possible by the Ganglia Web package (formerly called the Ganglia Web Frontend). We’ll discuss Ganglia Web shortly.

---

<sup>1</sup> See the introduction for a description of the cluster node manager.

# Installing Ganglia's Prerequisite Packages

I'll explain how Ganglia works as I describe how to install and use it. Begin by installing `gmond` on the cluster nodes and then `gmetad` and the Ganglia Web package on the cluster node manager. But first, install the other packages required for `gmetad` and the Ganglia Web package to run on the cluster node manager: PHP, RRDtool, and Apache.

## Install PHP

The Ganglia Web package requires PHP. The source code for PHP is included in the `chapter18/version2/dependencies` directory on the CD-ROM included with this book, but you'll probably find it easier to install the PHP software included with your distribution. If you are using Red Hat, for example, the name of the rpm file for php is `php-<version-number>.rpm`. You must install PHP on the cluster node manager before attempting to install Ganglia.

## Install RRDtool

Next, install the RRDtool on the cluster node manager from source code (`chapter18/version2/dependencies` on the CD-ROM). Untar the source file, and then use the following statements at the top of the RRDtool source directory tree to compile it:

**NOTE** *Watch for the small, harmless joke from the RRDtool installation script as it pretends to order a CD for developer Tobi Oeriker. You can help support RRDtool development and maintenance by contributing to the Happy Tobi Project (see <http://www.rrdtool.com/license.html>).*

---

```
#!/configure --enable-shared
#make site-perl-install
#make install
```

---

**NOTE** *You only need to install PHP and RRDtool on the cluster node manager.*

The Ganglia monitoring core `gmetad` RPM expects to find the shared library for the RRDtool (`librrd.so.0`) in the `/usr/lib` directory. Copy it into place with the following command:

---

```
#cp /usr/local/RRDtool-<version>/lib/librrd.so* /usr/lib
```

---

Next, place the RRDtool binaries in a standard system path where Apache can find them:

---

```
#cp /usr/local/RRDtool-<version>/bin/* /usr/bin
```

---

## Install Apache on the Cluster Node Manager

Chapter 12 discusses how to install and configure Apache on cluster nodes (see also Appendix F), and you can use these instructions to install Apache on the cluster node manager as well.

For the Ganglia Web package to work properly, Apache must be configured to support the PHP module, as we'll discuss shortly.

## Installing Ganglia on the Cluster Node Manager

With the required packages installed, you're ready to install Ganglia on the cluster node manager.

Download the latest Ganglia software RPM packages from the Ganglia website, and install them using the standard RPM commands. Here is the list of RPM package names before and after Ganglia release 3.0:

Old RPM Package Name	3.0 and Later RPM Package Name
ganglia-monitor-core	ganglia-gmetad
ganglia-monitor-core-gmond	ganglia-gmond
ganglia-webfrontend	ganglia-web
ganglia-monitor-core-lib	ganglia-devel
gexec and authd	ganglia-gexec-client

On the cluster node manager, you need to install `ganglia-gmetad`, `ganglia-gmond`, `ganglia-web`, `ganglia-devel`, and (optionally) `ganglia-gexec-client`.

**NOTE** *You may have to use `--nodeps` when installing to force the `ganglia-web` RPM to ignore dependencies, because the RPM database will have no record of the RRDtool or PHP software if you installed RRDtool and PHP from source.*

*If you compiled `gmond` from source instead of loading it from RPM, you must use the `--enable-gexec` flag to support `gexec`.*

## Installing Ganglia on the Cluster Nodes

Install `ganglia-gmond` on the cluster nodes, but not `gmetad` (we do not want to collect and store historical performance metrics on each node—we'll do this only on the cluster node manager).

Here again are the RPM package names before and after the Ganglia 3.0 release:

Old RPM Package Name	New (3.0) RPM Package Name
ganglia-monitor-core-gmond	ganglia-gmond
ganglia-monitor-core-lib	ganglia-devel
gexec	ganglia-gexec-server

# Configuring gmetad and gmond on the Cluster Node Manager

It's time to configure Ganglia. Let's start with gmetad and gmond on the cluster node manager. gmetad is controlled by the `/etc/gmetad.conf` file, and gmond is controlled by the `/etc/gmond.conf` file.

## Modify `/etc/gmetad.conf`

The `/etc/gmetad.conf` file on the cluster node manager only needs to contain one line where you specify the name of the cluster<sup>2</sup> and the name of the nodes inside the cluster. For example, the following entry defines a cluster named `Cluster1` and places the localhost (the cluster node manager) and two cluster nodes named `clnode1` and `clnode2`:

---

```
data_source "Cluster1" localhost clnode1 clnode2
```

---

**NOTE** *If a cluster node name changes, you will need to modify the `/etc/gmetad.conf` file on the cluster node manager, restart gmond on all cluster nodes, and then restart gmetad on the cluster node manager to make the change take effect.*

*Each of the nodes in the data source should have identical information for Ganglia to monitor. Because Ganglia communication uses a symmetrical model (multicast), a failure on one node in the list means that Ganglia can connect to the next node. For this reason, it is not necessary to specify every host in a cluster for each `data_source` entry.*

## Modify `/etc/gmond.conf`

The `/etc/gmond.conf` file need only contain one line that specifies the name of the cluster. We'll use this same configuration file on all of the cluster nodes shortly. For example:

---

```
name "Cluster1"
```

---

**NOTE** *See the comments in the `/etc/gmond.conf` file for other options you may need to specify for your configuration (for example, the Ethernet port gmond should use, the number of multicast hops allowed if you are monitoring clusters that are not connected to the same Ethernet switch, and the level of compression gmond should use for XML data<sup>3</sup>).*

---

<sup>2</sup> This is the first (and only) time in this book that we have been required to provide a name for our cluster for any of the software packages we've installed (except for the host name you used for your VIP address that client computers use to connect to cluster services).

<sup>3</sup> The default compression level is 0, or no compression. You should only need to use compression on very large clusters, or when you are monitoring a grid (a cluster of clusters).

## Start gmond and gmetad

After making changes to the two configuration files (`/etc/gmond.conf` and `/etc/gmetad.conf`) on the cluster node manager, start `gmond` and `gmetad`:

---

```
#service gmond restart
#service gmetad restart
```

---

**NOTE** *See Chapter 1 for instructions on how to start these services automatically when the system boots.*

Now, test to make sure `gmond` is working properly by entering the following command on the cluster node manager:

---

```
#telnet 127.0.0.1 8649
```

---

Press ENTER twice to send a carriage return through port 8649 to the `gmond` daemon on the cluster node manager.<sup>4</sup> This should cause your screen to fill up with a long listing of XML code containing the performance metrics that `gmond` is monitoring.

**NOTE** *The `gmetad` daemon stores this information using RRDtool in a subdirectory underneath the `/var/lib/ganglia/rrds/` directory. For clusters with over 100 nodes, you may want to put this directory on a RAM filesystem because the disk I/O for this database will be very high.*

## Add the Ganglia Page to Your Apache Configuration

You're now ready to point Apache to the Ganglia PHP scripts. The RPM installation of `gmetad` placed the PHP programs for Ganglia into the directory `/var/www/html/ganglia/`. You'll need to create an Apache virtual host to point to these files or configure Apache's `DocumentRoot` to use the `/var/www/html/ganglia/` directory as the default directory. (See "Modifying the Apache Configuration File" in Appendix F.)

Once you've changed the Apache configuration file<sup>5</sup> and restarted the Apache daemon, you are ready to point your web browser to the Ganglia Web package web page. If, for example, you set the Apache `DocumentRoot` to `/var/www/html`, you can use the following URL: `http://clusternodemanager/ganglia/index.php`.

In this example, the host name `clusternodemanager` resolves to the IP address of the cluster node manager.

Your web browser should now display the Ganglia main page. In the next section, we'll examine this web page in more detail.

---

<sup>4</sup> If you tell `gmond` to compress the XML data (in the `/etc/gmond.conf` file), you'll need to use the `netcat` (`netcat.sf.net`) program to uncompress that XML data to make it legible. After installing `netcat` the command to use is `nc localhost 8649 | gunzip`.

<sup>5</sup> On some versions of Apache, you'll need to uncomment the lines in the `httpd.conf` configuration file to enable the `LoadModule` command to load the `php4_module`.

# The Ganglia Web Package

The Ganglia Web package allows you to view a current snapshot of the performance metrics stored in the RRDtool round-robin database discussed earlier. We'll divide the Ganglia Web package into two sections: *title* and *node snapshot*. Let's look at each in turn.

**NOTE** Our example Ganglia web page comes from the cluster named *IBM Cluster* at the *Berkeley Millennium project* (<http://monitor.millennium.berkeley.edu>). You can call up this public web page from your web browser to see a live demonstration of the figures we'll use in this chapter.

## The Title Section

The Ganglia Web package title section is shown in Figure 18-1. By default, the package uses the Full View to display the cluster (click the Physical View link to see the same data in a different format). The data displayed is a snapshot of the performance metrics collected by *gmond*. To refresh the data, click the **Get Fresh Data** button.

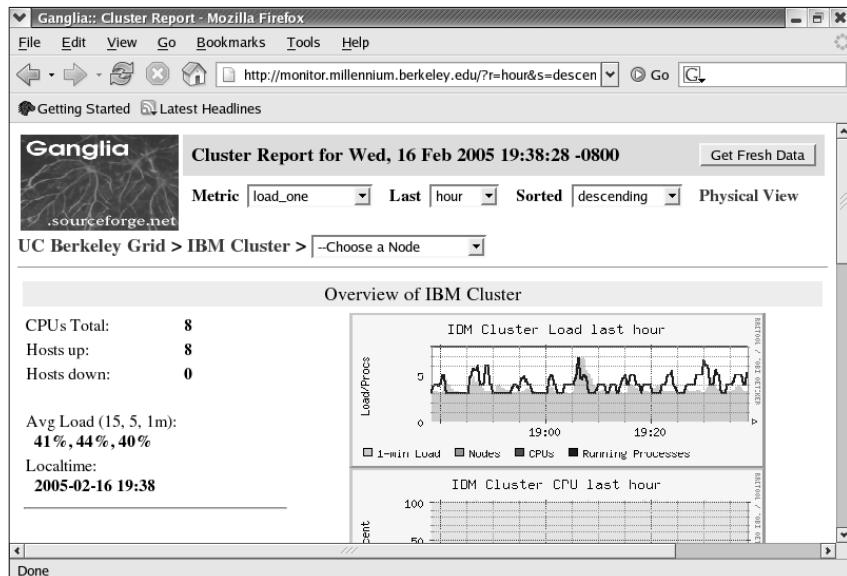


Figure 18-1: The Ganglia Web package title section

**NOTE** The webfrontend pages refresh every 300 seconds (5 minutes) by default. This can be modified in *./config.php* as with all Ganglia Web parameters.

The pull-down lists in the title section control what is displayed in the overview and snapshot sections. The Last menu lets you select up to a year's worth of data that will be graphed in the overview section. When you click the Metric pull-down list, you'll see the list of collected performance metrics. The default metric is *load\_one*, which is the one-minute *load average* on the cluster nodes. (We'll discuss this shortly.)

Also notice that you can specify a sort order for the output of the metric you select, which is used in the display of the nodes in the node snapshot section.

The last line of text in the title section allows you to select a different cluster to monitor if you have configured your gmetad daemon to monitor the gmond daemons on more than one cluster. In this example, the *group* of clusters monitored can be seen when you click the Grid link. As you can see in the figure, the IBM Cluster is selected and will be used to display the information contained in the next two sections of the web page.

## The Node Snapshot Section

The node snapshot section of the Ganglia Web page (Figure 18-2) contains one graph for each cluster node, representing the metric and time period you selected in the title section. The graphs are sorted in the order you select in the title section: descending, ascending, or by host name.

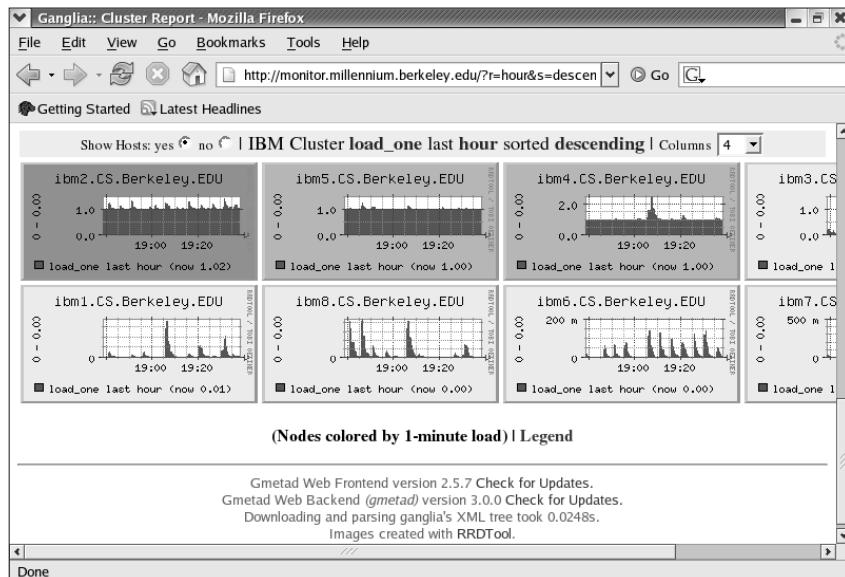


Figure 18-2: The Ganglia Web package node snapshot section

The node snapshot section allows you to examine the history of one performance metric on all of the cluster nodes. The graphs in Figure 18-2 show the load\_one performance metric for the previous hour for each cluster node. Let's examine the load\_one (one-minute load average) performance metric in more detail.

### Load Average

Inside the kernel,<sup>6</sup> the 1-, 5-, and 15-minute load average is recalculated automatically every five seconds based on the current system load.

<sup>6</sup> See sched.h in the kernel source tree.

The current system load is the number of processes that are running on the system plus the number of processes that are waiting to run<sup>7</sup> (in other words, processes in the “TASK\_RUNNING” and “TASK\_UNINTERRUPTIBLE” state). On a lightly loaded system (a system that is not processor bound), the current system load fluctuates between 1 and 0.

**NOTE** *You can also use the utilities `uptime`, `w`, and `procinfo` from the shell prompt to see the 1-, 5-, and 15-minute load averages.*

The kernel uses three numbers to store the 1-, 5-, and 15-minute load averages in memory; it does not keep any historical information about the system load except these three numbers in memory. Thus, to make the three load averages represent an *average* of the system load for the three time intervals (1, 5, and 15 minutes), a sophisticated mathematical formula called an *exponentially smoothed moving average function*<sup>8</sup> is used to apply a portion of the current system load toward increasing or decreasing the load averages, depending on whether the current system load is higher or lower than the load averages that were calculated 5 seconds ago. The longer the time interval represented by the load average value, the lower the impact the current system load has on the new load averages. Thus, a sudden increase or drop in the current load affects the 1-minute load average more than it affects the 5- and 15-minute load averages.

The load average is not the same as the CPU utilization, even though the CPU is usually 100 percent busy when the load average is 1. For example, if a cluster node is idle except for three concurrently running processes that each consume about 33 percent of the total CPU time, the CPU utilization is 100 percent and the load average is 3.<sup>9</sup> Or, to take another example, if an idle system with a load average of 0 starts running one CPU-bound process, the CPU utilization is immediately 100 percent, but the 1-minute load average will only be approximately .5 after 30 seconds and won’t reach 1 until about one minute.

### Red Cluster Nodes

When you see a node turn red in the Ganglia Web overview section, it means the per-CPU load average on a cluster node is greater than 1. When this happens, one or more processes on a cluster node are processor bound; that is, they would run faster on a cluster node that was less heavily loaded or on a cluster node with a faster CPU.

In the Linux Enterprise Cluster, where users running reports compete with users doing data transactions, a red cluster node in the overview section may simply indicate that a report is running on one of the cluster nodes. If *all* of the cluster nodes are red, however, your users are waiting for CPU resources (even the users who do data transactions), and they would therefore benefit from additional or faster cluster nodes.

<sup>7</sup>Process in the run queue or a short-term sleep state.

<sup>8</sup>See <http://www.teamquest.com/html/gunther/ldavg1.shtml> for complete details.

<sup>9</sup>That is, the one-minute load average is 3 if the processes have been running for more than a minute.

## Examining the Cluster Node from the Ganglia Web Package

If you click a node in the Ganglia Web package overview section, your browser will display the Ganglia Web Host Report web page for the node. The Host Report web page contains all of the information that has been collected by gmond for *one* cluster node. Like the main Ganglia Web web page, the Host Report web page contains a title section that allows you to select the period of time you would like to use for the graphs presented on the page (you can select hour, day, week, month, or year), as shown in Figure 18-3.

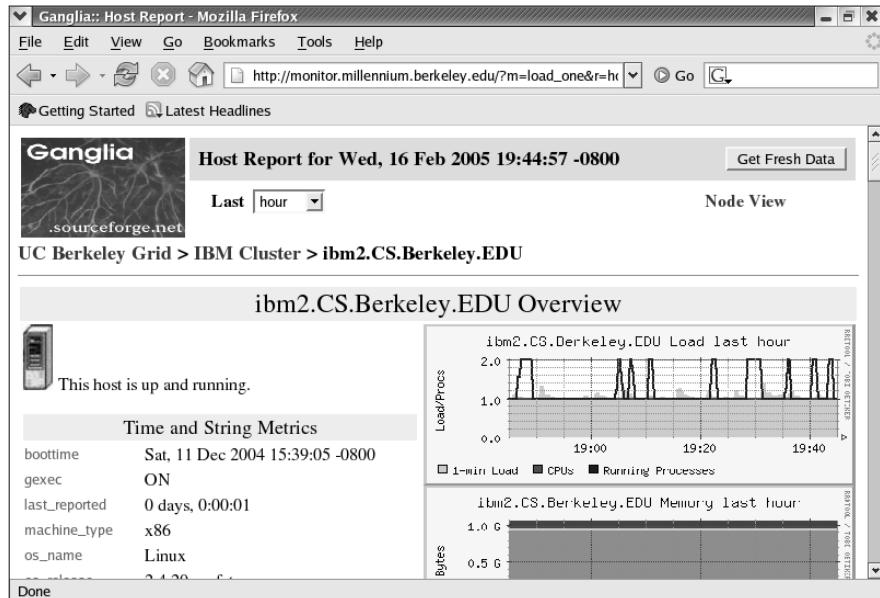


Figure 18-3: The Host Report Title Section

The next section of this web page is the Host Report overview section. As shown in Figure 18-3, this section summarizes the string and constant metrics, and it displays three graphs of the CPU load and CPU and memory utilization on the cluster node. When the string metric `gexec` is set to `ON`, the `gexec` batch jobs can be run on the cluster node. This screen also lists the version of the Linux kernel running on the cluster node and displays several basic pieces of information about the cluster node.

The constant metrics values, such as the total amount of memory, the number of CPUs in the system, and processor speed, normally do not change without a system reboot.

The next section of the Host Report web page shown in Figure 18-4 (again, we are now looking at just one cluster node) contains one graph for each volatile system metric you collect using the `gmond` daemon. (I'll describe how to add your own metric to this list later, in the "Creating Custom Metrics with `gmetric`" section in this chapter.) When you select a time period in the

Host Report title section that is greater than one hour—such as a day or week—these graphs can help you to spot trends in system hardware resource utilization during periods of peak system load so that you can determine whether additional hardware is required to meet the processing needs of your users.

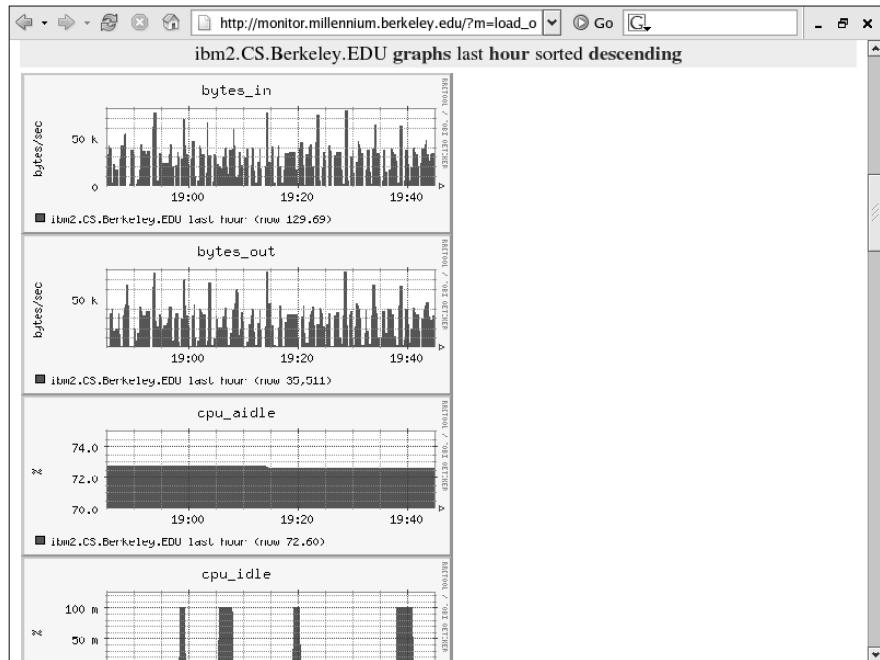


Figure 18-4: The Host Report overview section

While we won't discuss in detail how to interpret all of the data you see on these graphs, we will describe how to create and interpret a custom metric in the "Creating Custom Metrics with gmetric" section in this chapter.

### **Examining the Cluster Node from the Shell Prompt**

When a node turns red, meaning its load average per CPU is greater than 1, you can examine the collected performance metrics with the Ganglia Web page, or you can log on to the cluster node and run `top` to see a list of processes that are consuming the most CPU time. The `top` command ranks the running processes based on the amount of CPU time each is consuming and automatically refreshes the list every five seconds. To investigate a particular process, run the `strace` command against the PID of the process:

---

```
#strace -p 3456
```

---

where 3456 is the PID of the process that is hogging the CPU. `strace` will display the calls made by this process on your screen. To learn more about these system calls, look at the man pages for each. Looking at these manual

pages may help you to see if the process is stuck in a loop and not doing anything other than wasting CPU cycles.

**NOTE** *If the process is functioning normally, see if it can be moved to a dedicated cluster node or scheduled to run after hours, so that it will not affect other users contending for CPU time.*

### ***gstat***

Installation of the gmond RPM also installed gstat on your cluster nodes. You can use gstat to list information about the cluster if you need to manually decide which node has the least load. To try it out enter:

---

```
#gstat
```

---

This command outputs a list of cluster nodes based on gmond's knowledge of the cluster. The node with the least load is listed first. (Enter gstat -h for a list of the command line switches gstat supports.)

**NOTE** *You won't be using gstat for maintenance or monitoring. I mention it for the sake of completeness.*

### ***Running a Command on the Least-Loaded Cluster Node Using gstat***

You can use the secure shell (ssh) and the output of gstat to run a command on the least-loaded cluster node, as shown in the following example:

---

```
#ssh `gstat -m | head -1 | awk -F: '{print $1}'` uptime
```

---

This command can be executed from any cluster node, and ssh will run the command you enter on the least-loaded node (the first node returned by the gstat command). In this example, the command `uptime` will run on the cluster node, and the output of the command will be displayed on your screen. (See Chapters 4 and 19 for a more detailed discussion of ssh.)

## **Creating Custom Metrics with gmetric**

If you want to use Ganglia to report and graph a performance metric that is not included with the standard Ganglia distribution, you can write your own performance monitoring script to report to the locally running gmond on each cluster node with gmetric. You'll find sample gmetric scripts in the Gmetric Script Repository available online (<http://ganglia.sourceforge.net/gmetric>).

The following is a sample custom gmetric script to create four NFS client performance metrics.

---

```

#!/bin/bash
#
# Linux NFS Client statistics
#
# Report number of NFS client read, write and getattr calls Because we were
last called.
#
# (Use utility "nfsstat -c" to look at the same thing).
#
# Note: Uses temp files in /tmp
#

# GETATTR
if [ -f /tmp/nfsclientgetattr ]; then
    thisnfsggetattr=`cat /proc/net/rpc/nfs | tail -1 | awk '{printf "%s\\
n", $4}'`^
        lastnfsggetattr=`cat /tmp/nfsclientgetattr`^
        let "deltagetattr = thisnfsggetattr - lastnfsggetattr"^
        # echo "delta getattr $deltagetattr"^
        /usr/bin/gmetric -nnfsggetattr -v$deltagetattr -tuint16 -ucalls
fi

# READ
if [ -f /tmp/nfsclientread ]; then
    thisnfsread=`cat /proc/net/rpc/nfs | tail -1 | awk '{printf "%s\\
n", $9}'`^
        lastnfsread=`cat /tmp/nfsclientread`^
        let "deltaread = thisnfsread - lastnfsread"^
        # echo "delta read $deltaread"^
        /usr/bin/gmetric -nnfsread -v$deltaread -tuint16 -ucalls
fi

# WRITE
if [ -f /tmp/nfsclientwrite ]; then
    thisnfswrite=`cat /proc/net/rpc/nfs | tail -1 | awk '{printf "%s\\
n", $10}'`^
        lastnfswrite=`cat /tmp/nfsclientwrite`^
        let "deltawrite = thisnfswrite - lastnfswrite"^
        # echo "delta write $deltawrite"^
        /usr/bin/gmetric -nnfswrite -v$deltawrite -tuint16 -ucalls
fi

# NFS Quality Assurance RATIO (nfsqaratio)
# If this value shrinks too much then perhaps an application
# program change introduced excessive GETATTR calls into production.
if [ "$deltagetattr" -ne 0 ]; then
    let "nfsqaratio = (deltaread + deltawrite) / deltagetattr"
    /usr/bin/gmetric -nnfsqaratio -v$nfsqaratio -tuint16 -ucalls
fi

```

```
# Update the old values on disk for the next time around. (We ignore
# the fact that they have probably already changed while we made this
# calculation).
cat /proc/net/rpc/nfs | tail -1 | awk '{printf "%s\n",$9}' > /tmp/
nfsclientread
cat /proc/net/rpc/nfs | tail -1 | awk '{printf "%s\n",$10}' > /tmp/
nfsclientwrite
cat /proc/net/rpc/nfs | tail -1 | awk '{printf "%s\n",$4}' > /tmp/
nfsclientgetattr
```

---

Notice that this script calculates four custom NFS client performance metrics and then stores them in variables (`deltagetattr`, `deltaread`, `deltawrite`, `nfsqratio`). It then reports these values using `gmetric`. The script divides the sum of the NFS read and write calls by the number of `GETATTR` calls to produce an NFS quality assurance ratio (`nfsqratio`). As the value of the `nfsqratio` gets bigger, there are fewer NFS `GETATTR` calls relative to the total NFS read and write calls—which is a good thing—but if this value shrinks while the system is busy doing numerous NFS read and write operations, it means that programs using the NFS filesystem are causing excessive NFS `GETATTR` calls. (As discussed in Chapter 16, an excessive number of `GETATTR` calls will cause your network filesystem to perform poorly.)

We'll see how to schedule this script as a batch job for execution on all cluster nodes in the next chapter.

## In Conclusion

In this chapter, I provided a brief introduction to the sophisticated Ganglia package. Ganglia uses the `RRDtool` to store performance metrics from all of the cluster nodes into a round-robin database on the cluster node manager. The Ganglia Web package displays the cluster nodes through an Apache web server using colors to graphically represent the load average on each cluster node and to display performance metrics collected by `gmond`. This allows you to quickly spot problems on the cluster nodes and to drill down into the detailed performance metric information for a specific cluster node by simply selecting that node.

# 19

## CASE STUDIES IN CLUSTER ADMINISTRATION



Cluster administration is more complex than system administration because a cluster has multiple points of control and multiple points of failure. In a cluster, for example, there is more than one place where a print job can get stuck and more than one place where a user's logon can fail. The cluster administrator needs to know where all of these points of control and points of failure are in order to troubleshoot problems.

Because Linux Enterprise Cluster administration is a new discipline, best practices for Linux cluster administration have not yet evolved. In this chapter, I will therefore describe several case studies in Linux cluster administration:

- Administering accounts without Active Directory
- Building a fault-tolerant print spooler
- Rebooting nodes for preventative maintenance
- Sending and receiving email in a cluster environment
- Remote administration of cluster nodes

## Administering Accounts Without Active Directory

The cluster administrator must decide which account validation method to use to grant users access to the cluster and to the shared data inside the cluster.

For example, if an organization already uses Microsoft's Active Directory product for its email and Windows desktop systems, the system administration staff will naturally be inclined to use Active Directory for cluster account validation. But careful consideration during the design phase of the cluster implementation project should be given to the potential problems that using Active Directory could add to the cluster—in terms of both complexity and reliability.

Using Active Directory for account validation will introduce an additional proprietary high-availability system (assuming you use Microsoft's recommended methodologies for deploying Active Directory) *on top of* the open source high-availability software stack that was described in Part II of this book. The cluster administrator will have to know both, and if either one fails, the cluster will appear to be inoperable. If Active Directory topples, in other words, it will take the cluster with it. The added complexity of supporting two high-availability systems is therefore likely to *decrease* the reliability of the cluster.

Fortunately, there are at least three account validation methods available on Linux that have been used in Unix environments for years: local password files, NIS,<sup>1</sup> and LDAP. There are benefits and drawbacks to each of these methods, and no single method is best for all situations. In this case study, we will explore how to build a cluster account validation method that is based on all three of these methods.

**NOTE** *If you build a Zope (or Plone) web cluster (see Chapter 20), you will very likely use the password authentication mechanism built into the Zope Content Management Framework (CMF) instead of one of these operating system methods of user authentication.*

### Legacy Unix Account Administration Methods: The Problem

The NIS and LDAP systems can be deployed using a master server and multiple slave servers that can take over for the master server if it crashes. The problem with these methods is that they can be difficult to set up and cumbersome to administer. (However, many organizations may have them already, so for them, adding a Linux Enterprise Cluster that uses these methods is easy.)

The NIS and LDAP servers can also be placed under Heartbeat's control so that a catastrophic failure of the primary, or master, account authentication server will cause the account authentication *resource*<sup>2</sup> (NIS or LDAP) to failover to a backup server. The benefit of doing this is that the system administrator only needs to understand one high-availability technique for

---

<sup>1</sup> Either the NIS or the NIS+ system.

<sup>2</sup> See Part II of this book for a discussion of Heartbeat resources.

all services (DNS, NIS, LDAP, email, MySQL, and so on). However, the lone primary NIS or LDAP server can then become a performance bottleneck for the cluster if it is running legacy Unix applications (user credentials may need to be examined more than once to complete a single transaction, for example).

### ***The Best of Both Worlds***

In this case study, we will therefore see how to deploy an account authentication system that will support the Linux Enterprise Cluster for an organization that does not already use NIS or LDAP. This hypothetical organization has deployed Active Directory for Windows users but would like to use a different system on its mission-critical business system. Because this organization has several hundred users, a centralized method of administering the accounts is required, but the mission-critical application running on the Linux Enterprise Cluster will be a legacy Unix application that requires multiple account validation operations to complete a single business transaction. Thus, using the local password files is the preferred method. (In fact, this is the method that is currently used on the monolithic Unix server that is running the mission-critical application that the organization would like to move to the Linux Enterprise Cluster.)

## ***Building a Reliable Cluster Account Authentication Mechanism***

The system administrator would like to use a central LDAP server to administer all of the users accounts, but would also like to use the local `/etc/passwd` and `/etc/group` files on each cluster node to improve the reliability and performance of the account authentication system. The system administrator also needs to support a few Unix servers that are currently NIS clients of an existing NIS system that is used along with the local `/etc/passwd` file on the existing Unix monolithic server. The Linux Enterprise Cluster in this case study will be protected by a firewall, and only internal employees will be allowed to access it.

The goal of this case study, then, is to make the `/etc/passwd` (and `/etc/group`) file the same on all of the cluster nodes by using a simple script that will pull the account and group information out of the LDAP server. This script can then run as a regularly scheduled cron job on all of the cluster nodes.

### ***Using the Local `passwd` and `Group` File on Each Cluster Node***

Unix and Linux systems normally store user accounts in the `/etc/passwd` file and encrypted user passwords in the `/etc/shadow` file. The historical reason for doing this is to protect the encrypted passwords by “hiding” them from users in a file that only the root account can access (so users can’t grab the encrypted passwords and run a tool like Crack that will guess the password).

In this case study, however, the cluster nodes sit behind a firewall, and the system administrator does not allow users access to a shell (where they might be able to run the crack program to guess encrypted passwords), so the encrypted passwords do not need to be stored in the `/etc/shadow` file. (Linux will use an encrypted password if it finds one in the second field in the `/etc/passwd` file instead of consulting the `/etc/shadow` file.) We can therefore meet the needs of this organization by writing a simple script that pulls the user and group information out of the LDAP server and places it into the local `/etc/group` and `/etc/passwd` file on each node, and we need only update the `/etc/nsswitch.conf` file on the cluster nodes to tell them to use their local files for accounts and groups.

**NOTE** *See <http://webmin.com> and <http://symark.com> for two additional methods of distributing user accounts to each cluster node.*

## ***A Simple Script***

The script that runs on each cluster node could pull the user and account information out of the LDAP server using the `ldapsearch` command, but because this organization also needs to continue to support NIS clients, it has decided to purchase the NIS/LDAP gateway from PADL software (<http://www.padl.com>). With this software installed on the high-availability LDAP server (the cluster node manager running the LDAP server as a resource under Heartbeat's control), the cluster nodes can run the `ypbind` daemon and communicate with the LDAP server as if it were an NIS server. The script can therefore use the familiar `ypcat` command to access the user and group information that is stored on the LDAP server, even though the server is really an LDAP server and the `/etc/nsswitch.conf` file on the cluster nodes points to the local `/etc/passwd` and `/etc/group` files. The script then only needs to contain the following two lines:

---

```
ypcat passwd > /etc/passwd
ypcat group > /etc/group
```

---

These commands will overwrite all of the `/etc/passwd` and `/etc/group` entries on the system with the information stored on the NIS (LDAP) server, so be sure to create all of the normal system accounts (especially the `root` user account) on the LDAP server.

One downside to this method is that a glitch in the network or NIS system may cause corrupted account information to overwrite the local `/etc/passwd` file on all cluster nodes. Thus, an even better way to apply the changes to each node is with the `patch` and `diff` commands; for example, with a shell script like this:

---

```
ypcat passwd > /tmp/passwd
diff -e /etc/passwd /tmp/passwd > /tmp/passwd.diff
patch -be /etc/passwd /tmp/passwd.diff
```

---

Because these commands use the ed editor (the -e option) to modify only the changed lines in the /etc/passwd file, this script can be run at any time without affecting users logged on to the system. To avoid corrupting the /etc/passwd file with garbage data, have this script check to make sure that the NIS server is operating normally with a test like this:

---

```
ypcat passwd > /tmp/passwd

if [ ! -s /tmp/passwd ]; then
    logger "Empty password file retrieved from LDAP. Aborting."
    exit 1
fi
```

---

The if statement in this bash shell script code will check to make sure the file that contains the passwd lines that were downloaded from the LDAP server is not empty. This test should be done before the patch command is used to apply the changes (if there are any) to the local passwd file as described in the preceding paragraphs.

As you can see from this case study, choosing the right account administration technique for your Linux Enterprise Cluster will depend on your situation.

## Building a Fault-Tolerant Print Spooler

In this case study, we'll build a central print spooler that allows all of the cluster nodes to send their print jobs through a central print spooler on the cluster node manager. As with the preceding case study, the "right" choice for the printing software package you choose and for the architecture of your printing system will depend upon your situation. The hypothetical organization that we will use in this case study has a very heavy volume of print jobs that are sent to laser printers located on the internal company LAN.

Let's look at how to install a printing system that will meet the needs of this organization and provide a single point of control for print jobs sent from the cluster nodes without creating a single point of failure.

### **Cluster Nodes and Job Ordering**

For proper job ordering, print jobs should be sent to this central print server as quickly as possible. If, for example, user A on node 1 sends 100 print jobs to a printer, then shortly thereafter user B on node 2 sends another 100 print jobs to the same printer, we want user A's jobs to finish printing before B's jobs start. This is normally not a problem with a monolithic server because it takes only a few seconds for user A's application to finish sending the jobs to the local print queue. However, in a cluster environment, *as with any distributed printing environment with a central print spooler*, the jobs may take more time to travel from the cluster node to the central print server. This

introduces the possibility that another application on another cluster node printing to the same printer will begin interleaving print jobs. Due to the large number of print jobs used by our hypothetical organization, we'll need to take care when configuring a printing system for the cluster so that print jobs do not sit for very long in the queue or spool directory on the cluster nodes.

## **LPRng: A Linux Enterprise Cluster Printing System**

While LPRng can be difficult to install and complex to use, it is quite powerful and a great choice for use as an enterprise cluster printing system.

**NOTE** *This section provides a sample printer configuration and a few sample printer commands. It is not meant to replace the extensive documentation written by Dr. Patrick Powell and available online in the LPRng-HOWTO and the IFHP-HOWTO.*

### **Cluster Nodes and Print Jobs**

As Figure 19-1 shows, applications running on cluster nodes create print jobs and send them to the locally running `lpd` daemon on the cluster node. The `lpd` daemon places each print job onto the local hard drive of the cluster node.<sup>3</sup> While the `lpd` daemon on the cluster node could send the jobs directly to the printers, for better administration of the print queues and printer configurations, all print jobs will be sent to a central print server—the highly available cluster node manager. The central print server will then save, or spool, the print jobs again onto its local hard drive before sending them on to the printer.

As you can see in Figure 19-1, each cluster node is storing print jobs locally before passing them on to the central print server. As previously mentioned, the cluster nodes should hold their print jobs in the local spooler on each cluster node and then send them to the central print spooler as quickly as possible to avoid job interleaving problems when multiple cluster nodes print to the same printer at the same time. Also, as you can see in the figure, the central print spooler is where the connection is made to the actual print device—the laser printer in this case. If this central print spooler fails (though it shouldn't, because it will be built as a resource under Heartbeat's control), the print jobs will wait in the print spooler on each cluster node until the central print spooler returns to normal operation.

---

<sup>3</sup> LPRng clients need not run the `lpd` daemon. For example, if you need to print very large GIS files, you do not need to spool them to the local hard drive on the cluster node before sending them to the central print spooler.

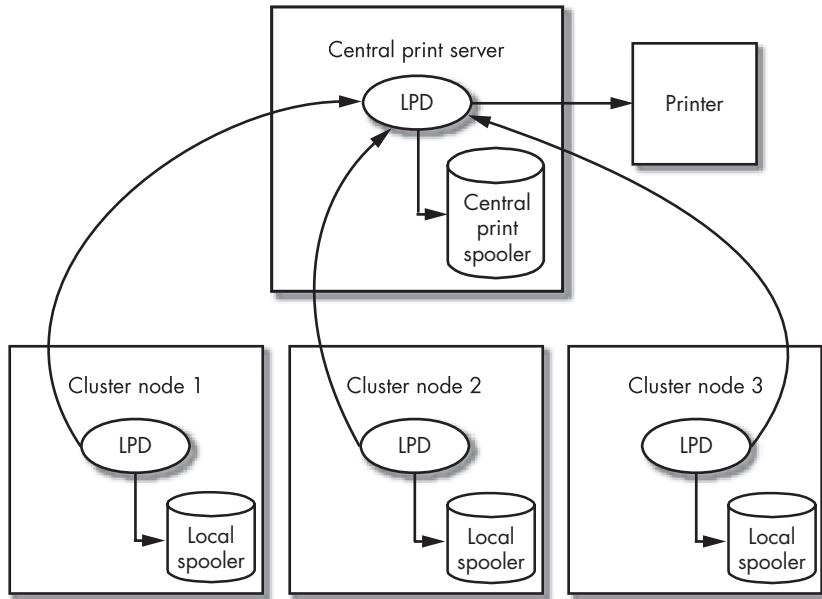


Figure 19-1: LPRng in a cluster environment

### **Building the Cluster Printing System Based on LPRng**

Now that you know which printer software package you want to use (LPRng) and how you would like to configure it, you are ready to install the software on each cluster node and on the central print spooler (the cluster node manager).

To build the configuration shown in Figure 19-1, first download and install the “stock” version of the LPRng package (version 3.8.20 or later) on the cluster node manager and on all of your cluster nodes.

#### **Install LPRng on the Central Print Server**

Once you’ve done that, you can modify the standard `/etc/init.d/lpd` initialization script *on the central print server* so that it no longer runs the `printconf-backend` program from Red Hat and instead just starts up the “stock” LPRng `lpd` daemon. The `start` section of the `/etc/init.d/lpd` init script need only contain the following:

---

```
Start () {
    /usr/sbin/checkpc -f
    daemon /usr/sbin/lpd
}
```

---

The `checkpc` program is part of the LPRng package and checks the printer configuration. If any small problems are found, such as directory permissions not set correctly on the spool directories, the `checkpc` program

attempts to fix them automatically if you pass it the `-f` argument. Normally, you use this command after making a change to the printing system.

### ***Install LPRng on the Cluster Nodes***

On the cluster nodes, however, you do not need to modify the `/etc/init.d/1pd` script. You can use the standard version supplied by Red Hat or your distribution. When the `printconf-backend` script runs (each time the script is called with the `start` argument), the `/etc/printcap` file is overwritten on the cluster node. Thus, you never modify the `/etc/printcap` file on the cluster nodes. Instead, each cluster node knows what printers are available on the central print server because the printers are listed in the `/etc/printcap.local` file on each cluster node (of course, this can be a symbolic link to a single file stored on an NFS-mounted drive, so you only need to make changes to one file for all cluster nodes to see the change). Each time you modify the `/etc/printcap.local` file, you must restart the `1pd` daemon on all cluster nodes for the change to take effect.

**NOTE** *On the central printer server (with the modified `/etc/init.d/1pd` init script), you always modify the `/etc/printcap` file, because the `/etc/printcap.local` file is not used on the central print server.*

To summarize the configuration we've just built: the `/etc/printcap` file on the central print server contains the actual printer definitions (what interface script to use for a particular printer, what IP address location the printer has, and so on), and the `/etc/printcap.local` file on the cluster nodes contains only enough information for the cluster nodes to be able to send the print jobs to the central print server (how long the network timeouts should last when communicating with the central print server, the host name of the central print server, and so forth).

### ***Modify the `/etc/printcap.local` File on the Cluster Nodes***

An entry in the `/etc/printcap.local` file on each of the cluster nodes looks like this:

---

```
# All printers on the cluster nodes can use these same default values:
.default:
:fifo
:sd=/var/spool/lpd/%P
:lp=%P@centralprintserver
:connect_interval=60
:connect_grace=3
:reuse_addr=true
:send_job_rw_timeout=0
:send_query_rw_timeout=0
:send_try=0
```

```
:retry_econnrefused=true
:retry_nolink=true
:wait_for_eof=true
:connect_timeout=600
:force_localhost

# Then place each printer after the default values in the
# file (/etc/printcap.local) with entries like this:

printer1:tc=.default
printer2:tc=.default
printer3:tc=.default
```

---

where *centralprintserver* in this example is the host name of the central print server (and is resolved using DNS/LDAP/NIS or the local /etc/hosts file on each cluster node).

### ***Modify the /etc/printcap File on the Central Print Server***

On the LPRng central print server an /etc/printcap entry looks like this:

---

```
# A list of definitions for laserjet style printers that will use the
# "ifhp" print filter.
```

```
.laserjet
:fifo
:filter=/usr/libexec/filters/ifhp
:ifhp=waitend@
:af=acct
:lf=log
:mx=0
:sd=/var/spool/lpd/%P
:sh
```

```
# The following printers use the "laserjet" definition
# above. Here are their IP addresses.
```

```
printer1
:tc=.laserjet
:lp=192.160.0.201%9100
```

```
printer2
:tc=.laserjet
:lp=192.160.0.202%9100
```

```
printer3
:tc=.laserjet
:lp=192.160.0.203%9100
```

---

Of course, you can have more than one type of printer—a printer type other than laserjet—defined at the top of the `/etc/printcap` file on the central print server.

**NOTE** *Because the standard Red Hat distribution uses a slightly modified version of the LPRng system, you may see references to the “printconf-tui” (tui stands for text user interface) and “printconf-gui” (the graphical user interface version of printconf available under an X Windows display) tools. Do not use either of these printconf utilities on the central print server.*

This case study will conclude with a look at the day-to-day activities the system administrator will use to administer print jobs on the cluster using the LPRng printing system.

### ***Managing Print Jobs on the Central Print Server***

Print queues and print jobs can then be managed with the `lpc` command, as follows, on the central print server.

List all printers and their current status:

---

```
#lpc status all
```

---

Hold all jobs currently waiting to print on printer1:

---

```
#lpc hold printer1 all
```

---

Release jobs waiting to print on printer1:

---

```
#lpc release printer1 all
```

---

Place the printer in a hold state so new jobs can still be sent from the cluster nodes, but the central print server will not send them on to the printer:

---

```
#lpc holdall printer1
```

---

Take the printer out of the “hold” state:

---

```
#lpc noholdall printer1
```

---

When a printer hangs, use the following command (print jobs are not removed or lost, but the subserver process started by the main `lpd` daemon to process the jobs within the print queue will be killed and a new one restarted):

---

```
#lpc kill printer1
```

---

## Managing Print Jobs from the Cluster Nodes

The same commands can be used by users<sup>4</sup> on the cluster nodes with the added -S option:

---

```
#lpc -Scentralprinterserver hold printer1 all
#lpc -Scentralprinterserver holdall printer1
#lpc -Scentralprinterserver release printer1 all
#lpc -Scentralprinterserver noholdall printer1
```

---

When the lpc command is used from a cluster node without the -S argument, the command only affects the print queue on the cluster node (the local print queue on the cluster node). Forgetting to use the -S argument to the lpc command when you issue the holdall command for one of the printers on a cluster node can lead to the situation where a printer stops working for some users but not for all of the users. When this happens, the print jobs are not lost, however, and the cluster administrator need only log on to the cluster node and release the print jobs.

### CREATE CUSTOM SCRIPTS ON THE CENTRAL PRINT SERVER

One of the useful features of the LPRng printing system is that you can write your own scripts to handle print jobs. The script can contain a line as simple as this:

---

```
cat > /tmp/job.$$
```

---

This line will take the input stream of characters (normally PostScript) and send them to the file /tmp/job.pid, where pid is the process ID of your custom script when it is launched by the lpd daemon on the central print server. You can then manipulate the file (/tmp/job.\$\$) within your script to do things like send the job as a fax (using the Hylafax program<sup>1</sup>) or convert the file into PDF using the ps2pdf utility and store the file in an SQL database as a *binary large object* (*blob*).<sup>2</sup>

Note: Windows PC users can print to the LPRng printers using the Samba package. See <http://www.samba.org>. (Samba is probably already included with your distribution.)

---

<sup>1</sup> See the smbfax project.

<sup>2</sup> You can then later retrieve the blob using a variety of methods such as the Perl DBI package or a Python script in conjunction with the Zope or Plone packages for display on a web server.

---

<sup>4</sup> The LPRng system has a security system to control which commands a user can apply to a printer queue. See the LPRng documentation for details.

## Rebooting Nodes for Preventative Maintenance

In this case study, we'll examine one method of performing preventative maintenance on the cluster. The cluster was built as described in Part III of this book, using ldirectord in conjunction with the Heartbeat program to make the LVS Director a highly available server (a primary and a backup server with no single point of failure). When built this way, there are several options for removing cluster nodes for maintenance. Cluster nodes can be removed by doing any one of the following:

- Modifying the contents of the health check file (`htdocs/.healthcheck.html` was used in the examples in Chapter 15) on the cluster node. ldirectord will then automatically remove the node from the cluster.
- Killing the daemon responsible for the port that ldirectord is monitoring on the cluster node. For, example if ldirectord is monitoring port 80 (the http port), simply stopping the `httpd` daemon on the cluster node will cause ldirectord to remove it from the cluster.
- Modifying the ldirectord configuration<sup>5</sup> file on the primary Director and removing the cluster node. If you have set the `autoreload=yes` option in this file, the change will be made automatically by ldirectord.
- Modifying the IPVS table on the Director using `ipvsadm` commands. If this method is used, the ldirectord configuration file should also be modified to match the changes made with the `ipvsadm` command.

**NOTE** *In this case study we will use the fourth method.*

### Using `ipvsadm` Commands to Remove a Cluster Node

There are two methods for removing a cluster node from the IPVS table using `ipvsadm` commands. The first is to set the weight of the cluster node to 0 (assuming you are using a weighted scheduling algorithm), and the second is to simply remove the server from the table.

### Changing the Weight of a Cluster Node to 0

By setting the weight of a cluster node, or real server, to 0, you deny any future clients access to the node. However, all of the existing connections remain unaffected. The purpose for using this method is to remove nodes for maintenance without affecting users. Unfortunately, this method does not always guarantee that you will have an unloaded node if users continue to reconnect to the cluster before their connection template entry expires (see Chapter 14), especially if you are using a large TCP session timeout value.

---

<sup>5</sup> Usually located in the `/etc/ha.d/conf` directory.

## **Disabling Telnet Access to One of the Cluster Nodes**

Clients only connect to the Linux Enterprise Cluster in this case study using telnet. To remove a node from the cluster, the cluster administrator will therefore only need to use the following command to edit the IPVS table on the LVS Director:

---

```
#ipvsadm -e -t 209.100.100.100:23 -r 10.0.0.1:23 -g -w 0
```

---

In this command, 209.100.100.100 is the VIP, 10.0.0.1 is the cluster node to remove, and :23 is the telnet port.

New users that connect to the cluster (the VIP) will not be able to access the 10.0.0.1 cluster node until its weight is set back to a number greater than 0. The hypothetical organization in this case study changes shifts at 5:00 p.m. each day. When all of the users log out at 5:00 p.m. and the new shift comes on duty, the system administrator can safely reboot the cluster node without affecting any of the users.

When Idirectord running on the Director sees the cluster node come back online after the reboot, it will automatically add it back into the IPVS table and make it available for the end-users.

## **Sending and Receiving Email in a Cluster Environment**

In this case study, we want to use an application on each of the cluster nodes that sends outbound email to customers. Each cluster node will therefore need to masquerade outbound email so it appears to come from a single domain using the following entry in the `/etc/mail/sendmail.mc` file on each cluster node:

---

```
MASQUERADE_AS(`yourdomain.com')dn1
FEATURE(`always_add_domain')dn1
FEATURE(`masquerade_envelope')dn1
FEATURE(`allmasquerade')dn1
```

---

We'll also send outbound email to a central (relay) email server, to make it easier to troubleshoot problems when email is not delivered properly, by adding the following line to the `/etc/mail/sendmail.mc` file on each cluster node:

---

```
define(`SMART_HOST', `mailserver.yourdomain.com')dn1
```

---

where `mailserver.yourdomain.com` is, of course, the host name of your SMTP server.

This `SMART_HOST` rule, however, will not send email to the central mail server when the mail is destined for a user with a `passwd` entry on the local cluster node (either in the local `/etc/passwd` file, LDAP, or NIS, and so on), and nothing is specified after the @ sign in the email address. An example

would be email sent to `joseph` or `sam` when Joseph and Sam have accounts on the cluster node. To force these email messages to the central mail server, you will also need to add:

---

```
define(`LOCAL_RELAY',`mailserver.yourdomain.com')dnl
```

---

If you need to make an exception and force some mail to be delivered locally (for example, email sent to `orderprocessing` that is really an email alias that sends email on to a program for automated processing) on the cluster nodes, you can use an entry like this:

---

```
LOCAL_USER(`orderprocessing')dnl
```

---

**NOTE** *The user `root` is always on the `LOCAL_USER` exception list.*

Once the changes have been made to the `sendmail.mc` file on each cluster node, you need to create a new `sendmail.cf` file with the command:

---

```
#m4 /etc/mail/sendmail.mc > /etc/sendmail.cf
```

---

Then, stop and restart the `sendmail` daemon on all cluster nodes.

#### TESTING EMAIL FROM THE COMMAND LINE

You can use the command `mail -v testuser@whereever.com` to test if this change makes any difference to the `From` address of an outbound email. This command will prompt you for a `Subject` line and then give you the opportunity to enter email message body text. Enter `.` on a line by itself to indicate the end of the message and press `ENTER` when asked for the `Cc:` email address. You'll see the transcript of the email session as `sendmail` running on the cluster node attempts to contact the remote mail server (or central relay email server, if you have used the `SMART_HOST` option just described) and the `From` address the cluster node is using.

## Creating a Batch Job-Scheduling System with No Single Point of Failure

In the final case study, we want to build a batch job scheduler with no single point of failure. In this case study, we again assume the cluster is protected by a firewall and security barriers are not required between the cluster nodes. (As with a monolithic server of old, a root compromise means the cracker will have access to everything.)

We'll first use `ssh` to construct a method of accessing a shell or running a command on any cluster node from a single machine.

## Run `ssh-keygen`

Recall from Chapter 4 that the `ssh-keygen` command can be used without a passphrase, so the `ssh` command can be used later without the need to key in a password each time. Run the following command on the cluster node manager:

---

```
#ssh-keygen -t rsa
```

---

Do not enter a passphrase when prompted. Now copy the contents of the file `/root/.ssh/id_rsa.pub` on the cluster node manager to each cluster node's `/root/.ssh/authorized_keys2` file.

## Modify the `sshd_config` File on Each Cluster Node

We are trusting in the ssh keys to protect the cluster from a root user attack (you have to know the private half of the RSA key used on the cluster node manager to connect as the root user to the cluster node). The root user account on the cluster node manager and the protection of the RSA key will now be as important as protecting the root password. You can use LIDS (<http://www.lids.org>) to hide this file for increased security, or you can create a separate cluster administrative account without root privileges.<sup>6</sup>

Modify the `PermitRootLogin` line in each cluster node's `/etc/ssh/sshd_config` file so it looks like this:

---

```
permitRootLogin without-password
```

---

Then start or restart the `sshd` daemon on each cluster node.

**NOTE** Make sure the `sshd` daemon is started each time the cluster nodes boot (see Chapter 1).

## Create the (RSA) `known_hosts` Entries on the Cluster Node Manager

Type a simple command such as the following on the cluster node manager:

---

```
#ssh c1node1 hostname
```

---

Run the command once for each cluster node, substituting `c1node1` in this example with the name of each cluster node. You should see a message like the following that will allow you to save the host key for each node on the cluster node manager.

---

<sup>6</sup> See the discussion of the `sudo` command in Chapter 4 for a method of giving special administrative account access to only certain root-level commands or capabilities.

---

```
The authenticity of host 'node1' can't be established.
RSA key fingerprint is 04:54:c1:f3:33:ff:89:14:a0:8d:f3:5d:67:f5:43:21.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'node1' (RSA) to the list of known hosts.
```

---

**NOTE** *If you are prompted for the password on each cluster node, review Chapter 4 for details on how to troubleshoot your ssh configuration.*

## **The Batch Job Scheduler**

We'll call the system where we launch batch jobs the *batch job scheduler*. The batch job scheduler in this case study is the cluster node manager (that now has ssh access to all of the cluster nodes). The batch job scheduler is a highly available server pair that places the cron daemon under Heartbeat's control and uses crontab<sup>7</sup> entries to submit batch jobs using the clustersh script that is included on the CD-ROM in the chapter19 directory. A sample cron table entry on the batch job scheduler looks like this:

---

```
30 14 * * * clustersh /clusterdata/scripts/myscript
```

---

This crontab entry says that every day at 2:30 p.m. the program /clusterdata/scripts/myscript should run on the cluster node with the least activity.

**NOTE** *The clustersh script contains the host names of the nodes inside your cluster. You'll need to modify this script file before you can use it on your cluster (see the script for detailed documentation).*

If a cluster node goes down, the clustersh script will not attempt to execute the /clusterdata/scripts/myscript on it.

To run the same script on all cluster nodes at 1:30 p.m. each day, the crontab entry would look like this:

---

```
30 14 * * * clustersh -a /clusterdata/scripts/myscript
```

---

By running cron jobs from the highly available batch job scheduler, you avoid a single point of failure for your regularly scheduled batch jobs.

**NOTE** *The /etc/ha.d/haresources file, in the Heartbeat configuration file on the batch job scheduler, should specify that the crond daemon is started only on the primary server and that it starts automatically on the backup server only if the primary server goes down.*

One danger of using this configuration, however, is that you may change the crontab file on the primary batch job scheduler and forget to make the same change on the backup batch job scheduler. To automate copying the

---

<sup>7</sup>The crontab file is the name of the file the cron daemon uses to store the list of cron jobs it should run. Edit the crontab file with crontab -e.

crontab file from the primary batch job scheduler to the backup batch job scheduler, use rsync from the following script, which copies the contents of the /var/spool/cron directory (where the crontab file resides):

---

```
#!/bin/bash
OPTS=" --force --delete --recursive --ignore-errors -a -e ssh -z "
rsync $OPTS /etc/ha.d/           backup:/etc/ha.d/
rsync $OPTS /etc/hosts           backup:/etc/hosts
rsync $OPTS /etc/printcap        backup:/etc/printcap
rsync $OPTS /var/spool/lpd/      backup:/var/spool/lpd/
rsync $OPTS /etc/mon             backup:/etc/mon
rsync $OPTS /usr/lib/mon/        backup:/usr/lib/mon/
rsync $OPTS /var/spool/cron/     backup:/var/spool/cron/
```

---

The last line in this configuration file tells rsync to copy the contents of the /var/spool/cron file to the backup batch job scheduler with the host name backup.

## In Conclusion

As with monolithic server administration, cluster administration is more of an art than a science, requiring an in-depth knowledge of the environment where the cluster will be deployed. Each of the cluster administration design and implementation decisions will impact the reliability and availability of the cluster, so precautions should be taken to avoid single points of failure even when single points of administrative control are required. In this chapter, we have examined a few of the common system administration tasks and seen case studies that describe how these administration tasks translate into a cluster environment.



# 20

## THE LINUX CLUSTER ENVIRONMENT



In this chapter, I'll provide a high-level review of the material presented in this book and a proposal for a new type of computing environment that is based on parallel processing and hardware redundancy using free software.

Before I begin, I should point out that I do not expect anyone who reads this book to have the opportunity to build a data center for a large organization from scratch. Large organizations already have data centers, and even small organizations that grow big enough to need a data center will have a legacy of information technology decisions that influence the development and design of their first data center. You will, therefore, probably not have free dominion over every design decision that lets you build the ideal data center or enterprise computing environment, no matter what your situation is. In this chapter, however, I would like to ask the question: "What would the ideal data center look like if you could build it?" My answer is based on my practical experience building the cluster I've described in this book. As you'll

see in this chapter, I believe the three foundations of the ideal data center are free software, low-cost commodity hardware, and a highly available Linux Enterprise Cluster.

## The Linux Enterprise Cluster

Let's start with the cluster. Remember that all cluster nodes run the same application programs (they offer the same services to the client computers). Console access to these nodes is made possible by a keyboard video mouse (KVM) device, as shown in Figure 20-1.

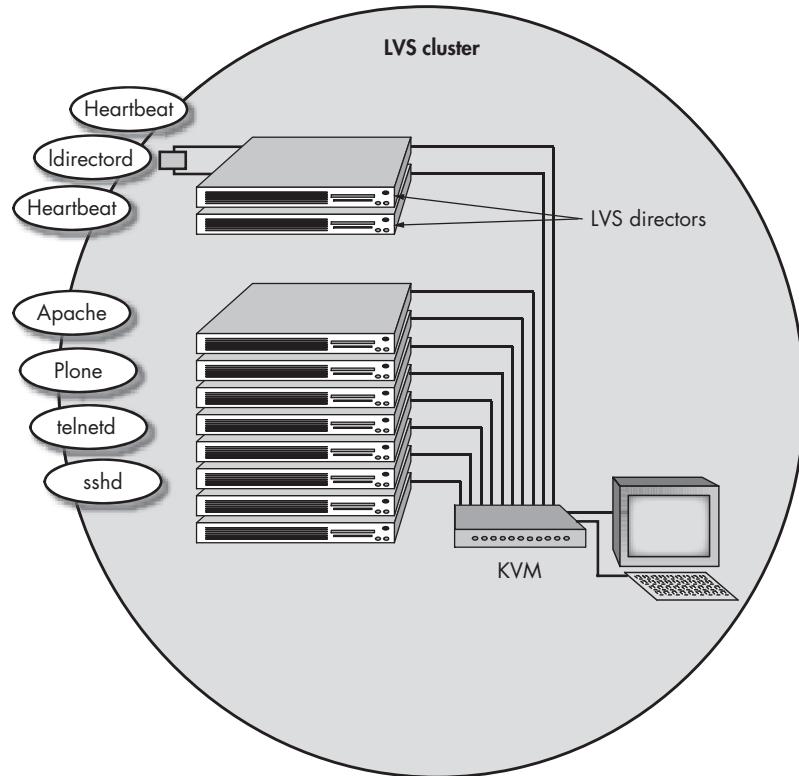


Figure 20-1: Figure of a ten-node cluster with a KVM device

The black box shown connected to the first two servers is a Stonith device that allows the primary LVS Director to reset the power to backup LVS Director. This hardware configuration is required as part of the Heartbeat package (a typical Stonith configuration will require two Stonith devices). Two computers act as a primary and backup load balancer and run the Heartbeat daemon. The `ldirectord` daemon runs on the primary LVS load balancer and removes nodes from the cluster when they fail. If the primary LVS load balancer crashes, Heartbeat running on the backup LVS load balancer starts the `ldirectord` daemon and begins load balancing the incoming requests from the client computers.

The `telnetd` and `sshd` daemons are also shown in Figure 20-1. Like the Apache daemon, they run on all cluster nodes to allow client computers to access the cluster. What about the business applications running inside the cluster? In the next section, I'll examine applications that run in the ideal data center on a cluster more closely.

## ***Applications Running on the Cluster***

The cluster runs objects that are the business tier of N-Tier applications. The term *N-Tier applications* refers to a conceptual model for breaking up an application program into layers in order to separate the application's functional components into basic categories that must exist for an application to do its job properly. The three most fundamental tiers in the N-Tier application model are the data, business, and presentation tiers. Briefly, the data tier is where the database server resides, the business tier is where the objects that define business rules reside, and the presentation tier is simply the user interface to the application (the user interface to the other tiers). So to refine the statement that this paragraph started with, I should say that an enterprise cluster is where *business tier objects run in parallel*. This is the same thing as saying that copies of the business tier object-oriented programs will run inside the cluster on all of the cluster nodes.

In a Linux Enterprise Cluster like business tier objects run in parallel; however, they do not know of each other's existence. Applications running inside the cluster, therefore, do not need to communicate with each other and objects that remain inside the cluster never need to be serialized. Shared data and stateful information about user sessions is stored in a session database outside the cluster. If a cluster node crashes, the load balancer reassigned user sessions to a new node, but the user sessions are persistent because the business tier objects running inside the cluster use the unique session ID as the key into the session database.

## **The Cluster Node Manager**

The cluster node manager sits outside the cluster and provides services to all of the cluster nodes. It helps to support the mission-critical applications running inside the cluster. For example, the cluster node manager may:

- Be the central account database for LDAP, NIS, or Active Directory accounts.<sup>1</sup>
- Be a central print server using the LPRng software.
- Run the Ganglia software to display colorful graphs of the cluster's activity.
- Run the `cron` daemon to schedule batch jobs for submission to the cluster.

---

<sup>1</sup> An LDAP server can act like an NIS server, using software from PADL software (<http://www.padl.com>), as discussed in Chapter 19.

- Run license management software (such as Flexlm).
- Run the Mon monitoring package to monitor cluster nodes.
- Act as a central fax server to send and receive faxes using Hylafax.
- Act as a DNS or email server for the cluster.

By using the high-availability techniques we've discussed in Part II of this book, we can build the cluster node manager to make these cluster server services highly available. (See Figure 20-2.)

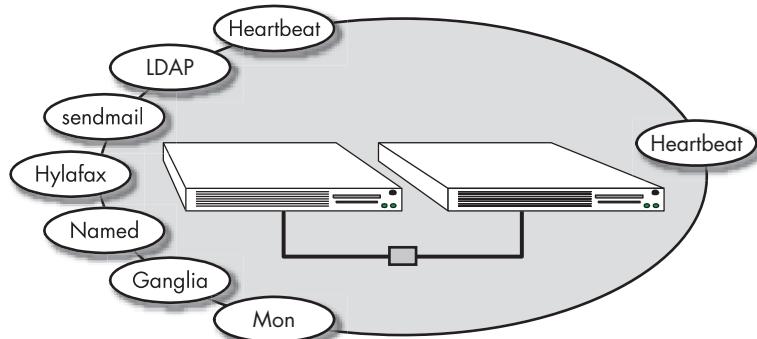


Figure 20-2: High-availability cluster node manager

Like the black box shown in Figure 20-1, the black box in Figure 20-2 represents a Stonith device. This device makes it possible for the backup cluster node manager to reset the power to the primary cluster node manager should it need to failover the resources. When this happens, the LDAP, sendmail, Hylafax, Named, Mon, and Ganglia (`gmetad` and `gmond`) daemons move to the backup server.

## The Clients

The end-user sits at a client computer and accesses the services offered by the cluster using code (objects) at the presentation tier. The client computer can be any type of operating system capable of running the presentation tier application. This includes desktop machines that contain a local copy of an operating system and all of the application code stored locally on their disk drives (fat clients) and desktop machines that boot from a centrally stored copy of the operating system where the application programs reside (Thin Clients). One type of Thin Client based on Linux is called the Linux Terminal Server Project (LTSP) Thin Client (see <http://www.ltsp.org>). LTSP Thin Clients run the LTSP client software on old PC hardware or small, diskless workstations designed to run LTSP (see <http://www.disklessworkstations.com>). The clients can then run X applications that run on Linux, such as the Mozilla web browser or the OpenOffice suite of applications. The LTSP server software runs on a pair of servers (to provide high-availability), as shown in Figure 20-3.

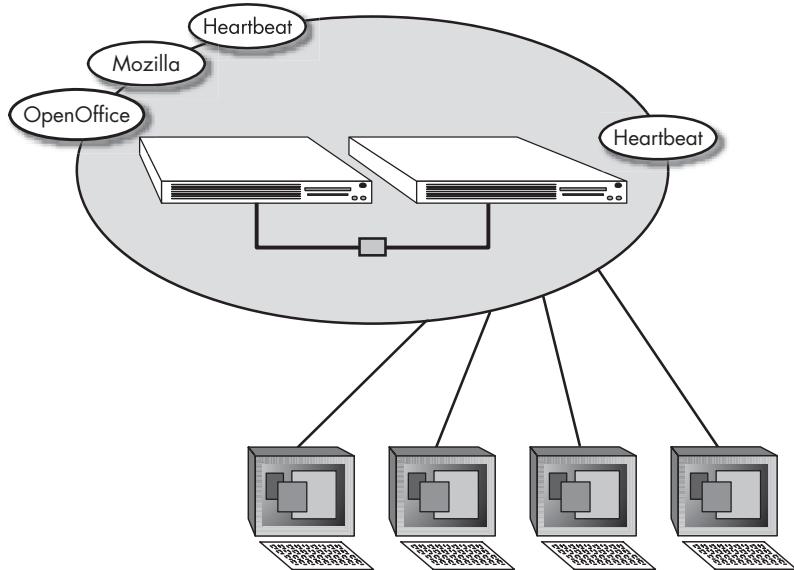


Figure 20-3: High-availability LTSP servers and Thin Clients

Notice in this configuration that all requests for cluster services originate from the LTSP servers, because the X applications will normally run on the CPU of one of the LTSP servers.<sup>2</sup> Thus, care must be taken to ensure that load balancing works properly when all of the client requests originate from a single IP address (as discussed in the section entitled “LVS Persistence” in Chapter 14).

**NOTE** *It may be possible to build an LVS cluster in which all real servers run the LTSP server software. Depending upon your situation, however, this may make it harder to maintain both the LTSP server software and your mission-critical applications, if they must run on the same machine (a cluster node). A cluster that runs presentation and business tier objects, in other words, may not be the ideal cluster.*

## The NAS Server

The NAS server (see Figure 20-4) stores all data used by the cluster legacy applications and the Thin Clients, and it can run a lock manager (discussed in Chapter 16) that can be said to reside at the data tier to arbitrate business tier object access to data. (If all of the shared data is stored in a database management system, the DBMS may act as the lock arbitrator at the data tier instead of the lock manager running on the NAS device.)

The NAS server will normally connect to the Ethernet backbone using one or more high-speed connections and may provide enough redundancy of critical components (power supply, fans, disks, and so on) in a single enclosure to eliminate worry about a second NAS device for high availability.

<sup>2</sup> Although LTSP clients can be configured to run applications on the local CPU of each Thin Client.

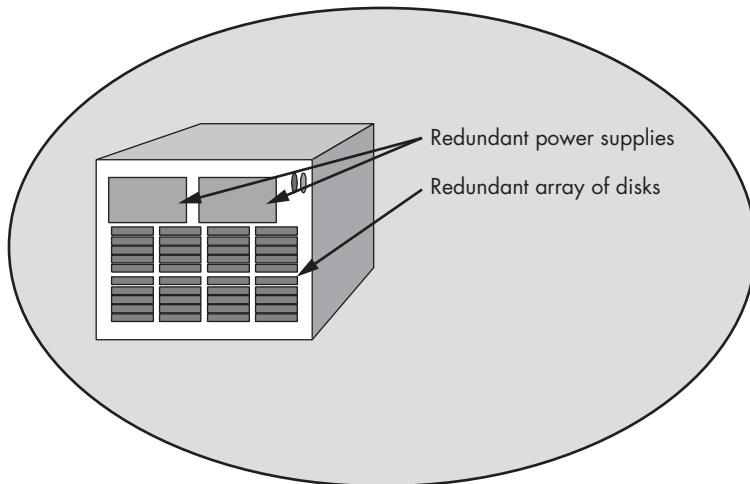


Figure 20-4: High-availability NAS server

### **High-Availability NAS Server**

As shown in Figure 20-4, your highly available NAS server will use RAID drives to eliminate the disk drives as a single point of failure. Not shown in Figure 20-4 is the network connection used by the NAS device. Normally, a mission-critical NAS configuration will also use two physical connections to the Ethernet backbone and thus eliminate the network connection as a single point of failure.

**NOTE** *A second NAS server is required at another location for disaster recovery.*

Storing your data centrally makes it easier to back it up, allows you to create online snapshots, gives you the ability to grow volume sets without reformatting disk drives as disk space requirements grow and lets you use a vendor-supplied data replication system for creating a disaster recovery site.<sup>3</sup>

### **Highly Available Serial Devices**

The ideal data center will support both past and future hardware devices, and one type of device you are likely to use in your data center from the past is a serial communication device.

Although the need for most serial devices has been eliminated by newer technology, most data centers retain some legacy hardware that still uses this simple method of transferring data. Fortunately, several vendors now build a device that converts serial communication into network packets. These serial-to-IP network devices make it possible to communicate with the serial device from any computer attached to the network.

---

<sup>3</sup> Many, if not all, of these functions can be built using open source software such as RAID, LVM, and the rsync program (rsync was discussed in Chapter 4). Describing how to build a NAS server and how to implement disaster recovery data replication is outside the scope of this book.

For example, if you need to send faxes from a business tier application running on the cluster (or anywhere else inside your network), you may want to run a fax server software package such as the open source Hylafax program. The Hylafax server can run on the cluster node manager and provide faxing capabilities to all of the cluster nodes. To prevent the fax modem(s) from becoming a single point of failure, you can purchase double the number of modems you require and connect half of them to the primary cluster node manager and half of them to the backup cluster node manager. Alternatively, you can connect these modems to a serial-to-IP communication device.

### **Serial-to-IP Communication Device**

A serial-to-IP communication device (see Figure 20-5) allows you to failover the application program (Hylafax, in this example) that talks to the serial device from a primary to a backup server. When you use this type of device, you load special driver software from the serial-to-IP device manufacturer that allows you to fool the application programs running on the cluster node manager into thinking they are communicating with a serial device directly connected to the system, when in fact the serial device is really only accessible over the network. Of course, you must purchase two of these devices (and at least two serial devices) to avoid a single point of failure.

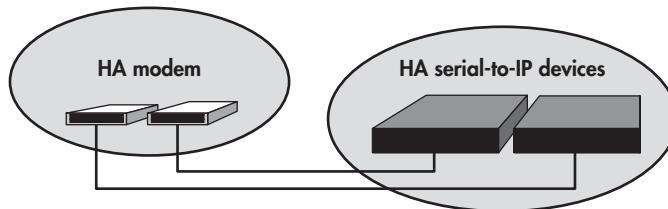


Figure 20-5: Serial-to-IP communication devices

### **High-Availability Modems**

The dark lines connecting the modems to the serial-to-IP devices in Figure 20-5 represent serial cables. The network cables connecting the serial-to-IP devices to the Ethernet backbone are not shown.

**NOTE** As noted in Chapter 8, Heartbeat's *cl\_respawn* program may be required to failover a service that needs to restart automatically when the service fails (typically the case for applications communicating with serial devices).

## **Highly Available Database Server**

If write access is required to data stored in an SQL database by the business tier objects running on the cluster nodes, an SQL server is placed outside the cluster to arbitrate access to this data (see Figure 20-6). The SQL client

program should allow you to specify the IP address of the SQL server when it first starts up. You should also be able to specify the host (along with port, user, password, and database) through any of the SQL APIs when connecting to a database. (You always connect to a database before doing anything else with the data.)

The SQL commands and data sent from the SQL client (on each cluster node) are then sent to the central SQL server where the SQL commands are executed.

The SQL server (Postgres, MySQL, Oracle, and so on) accesses the data from locally attached, high-speed RAID disk drives or (not shown) a SAN.

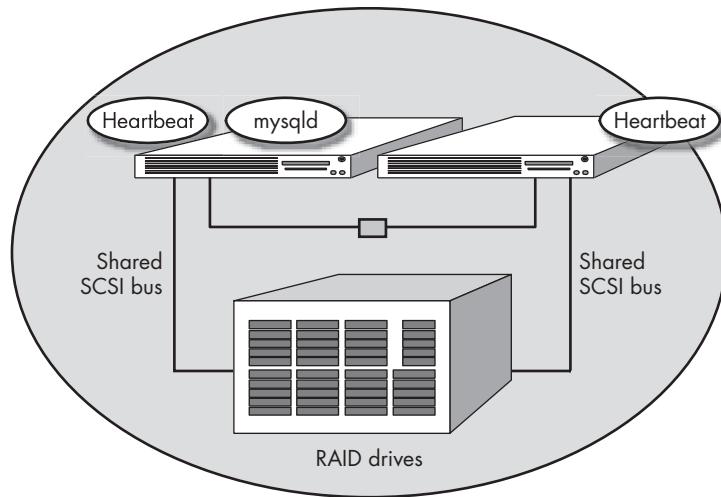


Figure 20-6: A highly available database server

### **High-Availability SQL Server**

As shown in Figure 20-6, a Stonith device is used to control the power to the primary SQL server. If it malfunctions, the Heartbeat program running on the backup SQL server resets or turns off the power to the primary server and takes ownership of the database resource. The database resource is the SQL server daemon(s), the filesystem where the SQL data resides, and the IP address where the cluster nodes send their SQL commands.

Another technique to avoid a single point of failure for SQL data is usually also provided through a replication mechanism built into the SQL server. (See the documentation for your SQL package for instructions on how to replicate your SQL data to a backup server.)

**NOTE** *SQL databases use special methods to protect the integrity of each transaction. These techniques are called ACID (Atomicity, Consistency, Isolation, Durability). See “SQL Databases for Linux” on freshmeat.net (at the following URL as of spring 2005: <http://freshmeat.net/articles/view/305>) for details.*

## CLUSTERING ZOPE

Another method of deploying a database in a cluster environment that is a relatively new technology is provided by the Zope Object Database (ZODB) and the Zope Enterprise Objects (ZEO) packages.

The Zope project (<http://www.zope.org>) uses Python to allow web developers to use an HTML-like language called TAL (or DTML) to write web pages that the Zope Python programs then interpret and execute to produce valid HTML code dynamically (when the web page is requested by a client computer). This dynamic HTML code (which is not the same thing as DHTML) at the business tier can access objects stored in the ZODB at the data tier, which is normally just a file on the local disk drive where the Zope server runs. With the addition of the ZEO program,<sup>1</sup> the dynamic HTML code interpreted by the Zope server can access ZODB objects at an IP address rather than a file. Each cluster node then becomes a ZEO client, and a highly available ZODB database server can be built outside the cluster as described in the previous SQL server example. The highly available ZODB server owns the IP address the ZEO clients use to access the ZODB objects and is therefore called a ZEO server.

Dynamic HTML pages written in TAL (or DTML) are called *products* by the Zope developers. The best known Zope product is perhaps the Plone project (<http://www.plone.org>). The Zope (business tier) products that access ZODB objects can be installed on all of the cluster nodes and share access to the same data stored on the ZEO server (or on your own server). The ZEO server arbitrates access to the data without using a locking mechanism by informing all of the ZEO clients when an object stored in the central ZODB has changed.<sup>2</sup> The ZEO client-server model is therefore designed for data objects that are read more than they are written or changed. (The <http://www.zope.org> website, by the way, runs on an LVS cluster.)<sup>3</sup>

Zope TAL (and DTML) programs can also connect to SQL databases using Zope database connection products. You can quickly develop applications to run on your cluster using Zope that access the highly available SQL or ZODB database server outside the cluster.

---

<sup>1</sup> ZEO ships as part of the ZODB package.

<sup>2</sup> See <http://www.zope.org/Wikis/ZODB/MultiVersionConcurrencyControl>.

<sup>3</sup> According to the documentation available on the [zope.org](http://zope.org) website at the time of this writing.

## Putting It All Together

The cluster node manager, the NAS device, the Thin Client server, the SQL server, and the high-availability serial solution we've built all sit outside the cluster, as shown in Figure 20-7.

### ***The Cluster Environment***

As you can see, both PC clients and LTSP Thin Clients can access the cluster services, using a web browser, for example. (For telnet access to Linux from a Windows PC, see the free telnet client called PuTTY.) Notice in this figure that all of the servers used to support the applications that run on the cluster are built with no single point of failure.

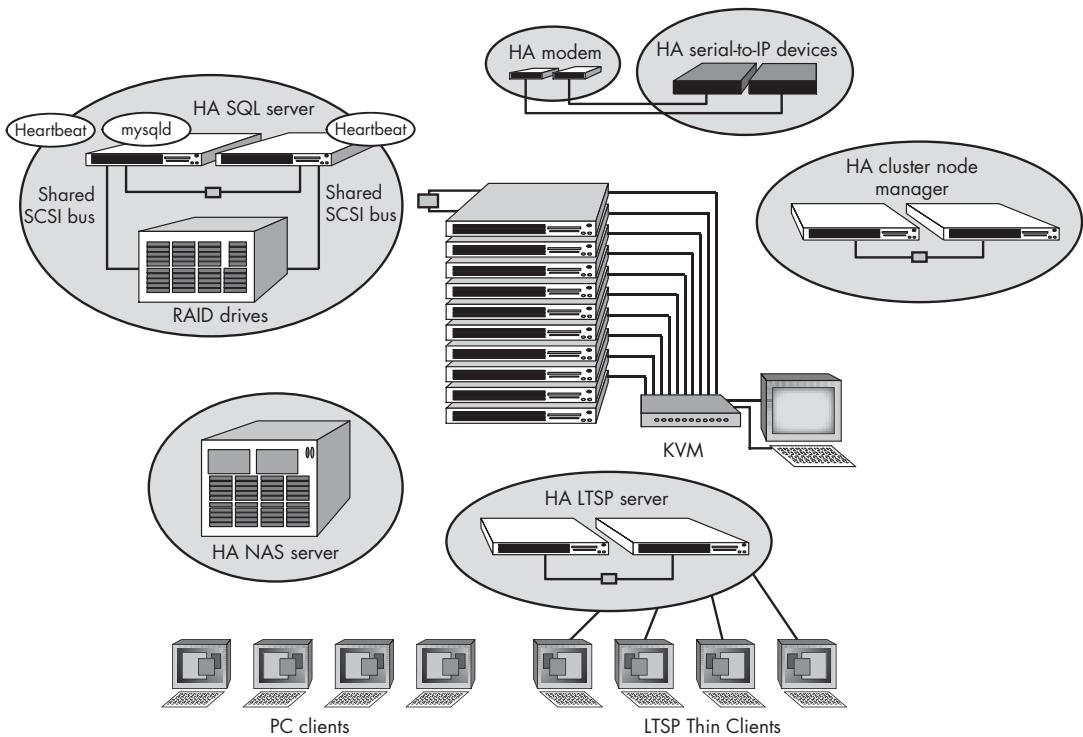


Figure 20-7: The Linux Enterprise Cluster

**NOTE** *Figure 20-7 does not show the underlying network infrastructure that the cluster is built on top of. If your budget permits, you should also build this network infrastructure with no single point of failure and split each server and half of the cluster nodes on to separate network hardware components (separate network switches). Also not shown in Figure 20-7 is the disaster recovery data center that contains a mirror copy of the hardware, software, and critical data.*

## In Conclusion

The cluster's ability to reliably offer business tier applications to the end-user rests upon servers outside the cluster nodes such as data tier servers and a cluster node manager. The data tier servers ensure data integrity by implementing a lock arbitration method that protects the data from changes made by the same business tier object running concurrently on multiple cluster nodes. The cluster node manager also helps you implement security policies and control access to external devices (such as printers and serial devices) that sit outside the cluster. Care must be taken when building data tier servers and the cluster node manager to ensure that they—like the cluster itself—have no single point of failure.

With open source software, you can build an enterprise-class cluster using inexpensive commodity hardware, and if you design the cluster environment properly, you'll eliminate downtime caused by single-component failures.

# A

## **DOWNLOADING SOFTWARE FROM THE INTERNET (FROM A TEXT TERMINAL)**



The File Transfer Protocol (FTP) is used, like the HyperText Transfer Protocol (HTTP), to move data over the Internet.

FTP, however, was designed specifically to transfer large files undamaged over a network.

You can use FTP from a text terminal with the ncftp utility. This utility makes downloading easier by doing the following:

- It automatically logs on to FTP servers anonymously.
- It automatically attempts to reconnect to FTP servers when they refuse your login request when too many users are already logged in to the server.
- It works through firewalls (it supports passive FTP).

If you purchased the Red Hat CD-ROM, you can install the ncftp program by inserting the CD-ROM and typing:

---

```
#mount /mnt/cdrom
#rpm -ivh /mnt/cdrom/RedHat/RPMS/ncftp*
```

---

Once the line of hash marks (#) goes by, you should be able to type:

---

```
#/usr/bin/ncftp
```

---

This will start the ncftp utility and provide you with the ncftp> prompt where you can type help for the list of commands ncftp understands.

If you don't have a CD-ROM with the ncftp utility, you can download it using the ftp utility.

Next, you need to find the FTP server with the files you need. When you are looking for Red Hat's RPM files, you should look at Red Hat's FTP site, one of the Red Hat mirror sites, or <http://www.rpmfind.net>. (Also see Appendix D for a description of the Yum package.) For the latest version of a package like Apache, however, you may want to go directly to their FTP server. Here are a few examples:

**Red Hat's FTP site**      [ftp.redhat.com](ftp://ftp.redhat.com)

**Apache's FTP site**      [ftp.apache.org](ftp://ftp.apache.org)

You can open a connection to the FTP server running at one of these addresses and log in using the anonymous account (this is what your web browser does for you when you use `ftp://` instead of `http://` in the Address field of the browser).

At the ncftp prompt, type:

---

```
ncftp>open ftp.redhat.com
```

---

Or at the Linux shell prompt, you could have typed:

---

```
#ncftp ftp.apache.org
```

---

You should see a banner message indicating a successful connection to an FTP server and eventually the `Logged in to ftp.apache.org` message.

From here, you will normally use the simple `cd` (change directory) and `get` commands to locate and download a file. To first position your local `ftp` client program in the proper directory where you would like the files downloaded, you can use the `lcd` (which means local change directory) command:

---

```
ncftp> lcd /usr/local/src
```

---

(This tells the `ftp` program that you would like files retrieved with the `get` command to be loaded locally into the `/usr/local/src` directory on your computer.)

To check which directory you are in, type:

---

```
ncftp> lcd .
```

---

**NOTE** *There is a space between the lcd and the period.*

Now you need to look on the remote FTP server to locate the file you are interested in downloading. This is usually somewhere underneath a directory called /pub (for public), so if you are not sure, you can start by changing to this directory and listing files:

---

```
ncftp>cd /pub
```

---

**NOTE** *This changes to the /pub directory on the FTP server.*

---

```
ncftp>ls
```

---

Once you find the file you want to download, you should tell the ncftp program if it contains binary data or ASCII (plaintext) before issuing the get command:

---

```
ncftp>bin
```

---

(If you download a file with a .zip, .gzip, .tar, or .Z extension, you should use binary.)

Then tell the ftp program to begin downloading the file to your local hard drive with the command:

---

```
ncftp>get apache.x.x.tar.Z
```

---

You should now see a display indicating that the file is downloading to your local hard drive.

Quit out of the ncftp program by typing exit.

---

```
ncftp>exit
```

---

Skip now to the discussion later in this appendix, “What to Do with the File You Downloaded.”

## Using Lynx

Some sites (like <http://www.linuxvirtualserver.org>) only allow you to download files using the HTTP protocol. For these cases, you can use the text-based web browser called Lynx or the non-interactive tool wget (discussed in the next section). The commands to start a download with lynx might look like this:

---

```
#cd /usr/local/src
#lynx www.linuxvirtualserver.org
```

---

Then press the down arrow key until the link you want is highlighted. When you press ENTER on a link that points to a file for downloading, the system will inform you of the file type and then prompt you: Download or Cancel. After you press D and the download completes, the system will display a screen with “Standard download options: Save to Disk” or “Local additions: View with less” (less is a program that will attempt to display the contents of the file on the screen one screen, or page, at a time).

Highlight “Save to Disk,” and press ENTER. Lynx will prompt you for a file name with the downloaded file name already set as the default. You can just press ENTER on this screen, and the file you just downloaded will be saved to whatever directory you were in when you started lynx (/usr/local/src, in this case).

After the downloaded copy of the file is saved to disk, quit out of lynx by pressing the Q key twice.

## Using Wget

The wget utility is similar to Lynx in that it uses the HTTP protocol, but wget is a non-interactive HTTP download utility, so you’ll need to know the exact URL of the file you want to download. To download the kernel source code tar ball, for example, with wget the command might look like this:

---

```
#wget http://www.kernel.org/pub/linux/kernel/v2.4/linux-x.x.x.tar.gz
```

---

Where x.x.x is the kernel version number you want to download. wget will display a percentage complete counter using text on your terminal screen as it downloads the file using the HTTP protocol.

### ***What to Do with the File You Downloaded***

The command you need to enter next depends on the name of the file you downloaded. To be more specific, it depends upon the extension of the file name you downloaded. You probably are downloading files that look like this: *filename.tar.gz* or *filename.tar.bz* or *filename.rpm.gz*, and so forth. For the moment, we are only concerned with the final portion of the file name extension that is indicating the type of compression used by the file’s creator to make the file smaller for transmission over slow networks. This extension and the command you need to use to uncompress the file will probably be one of the following:

<b>.bz</b> or <b>.tbz</b>	<code>#bunzip2 filename.tar.bz</code>	<i>or</i>	<code>filename.rpm.bz</code>
<b>.bz2</b> or <b>.tbz2</b>	<code>#bunzip2 filename.tar.bz2</code>	<i>or</i>	<code>filename.rpm.bz2</code>
<b>.Z</b>	<code>#uncompress filename.tar.Z</code>		
<b>.gz</b>	<code>#gunzip filename.tar.gz</code>	<i>or</i>	<code>filename.rpm.gz</code>
<b>.bz2.tar</b>	<code>#bzip2 -d filename.tar   tar xv</code>	<i>or</i>	<code>filename.rpm.gz</code>

After issuing one of these commands on the file you downloaded, you should have a file with either a .tar or .rpm extension (in the same directory as the original file). We'll discuss how to handle these two types of files next.

## ***tar Files***

The .tar portion of the file name indicates this is a tape archive file containing several files rolled into one. To untar the files in the tape archive, enter the command:

---

```
#tar xvf <filename>.tar
```

---

This normally creates a directory in the current directory (where the command is executed and where the file you downloaded resides) containing the uncompressed files (and subdirectories).

**NOTE** *You can uncompress a tar.gz file with a single tar command:*

---

```
#tar zxvf filename.tar.gz
```

---

## ***rpm Files***

To (optionally) view the list of files contained within this Red Hat Package Manager (RPM) file without installing them on your system, type the command:

---

```
#rpm -qlp filename.rpm
```

---

You can then install the package with the following command:

---

```
#rpm -ivh filename.rpm
```

---

If you receive an error that a previous version of this package is already installed, you can upgrade to this newer version with the command:

---

```
#rpm -Uvh filename.rpm
```

---

**NOTE** *To view the packages already installed on your system, type:*

---

```
#rpm -qa
```

---

*Combine this command with the grep command to search for a specific package:*

---

```
#rpm -qa | grep package-name
```

---

See Appendix D for an in-depth discussion of how to use the rpm program to install packages.

## Creating Your Own Tape Archive (tar) File

How do developers create those compressed tape archive files? In this section, we'll provide brief instructions using your kernel source files as an example.

First, place all of the files into the tar file with the command:

---

```
#tar cvf /tmp/kernellibs.2.6.x.tar /lib/modules/2.6.x
```

---

This command takes all of the files and all subdirectories of files and places them into the file `/tmp/kernellibs.2.6.x.tar` (leaving the original files untouched).

To reassure yourself that this tape archive file really contains all of your files and directories, you can list the contents of the file (without untarring it) with the command:

---

```
#tar tvf /tmp/kernellibs.2.6.x.tar
```

---

You can then compress this file with one of the following compression methods:

---

```
#compress /tmp/kernellibs.2.6.x.tar
#gzip /tmp/kernellibs.2.6.x.tar
#bzip2 /tmp/kernellibs.2.6.x.tar
```

---

This will change the name of the file and add the proper extension to indicate the type of compression you selected.

You can then copy these files over to the destination system and uncompress and untar them using the commands in the previous section.<sup>1</sup>

---

<sup>1</sup> Note that an easier method to accomplish the same thing when you are the administrator of both hosts is to use the `scp` or secure copy command with the `-r` option. See the `scp` man page for more information.

# B

## **TROUBLESHOOTING WITH THE TCPDUMP UTILITY**



The `tcpdump` utility can be used for low-level diagnostics of packet communications. Normally, you run `tcpdump` on the system that is either receiving or sending the packets and examine the output as it scrolls by.

The `tcpdump` utility supports command-line arguments that allow you to control which packets are displayed. `tcpdump` works outside of the `ipchains/iptables` and routing rules you have created. This makes it very useful for examining packets to see what is really coming into the system.

If, for example, you sit at host `realserver1` and have the following Netfilter rule in place:

---

```
#iptables -A INPUT -s client10 -j DROP
```

---

and then you run `tcpdump` with the argument to display only packets from `client10` with the following command:

---

```
#tcpdump host client10
```

---

you will still see the packets dump onto the screen even though `client10`'s packets never make it up to the listening daemons on the system.

You can also control which interface `tcpdump` listens on by adding a `-i <interfacename>` argument. For example, to only see the packets hitting the `eth1` interface, you would use:

---

```
#tcpdump -i eth1 host client10
```

---

**NOTE** *To view IP addresses instead of host names, use the `-n` argument to `tcpdump`.*

(If you leave off the `-i` argument, `tcpdump` will listen on all of its devices and add the device name to each line it prints out.)

Here is what one packet from `tcpdump` looks like:

---

```
16:36:25.667437 eth0 < 10-1-1-1.my.test.domain.1391 > 10-1-1-  
2.my.test.domain.telnet: S 2644956594:2644956594(0) win 16384 <mss
```

---

Let's examine this line in more detail.

After the timestamp, the `eth0` is indicating that this packet was seen on the `eth0` device. The next character is either a `<` to indicate an inbound packet or `>` to indicate an outbound packet. The next two parameters are the source and destination hosts followed by the port number or port name (or to be more accurate, the service name listed in `/etc/services`). This packet was sent to the `telnet` port on `eth0` from the `10.1.1.1` host. The `10.1.1.1` host is expecting a response back on port `1391` (an unprivileged port). Next, the letter `S` indicates the `SYN` flag was set in the packet. The complete list of IPv4 flags is: `S` (`SYN`), `F` (`FIN`), `P` (`PUSH`), or `R` (`RST`) or a single `.` to indicate no IPv4 flag is set.

You have to know more about how network communication protocols are supposed to work to troubleshoot much further (see the `tcpdump` man page for some examples near the end that might help). But for most of the work you do to set up cluster, simply knowing that a packet was received on the correct port is often all the information you need. You test for packets received (or sent) on a particular port with the command:

---

```
#tcpdump -n port 80
```

---

This command listens to all HTTP packets on any Network Interface Card connected to the system.

You can combine tcpdump packet criteria (with `and`) to see if a specific client's HTTP request is received:

---

```
#tcpdump -i eth1 host client10 and port 80
```

---

This command says to listen on interface `eth1` for packets sent to or from `client10` on port 80.

One of the most useful things tcpdump will tell you is that the network communication between the client and the host is being reset. Look for packet reports from tcpdump that contain an `R` (instead of an `S` or no flag, which is usually the case). Using tcpdump, you can determine which host (the client or the real server or the Director) is attempting to reset the connection. This usually means it “sees” a packet coming in on an established connection but the packet does not contain the sequencing or acknowledgement number expected. If you see this happening, check to make sure there is a service or daemon running on the server that is sending the reset. Usually, the reset occurs when this server is not running the daemon the client computer is trying to talk to. You can check to see which ports have daemons listening by running the following command on the server:

---

```
#netstat -apn
```

---

Also look for tcpdump reports that indicate packets are only going in one direction. In other words, all of the packets you see between your web server and the host connecting to it contain only a `<` or they all contain only a `>`. This probably means you have created Netfilter rules that `DROP` (iptables) or `DENY` (ipchains) the packets you are interested in.

tcpdump might even show you an ICMP message you get back somewhere on the communication path between the web server and the client. If you see the message `Admin Prohibited Filter`, this indicates a firewall between you and the client is blocking the packets. (You'll know which one, because the ICMP source address—the address on the left—is the firewall that is blocking you.)

If you see a `port unreachable` message such as the following:

---

```
16:41:19.273329 eth0 > host1 > host2: icmp: host1 udp port tftp
unreachable [tos 0xc0]
```

---

this probably indicates that the specified port (tftp, in this case) is either blocked by ipchains/iptables rules or that there is no daemon running (`tftpd`, in this case) to answer requests on this port.

Another useful diagnostic tool is the free Ethereal tool available at <http://www.ethereal.com>.



# C

## **ADDING NETWORK INTERFACE CARDS TO YOUR SYSTEM**



The easiest way to configure network interface cards (NICs) on your system is to set them up during the installation of your Linux distribution.

The next easiest method is to use either the netconfig or the linuxconf utilities. In this appendix, we'll focus on the linuxconf utility. If your distribution does not include the linuxconf utility, you can download it from <http://www.solucorp.qc.ca/linuxconf/download.hc>.

(If you prefer not to run utilities to modify your network configuration and would rather make the changes "manually," you can simply edit the files in the `/etc/sysconfig/network-scripts` and the `/etc/sysconfig` directories.)

Before you can use either of these methods, you need to determine what type of NIC you are adding to your system and which kernel module (or driver) it uses.

The kernel module, or network driver software, allows the Linux kernel to work properly with your particular NIC and is normally supplied by the company that built the NIC. You will need to consult their website or documentation to find out the name of their Linux driver (they may even

have instructions for compiling and installing a new version of their driver software, but normally you will not need to do this because the kernel source usually contains the driver software you need).

### ***Monolithic Versus Modular Kernels***

If you are using a monolithic kernel and need to add a new NIC driver to your system, you must rebuild your kernel (described in Chapter 3) with the new driver compiled into the kernel binary. However, if you are using a modular kernel,<sup>1</sup> you probably already have a compiled network driver module that will support your NIC, and you simply need to insert this module into the kernel.

You can insert modules into the kernel using the `insmod` command. However, the system will forget about your `insmod` change to the running kernel the next time you reboot, so you need to use the `linuxconf` utility to make the change permanent—the `linuxconf` utility will also make changes to the proper files in the `/etc/sysconfig` directory and to the `/etc/modules.conf` and the `/etc/sysconfig` files to make your IP address and netmask settings permanent.

### ***View Your Existing Configuration***

Before installing the new NIC into your system, enter the following commands:

---

```
#lspci -v
```

---

and

---

```
#ifconfig -a | more
```

---

Note from the output of these commands which NIC's base addresses are already in use on your system and write them down (assuming it is a PCI NIC). You can also enter the command:

---

```
#lsmod
```

---

This will display the currently loaded modules your kernel is using (if you are using a modular kernel). On a standard Red Hat installation (which uses a modular kernel), this command will show you the network driver module the system has loaded to support your existing NIC(s).

---

<sup>1</sup> And you are using the `.config` file as supplied by Red Hat or one like it that builds the NIC drivers as modules in the `/lib/modules` directory.

## **Install the Card and Reboot**

Now power down the system and install the new NIC, reboot, and log back in as the `root` user and enter:

---

```
#lspci -v
```

---

You should be able to determine from this command what type of NIC the system thinks is connected to the PCI bus, and if you know the proper name of the Linux driver (based on the documentation from your NIC vendor), you are ready to run `linuxconf`.

### **Getting Ready to Run `linuxconf`**

Enter the following command:

---

```
#rpm -qa | grep linuxconf
```

---

If this command does not return the version number of the currently installed `linuxconf` RPM package, you need to load it by following this next set of instructions. If you already have `linuxconf` loaded on your system, go to the next section.

Insert the proper CD-ROM (probably the first or second CD-ROM) included with your Red Hat distribution, or the first CD in prior releases, into the CD-ROM drive, and then enter:

---

```
#mount /mnt/cdrom
#cd /mnt/cdrom/RedHat/RPMS
#ls linuxconf*
```

---

You should now see a list of `linuxconf` RPM packages on your CD. If this CD does not have `linuxconf` installed on it, you will need to try another CD (enter the command `cd / ; eject` and start over.)

Note that you can also download `linuxconf` from the [ftp.redhat.com](http://ftp.redhat.com) FTP site or one of its mirrors.

If you see a listing that contains `linuxconf-<version>` and `linuxconf-devel-<version>`, you can proceed with the following command to load it onto your system:

---

```
#rpm -ivh linuxconf-<version>
```

---

(You do not need the `linuxconf-devel-<version>` RPM package.)

Again, remember that you can use the TAB key to complete this command once you have entered enough characters to make the file name unique.

Eject the CD out of your system with the commands:

---

```
#cd /
#eject
```

---

Once the package is installed, enter the following command to tell your shell to review files in its current PATHs:

---

```
#rehash
```

---

(If you don't enter this command, you will get `command not found` when you type `linuxconf` until you start a new shell.)

## ***Run `linuxconf`***

You are now ready to start the `linuxconf` utility with the command:

---

```
#linuxconf
```

---

A welcome screen appears, and you need to click **QUIT** (or press the TAB key to highlight **QUIT** and press ENTER) to start using `linuxconf`.

Once you are past this screen, you can now use the arrow keys on your keyboard and press ENTER until you are at the following submenu:

Config

Networking

Client tasks

Host name and IP network devices

Pressing ENTER on "Host name and IP network devices" will bring up the screen that allows you to enter the following information to enable your NIC.

(First press the SPACE bar while on the "[ ] Enabled" line of the proper "Adaptor" section—Adaptor numbers should be the same as the eth device numbers to make things clear.)

### **Primary name + domain**

Enter a registered DNS host name if you have one for your system. This field is not required. (Most hosts within the Linux cluster will not have, or need, DNS registered host names.) However, if you are running `sendmail` or `lpd` on your cluster nodes, your host should have a name, and its IP address should be in the local `/etc/hosts` file on each cluster node.

### **Aliases (opt)**

Optional host name aliases for this machine.

### IP address

The IP address you want to use for the NIC.

### Netmasks (opt)

All of the hosts in your network should use the same netmask. The examples in this book (for the Linux clustered computers) are using 255.255.255.0 (meaning only the last byte is used for the host IP address—the first three bytes contain the network portion of the address).

### Net device

Enter the eth device number you determined above into this field.

For example:

---

eth1

---

### Kernel module

Enter the name of the kernel module. Here are a few examples:

For the Intel Ethernet Pro Card, you might use:

---

eepro100

---

For the old 3COM 3C509 card, you can use:

---

3c509

---

Another 3COM driver for 3c59 series network cards is:

---

3c59x

---

### I/P port (opt)

You can take the guesswork out of the eth device name mappings by entering the base address from the `lspci -v` command in this field.

As you quit out of `linuxconf`, tell the utility to activate your changes.

## ***Testing the New NIC***

Check to make sure the system recognizes your new NIC with the command:

---

`#ifconfig -a | more`

---

You should now see your new NIC listed along with its associated IP address. If not, try entering the command to manually bring up the NIC you just defined (for example):

---

`#ifup eth1`

---

## ***Changes Made by linuxconf***

linuxconf should have made changes to add the definition for your new NIC to the `/etc/sysconfig/network-scripts` directory, and created (or modified) a file in this directory named after the eth device number you have added (for example, `ifcfg-eth1`).

**NOTE** *The `ifcfg-1o` file in this directory is for the loopback device on your computer. Do not remove it.*

linuxconf may have also made changes to the `/etc/sysconfig/network` file. It should have also added an “alias” to tell the Linux kernel which network driver module to associate with this eth device (if you are using a modular kernel) in the `/etc/modules.conf` file.

These changes are permanent, because the boot script `/etc/rc.d/init.d/network` uses these files (on a Red Hat system) to know how to configure the NICs on your system.

## ***Using the New NIC***

Once you have added these NICs to these scripts, you can also use the following shortcut to bring the interfaces up or down (instead of entering the full `ifconfig` command):

---

```
#ifup eth1  
#ifdown eth1
```

---

## ***Testing and Troubleshooting***

You can test to see whether the new NIC is operational (at least the kernel module) by pinging its IP address (for example):

---

```
#ping 10.1.1.1
```

---

(Press CTRL-C to break out of this listing and see a summary report of your ping statistics. It should indicate “0% packet loss,” if everything is working properly.)

If it does not work, you will see:

---

```
10.1.1.1: Destination Host Unreachable
```

---

(Press CTRL-C to break out of this listing as well.)

The first step in troubleshooting this new NIC (after you have made sure the network cable is connected) is to enter the command:

---

```
#ifconfig -a | more
```

---

Make sure your new eth device has been added to this listing and contains the following line:

---

```
UP BROADCAST RUNNING MULTICAST MTU:nnnn Metric:n
```

---

(Where *n* indicates a number.)

If you do not see the word UP at the beginning of this line, it means your ifup command did not work properly. You will need to go back and make sure you have the correct eth device and network driver (either compiled into the kernel or compiled as a module in the /lib/modules directory).

If you do see the word UP, try pinging the IP address of this eth device (the local IP address on your machine) which in this example would be the following command:

---

```
#ping 10.1.1.2
```

---

If this does not report 0% packet loss when you press CTRL-C, it probably means you have made a mistake in the above steps and need to go back and check everything you have entered again.

If this works, however, it means there is still something blocking your packets from getting to the destination host and back (Internet Control Message Protocol or ICMP packets, to be precise).

Make sure you do not have any ipchains or iptables rules that would block communication between your system and another host. You can remove all ipchains or iptables rules with the -F (flush) option:

---

```
#ipchains -F      or      #iptables -F
```

---

List the rules with the -L option:

---

```
#ipchains -L      or      #iptables -L
```

---

Your input, forward, and output default policy should now be ACCEPT. (Of course, make sure the host you are trying to ping is not blocked by an intervening firewall and that it is also not using iptables/ipchains rules to block your ICMP ping packets.) See Chapter 2 for more information on how to configure the ipchains and iptables rules properly.

**NOTE** *If the ping command now works, you may need to modify your ipchains rules in the file /etc/sysconfig/ipchains or /etc/sysconfig/iptables.*

If the ping command still does not work, you might try using another method of communicating with the remote host (try to telnet to it). If this works, it probably means a firewall between your systems is blocking ICMP traffic.

See the online manual, “Linux Network Administrators Guide,” for more information about configuring and enabling NICs on your Linux system.



# D

## STRATEGIES FOR DEPENDENCY FAILURES



The strategies for dealing with rpm dependency failures can be broken down into two basic categories: automated methods of resolving dependency failures and manual methods. Automated methods are not available, however, when you install packages that are not a part of your distribution. In this appendix, I'll briefly describe how to use one of the automated methods (Yum) after I describe how to resolve dependency failures manually, but first you need to understand what they are and how rpm enforces them.

### What Are Dependencies?

Programs depend upon shared libraries of program code so that they can make system calls to send output to a device, open files, and so forth.<sup>1</sup> Without shared libraries, every programmer would have to rewrite these basic system operations from scratch every time they developed a new

---

<sup>1</sup> Shared libraries exist for a wide variety of things not limited to system calls.

program. When a program is compiled, the programmer *links* his or her code to these libraries. If the link is *static*, the compiled shared library object code is added to the program executable; if the link is *dynamic*, the compiled shared library object code is only loaded by the program when it is needed at run time. *Dynamic executables*, as they are called, depend upon the correct shared libraries or *shared objects* for normal operation. rpm dependencies try to enforce the shared object requirements of dynamic executables at installation time so that later—when the program runs—there won’t be any problems with the dynamic linking process.

**NOTE** *There is a second type of dependency that rpm enforces based on an explicit entry into the rpm configuration files by the programmer, but for the moment we’re not concerned with this type of dependency, which is easy to understand and resolve. We want to focus on the more complex shared object dependencies that rpm enforces.*

## ***Dynamic Executables and Shared Objects***

Dynamic executables look for shared objects using the shared object name or *soname* of the library file that was used at the time the program was originally compiled and linked. They look in a few standard places, such as in the `/lib` and `/usr/lib` directories and in whatever directories are specified by the `LD_LIBRARY_PATH` environment variable (see the man page for `ld.so` for the complete list of possible directories). The shared objects found in these library directories may not be real files, by the way; they might be symbolic links to the real library file located somewhere else (but normally, still in one of the standard library directories).

But the confusing thing—at least from the system administrator’s point of view—is that normally there is little relationship between the name of the shared library package that was used to create the shared library file and the name of the shared library file. For example, the GLIBC 2.3 package is used to create the `libc.so.6` shared library file. Notice also from this example that the version number tacked on to the end of the shared library file name (.6) is not related to the version number of the package used to create it (2.3). This is intentionally done by the shared library package developers so that a new release of GLIBC, for example, can reuse the same shared library file name, or soname `libc.so.6`. This allows you to load a new version of GLIBC on your system without breaking all of your programs that are dynamically linked to the `libc.so.6` shared library file—assuming, of course, that the new version of GLIBC is backward compatible with the older version of GLIBC that your dynamic executables were originally linked against. So even though the library files, or shared object files, have version numbers associated with them, these version numbers do not help you to determine which version of the shared library package they came from.

**NOTE** *Information about the existing shared objects that were loaded on the system using rpm packages is available when you use the `-whatprovides` option to the rpm query command.*

This confusion is compounded by the fact that a single shared library file may support a *range* of shared library package versions (for backward compatibility). To examine the GLIBC shared library package version numbers supported by the soname library file `/lib/libc.so.6`, for example, run the command:

---

```
#objdump --all-headers /lib/libc.so.6 | less
```

---

Scroll down through this report until you reach the `Version definitions:` section to see which GLIBC versions are supported by your `libc.so.6` shared library file:

---

```
Version definitions:
1 0x01 0x0865f4e6 libc.so.6
2 0x00 0x0d696910 GLIBC_2.0
3 0x00 0x0d696911 GLIBC_2.1
    GLIBC_2.0
4 0x00 0x09691f71 GLIBC_2.1.1
    GLIBC_2.1
5 0x00 0x09691f72 GLIBC_2.1.2
    GLIBC_2.1.1
6 0x00 0x09691f73 GLIBC_2.1.3
    GLIBC_2.1.2
7 0x00 0x0d696912 GLIBC_2.2
    GLIBC_2.1.3
8 0x00 0x09691a71 GLIBC_2.2.1
    GLIBC_2.2
9 0x00 0x09691a72 GLIBC_2.2.2
    GLIBC_2.2.1
10 0x00 0x09691a73 GLIBC_2.2.3
    GLIBC_2.2.2
11 0x00 0x09691a74 GLIBC_2.2.4
    GLIBC_2.2.3
12 0x00 0x0963cf85 GLIBC_PRIVATE
:
    GLIBC_2.2.4
13 0x00 0x0b792650 GCC_3.0
```

---

In this example, the `libc.so.6` shared library file supports all the dynamic executables that were originally developed for GLIBC versions 2.0 through 2.2.4.

**NOTE** You can also use the `objdump` command to extract the soname from a shared library file with a command like this:

---

```
#objdump --all-headers /lib/libcrypto.so.0.9.6b | grep SONAME
```

---

Next, I'll discuss how the `rpm` package is built so that these shared library dependencies are known when the `rpm` package is installed on a new system.

## **rpm Packages and Shared Library Dependencies**

When the programmer builds an rpm package, the `ldd` command is used to report all of the shared libraries used by all of the dynamic executables in their package. Another confusion is added by the fact that different dynamic executables *within the same software package* may have been linked with different versions of the same shared library package. For example, different programs within the Heartbeat package may have been developed and dynamically linked to different GLIBC versions of the `libc.so.6` soname shared library file.

You can see the full list of the shared libraries required by an rpm package with the `-q` and `--requires` arguments to the `rpm` command. For example, to see all of the required dependencies for the Heartbeat rpm package, use the command:

---

```
#rpm -q --requires -p heartbeat-1.x.x.i386.rpm
```

---

This produces the following report:

---

```
sysklogd
/bin/sh
/bin/sh
/usr/bin/python
ld-linux.so.2
libapphb.so.0
libc.so.6
libc.so.6(GLIBC_2.0)
libc.so.6(GLIBC_2.1)
libc.so.6(GLIBC_2.1.3)
libc.so.6(GLIBC_2.2)
libc.so.6(GLIBC_2.3)
libccmclient.so.0
libdl.so.2
libglib-1.2.so.0
libhbclient.so.0
libpils.so.0
libplumb.so.0
libpthread.so.0
librt.so.1
libstonith.so.0
```

---

Notice from this report that the `libc.so.6` soname is required and that this shared library must support dynamic executables that were linked using the GLIBC shared package version numbers 2.0, 2.1, 2.1.3, 2.2, and 2.3. This is due to the fact that different dynamic executables within the Heartbeat package were linked against each of these different versions of the `libc.so.6` library.

With this background knowledge of how dynamic executables, shared objects, sonames, and shared library packages relate to each other, we are ready to look at an example of what happens when you try to install an rpm package and it fails due to a dependency error.

## Fixing a Dependency Failure Manually

Usually, you will encounter rpm dependency errors when you attempt to install a package that was not included with your distribution (and therefore one that cannot have its dependencies automatically resolved by an update tool such as up2date, apt-get, or Yum).

For example, if you attempt to install all three Heartbeat rpm packages in the chapter7 subdirectory on the CD-ROM using the `rpm -ivh *rpm` command on an older Linux distribution, you may encounter the following error:

---

```
error: failed dependencies:
  libc.so.6(GLIBC_2.3) is needed by heartbeat-1.x.x
  libc.so.6(GLIBC_2.3) is needed by heartbeat-pils-1.x.x
  libcrypto.so.0.9.6 is needed by heartbeat-stonith-1.x.x
  libsnmp-0.4.2.6.so is needed by heartbeat-stonith-1.x.x
```

---

Notice that the `rpm` command did not bother to report each of the GLIBC shared library package version numbers that are required—it only reported the highest numbered one that was required (GLIBC\_2.3).<sup>2</sup> All of these failures are reporting the shared library names or sonames (not file names) that are required (sonames always start with “lib”). However, you can remove the version numbers tacked on to the end of the sonames reported by rpm and quickly check to see if you have these shared libraries installed on your system using the `locate` command.<sup>3</sup> For example, to look for the `libcrypto` shared library file you would enter:

---

```
#locate libcrypto
```

---

If this command does not find a `libcrypto` shared library file on your system, you’ll have to go on the Internet and find out which shared library package contains this shared library file. One quick and easy way to do this is to simply enter the name of the shared library into the Google Linux search at <http://www.google.com/linux>. If you enter the text `libcrypto.so` into this search page, you’ll quickly learn that this shared library is provided by the `openssl` package.

---

<sup>2</sup> The assumption is that the original package developer would not dynamically link executables within the same package to incompatible versions of a shared library package.

<sup>3</sup> Assuming your `locate` database is up to date. See the man page for `locate` or `slocate` for more information.

If an older version of the shared library package is already installed on your system, you can confirm that this package contains the shared library file you need with a command like this:

---

```
#rpm -q --provides openssl
```

---

This command reports everything this rpm package provides (which includes the sonames of the shared library files provided by the package).

**NOTE** *As stated previously, the shared library package version numbers do not, and should not, have any correspondence to the shared library file (soname) version number. I've left out of this discussion the fact that a soname symbolic link may point to different versions of a shared library file. This is also done in an effort to avoid breaking existing dynamic executables when a new version of a shared library package is installed.*

### **Fixing a Dependency Failure Automatically**

Dependency failures can quickly turn into a nightmare when you work with rpm packages to build, upgrade, or add new features to your system. You can avoid this nightmare by simply using your distribution vendor's update service or tool. For example, the Red Hat utility up2date automatically downloads and installs any rpm dependencies directly from Red Hat when you select an rpm package to install. Several community-supported, free methods to accomplish the same thing are listed on the website <http://www.rpm.org/software/updaters>. In this appendix, we'll take a closer look at just one of these automatic update tools: Yum.

## **Using Yum to Install rpm Packages**

The Yellow dog Updater, Modified, or Yum program can be downloaded (as an rpm package) from <http://linux.duke.edu/projects/yum/download.ptml>.

After downloading this package, you can install it like any other rpm package with the command:

---

```
#rpm -ivh yum*
```

---

You may need to update the repository you would like to use to download your rpm packages from. See the list of available Yum repositories for Fedora at <http://www.fedoratracker.org>. To switch to a different repository, download one of these files and install it as your `/etc/yum.conf` file.

Now we can tell Yum to report all packages stored in the Yum rpm Internet repositories that are available for install with the command:

---

```
#yum list
```

---

## Installing a New rpm Package with Yum

In this example, we'll install a new GLIBC package. Install the latest GLIBC and all of its dependencies with the simple command:

---

```
#yum update glibc
```

---

If all goes well, the Yum program will automatically detect, download, and install any rpm packages that are required for the latest GLIBC package that was built for your distribution (not necessarily the latest version of the GLIBC package that is available<sup>4</sup>).

You can also look for a package to install using the `list` argument to Yum and the `grep` command. For example, to look for a package with SNMP in the name, you would enter:

---

```
#yum list | grep snmp
```

---

This returns a report that looks like this:

---

php-snmp	i386	<version>
ucd-snmp	i386	<version>
ucd-snmp-devel	i386	<version>
ucd-snmp-utils	i386	<version>

---

You can now easily download and install all of these rpm packages using Yum.

## In Conclusion

You'll have to manually resolve rpm dependency failures when you install packages that are not a part of your distribution. However, once you know the name of the shared library package that was used to create the shared object, or soname, reported in the rpm dependency failure error message, you may be able to use an automated tool to download and install it.

The trick is that the soname reported in the dependency failure error message by rpm is normally not related to the shared library package name (and thus the shared library rpm package name) that was used to create it.

---

<sup>4</sup> Use the GLIBC shared library package version number that is approved for your distribution or you risk installing a version of the GLIBC package that does not work with the dynamic executables required for normal system operation.



# E

## OTHER POTENTIAL CLUSTER FILESYSTEMS AND LOCK ARBITRATION METHODS



Additional open source, or free, filesystem technologies that may be considered when building a shared data solution in a Linux enterprise environment include the following:

### CODA

Allows multiple file servers to store the same data. For the latest information, see <http://www.coda.cs.cmu.edu> (see the CODA FAQ “general” question, “How stable and usable is CODA?”).

### DRBD

Mirrors a storage device at the block level over the network. A primary node has read/write access to the data, and when used in combination with the Heartbeat package (discussed in Part II of this book), a backup node has the ability to gain read/write access to the data when the primary node fails. For the latest information, see <http://www.comp-lang.tuwien.ac.at/reisner/drbd>.

## **Intermezzo**

Another distributed open source filesystem with great promise. Intermezzo was “deeply inspired” by the CODA filesystem and entered the Linux kernel at version 2.4.15. However, Intermezzo is not designed for high-performance write access to a single file or database by multiple cluster nodes. For the latest information, see <http://www.inter-mezzo.org>.

## **Lustre**

A cluster filesystem that is capable of supporting tens of thousands of nodes. For the latest information, see <http://www.lustre.org>.

## **Polyserve**

A high performance NAS cluster for Linux that uses a SAN for data storage (<http://www.polyserve.com>).

## **Oracle Cluster File System (CFS)**

Part of the Linux Single System Image (SSI) project. For the latest information, see <http://ssic-linux.sourceforge.net>.

## **Panasas**

For the latest information, see <http://www.panasas.com>.

## **V9fs**

For the latest information, see <http://v9fs.sourceforge.net>.

## **The Open Distributed Lock Manager (DLM) project**

This is a GNU open source project originally started at IBM that will currently work with a two-node Heartbeat configuration. The Open DLM package provides application programming interface (API) routines that will perform lock arbitration. You can then use these lock arbitration APIs if you are writing your own application to access the data. For the latest information, see <http://oss.software.ibm.com/developerworks/projects/dlm>.

## **Open Global File System (GFS)**

This used to be an open source project that turned into a product from the Sistana company that was then purchased by Red Hat. Somewhere along the line the open source version of GFS was revived, so now there are two projects: Open GFS and GFS. At the time of this writing information about GFS is available online at <http://www.redhat.com/software/rha/gfs>.

# F

## **LVS CLUSTERS AND THE APACHE CONFIGURATION FILE**



You will use either IP-based or name-based VirtualHost containers in your Apache configuration file to override the general options set at the top of the file when configuring Apache to run on a real server in an LVS cluster. The IP-based or name-based virtual host containers select the proper directory for web files and CGI scripts<sup>1</sup> for each web host that the real server needs to offer to client computers. We'll discuss when and how to use these two types of Apache virtual hosts later in this appendix, but first let's look at some of the general options you can set in the Apache configuration that will enable the Apache daemon to work properly on an LVS real server.

---

<sup>1</sup> This recipe does not discuss CGI scripting. You will need to look at the general configuration file `ScriptAlias` directive in addition to the general directives mentioned here if you are working with CGI scripts.

You should check these general settings, as discussed in the following sections:

- ServerName
- DocumentRoot
- BindAddress (only for Apache versions prior to 2.0)
- Port (only for Apache versions prior to 2.0)
- Listen (replacement for BindAddress and Port)

## ServerName

Edit the ServerName line in the Apache configuration file (`/etc/httpd/conf/httpd.conf` on Red Hat), and specify the name of your server as follows:

---

```
#vi /etc/httpd/conf/httpd.conf
```

---

Remove the # in the line that looks like this:

---

```
#ServerName <yourserver>
```

---

and insert your server name:

---

```
ServerName www.yourdomain.com
```

---

Replace `www.yourdomain.com` in this example with the default web host name you have registered in DNS for your web server. Or, if you are testing in a lab environment, enter the IP address you want to assign to the machine. (We will override this value in a VirtualHost container shortly.)

## DocumentRoot

Set the DocumentRoot directive to the full path name of the default directory where your HTML files will reside (we will override this default value with a virtual host entry shortly).

---

```
DocumentRoot "/www/htdocs"
```

---

Next, you should modify the first line of the Directory container that follows the DocumentRoot directive to use the same directory you specified for the DocumentRoot. In this example, the line in the configuration file looks like this after you make the change:

---

```
<Directory "/www/htdocs">
```

---

## BindAddress

If you are using an Apache version prior to 2.0, make sure the `BindAddress` directive in this file is commented out, like this:

---

```
#BindAddress *
```

---

This tells the Apache daemon(s) to listen for connection requests on all RIPs used on the real server. If you add an IP address to this directive (and uncomment it), you limit the Apache daemon so that it can only receive requests on a particular IP address. (On Apache 2.0 and newer, use the `Listen` directive instead of the `BindAddress` directive, as described later in this section.)

**NOTE** *Leave the `BindAddress` line commented out, or use it as is (without an IP address).*

## Port

In Apache versions prior to 2.0, the `Port` entry points to the default port used when self-referential URLs in any of your virtual hosts attempt to connect back to the server (without explicitly appending a port number in their URLs). We'll discuss self-referential URLs in more detail later in this appendix. You can specify only one port. You should use either port 80 (the default) or port 443. Use port 443 if you want the default value for self-referential URLs to point to the secured content; otherwise, use port 80.

## Listen

The `Listen` directive replaced the `BindAddress` and `Port` directives in Apache version 2.0 and is now required. It can be used to combine an IP address and a port number that Apache should listen on.

Multiple `Listen` directives are allowed. For example:

---

```
Listen 209.100.100.3:80
Listen 209.100.100.3:443
```

---

While these entries would be valid, because we want Apache to listen on all IP addresses, you should not use an IP address here. If you must control the port numbers Apache listens on, specify them using the `Listen` directive without an IP address, like this:

---

```
Listen 80
Listen 443
```

---

Then start Apache, and check to make sure that it is listening on the proper ports with this command:

---

```
#netstat -apn | more
```

---

You should see netstat reporting that Apache is listening on both ports (80 and 443).

Now that we've looked at the general options you can use in the Apache configuration file, let's look at how you create an Apache virtual host.

## Apache Virtual Host Configuration on Cluster Nodes

If your node(s) only need to serve a single web host address (one DNS registered host name), you would be done at this point with your changes to the `httpd.conf` file. However, you will most likely need to configure each cluster node as a web server able to offer several different addresses or web host names. Apache lets you do so using either IP-based or name-based virtual hosts.

### Apache IP-Based Virtual Hosts

Older web browsers (Netscape version 1.1 or early releases of Internet Explorer) that do not know how to send requests with the web host name embedded (HTTP version 1.0) and HTTPS requests (even when sent from new web browsers) require IP-based virtual hosts.<sup>2</sup>

The Apache web servers running on the real servers inside the cluster can serve several IP-based web hosts from a single real server using IP-based `VirtualHost` containers in the Apache configuration file. Each VIP address the cluster offers to client computers will have one `VirtualHost` container associated with it that looks like this:

---

```
<VirtualHost 209.100.100.3:80 10.1.1.2:80>
#
#      209.100.100.3 is the VIP address known to client computers
#      10.1.1.2 is the RIP address on LVS-NAT cluster node 1
#
#      ServerName www.yourdomain.com
#      ServerAdmin yourname@yourdomain.com
#      DocumentRoot /www/htdocs
#      ErrorLog /var/log/httpd/error_log
#      TransferLog /var/log/httpd/access_log
</VirtualHost>
```

---

<sup>2</sup> IP-based virtual hosts are also required for the Microsoft FrontPage extensions at the time of this writing.

In this example, both the VIP and the RIP addresses are placed on the `VirtualHost` line so we can use the same `httpd.conf` file on both the real server and the Director.

When you add another cluster node, add it to the `VirtualHost` directive, as shown in the following example:

---

```
<VirtualHost 209.100.100.3:80 10.1.1.2:80 10.1.1.3:80>
#
#      209.100.100.3 is the VIP address known to client computers
#      10.1.1.2 is the RIP address on LVS-NAT cluster node 1
#      10.1.1.3 is the RIP address on LVS-NAT cluster node 2
#
ServerName www.yourdomain.com
ServerAdmin yourname@yourdomain.com
DocumentRoot /www/htdocs
ErrorLog /var/log/httpd/error_log
TransferLog /var/log/httpd/access_log
</VirtualHost>
```

---

The format of the `VirtualHost` container is:

---

```
<VirtualHost  VIP:Port  Virtual RIP1-1:Port  Virtual RIP2-1:Port  ... >
    Options that will override the general server options.
</VirtualHost>
```

---

Adding all of the cluster node RIP addresses to the `VirtualHost` directive allows you to use the same Apache configuration file on all cluster nodes instead of forcing you to maintain separate configuration files on each cluster node.

**NOTE** *In an LVS-DR cluster, you can leave out the RIP addresses and only use the VIP addresses because the real servers receive the HTTP packets from the Director on a VIP address.*

## **Name-Based Virtual Hosts**

All web browsers now know how to send a web host name embedded in the HTTP request. This allows you to conserve IP addresses and specify several different web host names (or CNAMEs in DNS) for a single VIP address. When the Apache<sup>3</sup> web server receives a request for a name-based virtual host, it needs to know which virtual host container in the `httpd.conf` file to use.

Apache decides which container to use for name-based virtual hosts by matching the IP address used in the HTTP request with a `VirtualHost` container and then looking inside this container at the `ServerName` directive.

---

<sup>3</sup> Name-based virtual hosting was introduced in Apache version 1.3; for the `NameVirtualHost` directive to accept the wildcard character \* (discussed in a moment), Apache version 1.3.13 or higher is required.

If the `ServerName` directive is equal to the web host name embedded in the client computer's HTTP request, Apache knows it has found the correct container.<sup>4</sup>

For example, if you had registered both `www1.yourdomain.com` and `www2.yourdomain.com` to IP address `209.100.100.3` using DNS, you could create `httpd.conf` entries that looked like this:

---

```
# Enable Name-based virtual hosts on all IP addresses.
NameVirtualHost *

# First name-based virtual web: www1.yourdomain.com
<VirtualHost *>
    ServerAdmin yourname@yourdomain.com
    UseCanonicalName off
    DocumentRoot /www/www1/htdocs
    ServerName www1.yourdomain.com
    ErrorLog /var/log/httpd/error_log
    TransferLog /var/log/httpd/access_log
</VirtualHost>

# Second name-based virtual web: www2.yourdomain.com
<VirtualHost *>
    ServerAdmin yourname@yourdomain.com
    UseCanonicalName off
    DocumentRoot /www/www2/htdocs
    ServerName www2.yourdomain.com
    ErrorLog /var/log/httpd/error_log
    TransferLog /var/log/httpd/access_log
</VirtualHost>
```

---

**NOTE** *First-generation web browsers that do not send a web host name embedded in their HTTP request will be sent, by default, to the first virtual host container defined. (See the workaround to overcome this problem using the `ServerPath` directive on the Apache website.)*

Notice that we have enabled named-based virtual hosts by adding the `NameVirtualHost` directive (before the `VirtualHost` containers).

You cannot use name-based virtual hosts on a web server that will use the SSL protocol (to provide a secure page to enter credit card information, for example). The encrypted SSL secure transport must be established prior to the HTTP connection (prior to the HTTP GET request that tells Apache which virtual host name the client computer is trying to access). One

---

<sup>4</sup> Apache will also match the IP address in the HTTP request against the `NameVirtualHost` directive if you use an IP address instead of the wildcard character `*` in the `NameVirtualHost` directive. If you do this, however, you will need multiple `NameVirtualServer` directives (one for each IP address) outside of the `VirtualHost` containers.

workaround to this problem is to specify the IP address that is shared by the virtual hosts along with HTTP port 80 in the `NameVirtualHost` directive and in each virtual host container, as shown in the following example:

---

```
# Enable Name-based virtual hosts for one IP address
NameVirtualHost 209.100.100.3:80

# First name-based virtual web: www1.yourdomain.com
<VirtualHost 209.100.100.3:80>
    ServerAdmin yourname@yourdomain.com
    UseCanonicalName off
    DocumentRoot /www/www1/htdocs
    ServerName www1.yourdomain.com
    ErrorLog /var/log/httpd/error_log
    TransferLog /var/log/httpd/access_log
</VirtualHost>

# Second name-based virtual web: www2.yourdomain.com
<VirtualHost 209.100.100.3:80>
    ServerAdmin yourname@yourdomain.com
    UseCanonicalName off
    DocumentRoot /www/www2/htdocs
    ServerName www2.yourdomain.com
    ErrorLog /var/log/httpd/error_log
    TransferLog /var/log/httpd/access_log
</VirtualHost>
```

---

Using this configuration will allow you to acquire a security certificate for your IP address (209.100.100.3, in this example) and use the SSL features of Apache to provide for secure and encrypted communication. Describing how to use SSL with Apache is outside the scope of this book.

## Self-Referential (Redirection) URLs

Normally, web page developers do not use the complete web host name, directory, subdirectory, and file name (called the *absolute path name*) for every link they create within a website. Web developers like to use *relative path names* to files (so their code can easily be moved around in the directory structure if need be), and relative path names allow them to leave out the web host name when they create a link on a web page. This type of web page link is called a self-referential (or redirection) URL because the web developer wanted Apache to assume—if the web host name was not included in the link—the URL refers back to the current web host or web server.

## **IP-Based Virtual Hosts and Self-Referential URLs**

With IP-based virtual hosts, the web host server name used in self-referential URLs is, by default, the value you give to the `ServerName` directive inside of the `VirtualHost` container. (Starting with version 2.0 of Apache, the `ServerName` directive can contain a fully qualified host name followed by an optional colon and port number.)

### **REVERSE DNS QUERIES AND SELF-REFERENTIAL URLs**

If, for some reason, you do not want to set the `ServerName` directive inside of each IP-based `VirtualHost` container, you can leave it out and instead set the directive `UseCanonicalName` to `dns`. When you set `UseCanonicalName` to `dns`, Apache will figure out the `ServerName` to use in self-referential URLs by performing a reverse DNS query on the IP address specified in the `VirtualHost` container (the IP address the client used in the HTTP request). This IP address is sent to a DNS server to find out which host name it is associated with. This method will not resolve self-referential URLs on real servers in an LVS-NAT cluster because the IP address will be a RIP address (not a VIP address), and RIP addresses are not stored in DNS.<sup>1</sup> (To resolve self-referential URLs in an LVS-NAT cluster, specify the `ServerName` directive in the `VirtualHost` container, and set the directive `UseCanonicalName` to `on`.)

---

<sup>1</sup> They can be stored in an internal naming service that the cluster nodes use to resolve IP addresses, though this is not normally done.

## **Name-Based Virtual Hosts and Self-Referential URLs**

For name-based virtual hosts, the situation is different and much easier. As shown in the previous name-based virtual host example, the `UseCanonicalName` directive was set to `off`. This means self-referential URLs<sup>5</sup> are set to the host name sent by the client computer (embedded in their HTTP request). The same Apache configuration file can then be used on all of the cluster nodes inside an LVS-NAT cluster without modification, and the self-referential URLs will work properly.<sup>6</sup> (Again, note that with Apache version 2.0 and later, the value you give to the `ServerName` directive can be the fully qualified host name followed by a colon and a port number.)

## **Verify Your Virtual Host Configuration**

When you have modified the Apache configuration file to add a virtual host, you can verify that it is configured properly before you start Apache with the following command.

---

<sup>5</sup> And the CGI script variables `SERVER_NAME` and `SERVER_PORT`.

<sup>6</sup> In Chapter 14, we discuss persistence and explain how to configure your cluster so that self-referential URLs will even end up back on the same cluster node.

---

```
#/usr/sbin/httpd -t -DDUMP_VHOSTS
```

---

**NOTE** *Most versions of Apache also support the same functionality with the command:*

---

```
/usr/sbin/httpd -S
```

---

The output of this command will depend on the virtual host method you selected.

For *IP-based virtual hosts*, both the VIP and the RIP addresses will appear on this report:

---

```
#/usr/sbin/httpd -t -DDUMP_VHOSTS
VirtualHost configuration:
10.1.1.2:80      www.yourdomain.com (/etc/httpd/conf/httpd.conf:1215)
209.100.100.3:80  www.yourdomain.com (/etc/httpd/conf/httpd.conf:1215)
```

---

This example output indicates that line 1215 in the `httpd.conf` file contains the definition for both of these VirtualHost IP addresses.

For *named-based virtual hosts*, the output of this command looks like this:

---

```
#/usr/sbin/httpd -t -DDUMP_VHOSTS
VirtualHost configuration:
wildcard NameVirtualHosts and _default_ servers:
*:*                  www1.yourdomain.com (/etc/httpd/conf/httpd.conf:1215)
*:*                  www2.yourdomain.com (/etc/httpd/conf/httpd.conf:1225)
```

---

This example output indicates that two name-based virtual hosts have been defined in the `httpd.conf` file. When Apache runs as a daemon on this server it will be able to serve content for these two virtual hosts in response to HTTP requests from client computers.

### IF DNS IS NOT CONFIGURED PROPERLY

If you are using IP-based virtual hosts and you used the `UseCanonicalName` DNS directive, this command will cause Apache to perform a reverse DNS query on the IP addresses you have specified in your `VirtualHost` containers. If you don't have DNS configured properly on your cluster nodes, Apache will complain:

---

```
10.1.1.2:80  Bogus_host_without_reverse_dns (/etc/httpd/conf/httpd.conf:1200)
```

---

This error indicates that line 1200 of the `/etc/httpd/conf/httpd.conf` file contains an IP address (the 10.1.1.2 RIP address, in this example) that Apache cannot find in DNS.



# INDEX

## A

absolute path names, 403  
account administration  
    without Active Directory, 342–343  
    for clusters, 189–190  
    in SSH, 78–79  
ACID techniques, 366  
active-active server configuration, 154  
active connections, 203  
Active Directory, 341–343, 361  
addclients program, 100, 105  
Address Resolution Protocol (ARP)  
    GARP, 122–125  
    for LVS-DR cluster broadcasts, 233–234, 236–238  
addresses, IP. *See* IP addresses  
administrative networks, 113  
ADSL (Asymmetric Digital  
    Subscriber Line), 26  
advanced power management, 22  
advisory locking, 280  
aep1000 script, 21  
affinity, port, 250  
alarms in Heartbeat, 157  
alerts  
    Heartbeat, 157–158  
    Mon, 302, 319  
    SNMP, 28  
aliases  
    IP  
        creating and deleting, 120–121  
        failing over, 116

    in haresources file, 148–149  
    in Heartbeat, 116–125  
    for source addresses, 117  
    in linuxconf, 382  
anacron script, 21–23, 29  
Apache  
    CGI programs under, 286  
    configuration, 397–398  
        BindAddress entry, 399  
        on cluster nodes, 400–403  
        DocumentRoot entry, 398  
        Listen entry, 399–400  
        Port entry, 399  
        on real server, 218–219  
        self-referential URLs, 403–404  
        ServerName entry, 398  
        verifying, 404–405  
    for Ganglia, 330, 332  
    superfluous messages in, 260  
apcmaster device, 168  
apcmastersnmp device, 168  
apcsmart device, 168  
API (application programming  
    interface), 3, 396  
apmd script, 22  
AppConfig module, 97  
arbitration, lock, 5, 280  
    kernel, 280–283  
    NLM for, 284–285  
    potential methods, 395–396  
ARP (Address Resolution Protocol)  
    GARP, 122–125  
    for LVS-DR cluster broadcasts, 233–234, 236–238  
arpwatch script, 22

Asymmetric Digital Subscriber Line (ADSL), 26  
atd script, 22, 316  
attribute caching, 291  
audible alarms, 157  
authentication  
    for cluster accounts, 343–345  
    in SSH, 76–77  
auto\_fallback entry, 159–160, 164, 176  
autofs script, 22  
automated processes  
    dependency failure fixing, 392  
    NIC selection process, 149–151  
autoreload entry, 268

## B

backup Directors, Heartbeat  
    installation on, 263  
backup servers in Heartbeat, 112  
    configuration on, 131  
    installation on, 139–140  
    launching on, 142–143  
backupserver entry, 171–172  
bash statements, 18  
batch job systems  
    for clusters, 190  
    in NFS, 292–293  
    with no single points of failure, 354–357  
baud entry, 135  
baytech device, 168  
bcast entry, 135  
bcm5820 script, 22  
big-endian byte ordering, 192  
bin command, 371  
BindAddress entry, 399  
boot floppies  
    failures in, 107  
    for Golden Client, 103–106  
/boot/grub/device.map file, 64  
/boot/grub/menu.lst file, 62  
boot loaders  
    configuring, 62–64  
    and root filesystems, 63–64  
/boot/System.map file, 61  
booting from SCSI disks, 64

bottlenecks, NFS, 288–290  
broadcasts  
    for Heartbeat, 122–125  
    for LVS-DR clusters, 233–234, 236–238  
BSD flock lock arbitration, 281, 284–285  
business tiers in N-Tier applications, 361  
byte offers for BSD flock, 281  
.bz file, 372  
.bz2.tar file, 372  
bzImage file, 60–61

## C

cable connections in Heartbeat, 113  
caching  
    attributes, 291  
    in round-robin DNS, 156  
capacity planning, 190–192  
case studies, 341  
account administration without Active Directory, 342–343  
batch job-scheduling systems, 354–357  
cluster account authentication  
    mechanisms, 343–345  
email, 353–354  
fault-tolerant print spoolers, 345–351  
rebooting nodes for  
    preventative maintenance, 352–353  
cat command, 59  
central print servers  
    custom scripts on, 351  
    /etc/printcap on, 349–350  
    LPRng installation on, 347–348  
    print job management on, 350  
centralizing account  
    administration, 189–190  
CFS (Cluster File System), 396  
CGI programs under Apache, 286  
chain policies, 38  
checkinterval entry, 268  
checkpoint entry, 271  
checktimeout entry, 267–268

checktype entry, 271  
chkconfig command, 18–20, 129, 138, 275  
CIP (client computer's IP)  
    addresses, 195  
cl\_respawn utility, 162  
client-servers for SSH. *See* two-node SSH client-servers  
clients  
    access to  
        in Heartbeat, 115–116  
        LVS-DR cluster services, 232–236  
        LVS-NAT cluster resources, 209–216  
    in data center example, 362–363  
    for Golden Client, 98–99  
    for LVS clusters, 194  
    for LVS-DR clusters and ARP broadcasts, 236–238  
    NFS configuration options, 296–297  
    in SystemImager, 108–109  
clock skew, 58  
cloned systems. *See* SystemImager package  
close-to-open (cto) cache consistency, 291  
Cluster File System (CFS), 396  
cluster node managers, 5–6  
    in data center example, 361–362  
    Ganglia installation on, 330  
    known\_hosts entries on, 355–356  
    SNMP files on, 317–318  
cluster nodes  
    Apache configuration on, 400–403  
    configuration changes for, 187  
    Ganglia for. *See* Ganglia software package  
    in Heartbeat configuration, 135–136  
    least-loaded, 338  
    load averages for, 334–335  
    LPRng installations on, 348  
monitoring  
    by ldirectord, 258–261  
    performance, 189  
net-SNMP client software  
    installation on, 310  
for print spoolers, 345–347  
    /etc/printcap file on, 349–350  
    /etc/printcap.local file on, 348–349  
    print job management on, 351  
purchasing, 190–192  
removing, 188, 352  
software updates on, 189  
sshd\_config file on, 355  
telnet access to, 353  
    weight of, 352  
cluster transition messages, 114  
clusters, 1, 185  
    architecture of, 4–5  
    authentication mechanisms for, 343–345  
    building, 186–192  
    capacity planning, 190–192  
    case studies. *See* case studies  
    filesystems for, 395–396  
    high-availability. *See* high-availability clusters  
    LVS-DR. *See* LVS-DR (Linux Virtual Server Direct Routing) clusters  
    LVS-NAT. *See* LVS-NAT (Linux Virtual Server Network Address Translation) clusters  
    partitioned, 113–114  
    properties of, 2–3  
    single points of failure in, 6–7  
clustersh script, 356  
CMF (Content Management Framework), 342  
CODA filesystem, 395  
command line, testing email from, 354  
common Unix printing system, 23  
compiling  
    fping package, 304

- compiling, *continued*
  - kernel, 53
    - boot loader configuration, 62–64
    - object code and configuration files for, 60–62
    - options for, 56–59
    - process, 59–60
    - requirements for, 54
    - source code for, 54–56
- Comprehensive Perl Archive Network (CPAN)
  - for module downloads, 264–266, 306
  - for Mon package, 305–306
- compression methods, 70, 331, 372–374
- .config file, 56–57
- CONFIG\_NFS\_DIRECTIO option, 296
- configuration files
  - copying with rsync, 91–92
  - Heartbeat, 130, 158–159
  - installing, 60–62
  - ldirectord, 267–272
  - connect entry, 271–272
  - connection templates, 247–248
  - connection tracking records
    - for Director, 242–243
    - timeout values for, 243–244
  - connections
    - in Heartbeat, 113
    - in LVS scheduling, 203
    - persistent. *See* persistence
    - in SSH, 82–86
  - Content Management Framework (CMF), 342
  - control messages in Heartbeat, 114–115
  - converting Unix legacy applications, 192
  - cooperative locking, 280
  - copying
    - encryption keys, 81–82
    - files with rsync, 89, 91–92
  - CPAN (Comprehensive Perl Archive Network)
- for module downloads, 264–266, 306
- for Mon package, 305–306
- cron jobs
  - for batch job-scheduling systems, 356–357
  - from cluster node manager, 361
  - for rsync snapshots, 90, 92
- crond script, 23, 316, 356
- crontab file
  - for batch job-scheduling systems, 356–357
  - format of, 92
  - for rsync snapshots, 90
- crossover cable, 112, 132
- cto (close-to-open) cache consistency, 291
- cups script, 23
- custom metrics, 338–340
- CustomLog entry, 260

## D

- D/RIP network, 195, 241, 288
- daemons, 10
  - for resource ownership, 126–127
  - respawning, 138
  - starting, 15–21
- data center example, 359–360
  - clients in, 362–363
  - cluster node manager in, 361–362
  - clusters in, 360–361
  - final product, 367–368
  - highly available database servers in, 365–366
  - highly available serial devices in, 364–365
  - NAS server in, 363–364
- data synchronization, 69–70
- data tiers in N-Tier applications, 361
- database servers
  - highly available, 365–366
  - in N-Tier applications, 361

deadtime entry, 135, 161  
debugging mode for Mon package, 309–310  
default chain policies, 38  
default routes and gateways  
    for Heartbeat, 150–151  
    for LVS-DR real server, 235–236  
    for LVS-NAT clusters, 219–220  
    for routing, 44–46  
    for SystemImager, 102  
Denial of Service (DoS) attacks, 244, 248  
dependencies, 387–388  
    dynamic executables and  
        shared objects, 388–389  
    failures in, 391–392  
    rpm package shared libraries, 390–391  
    SystemImager server software, 98  
        Yum for, 392–393  
    design goals for high-availability cluster, 256–257  
destination hashing scheduling methods, 202  
destination hosts, matching packets to, 45–47  
`/dev/ttys0` file, 173  
`/dev/watchdog` file, 177–178  
devices, Stonith. *See* Stonith (shoot the other node in the head)  
`df` command, 63, 99, 313  
DHCP servers, 101–103  
`diff` command, 30, 344  
`diffie-hellman-group1-sha1` key exchange process, 74  
DIP (director’s IP) addresses, 195  
direct I/O in NFS, 296  
direct routing, 198–200, 235  
Directors  
    connection tracking table for, 242–243  
    Heartbeat installation on, 263  
    for LVS  
        configuration, 221–224  
        installation, 220  
        redundant, 255–256  
    as real servers, 227–229  
Director’s IP (DIP) addresses, 195  
disaster recovery, 364  
discretionary locking, 280  
diskless workstations, 362  
distributed content, 157  
Distributed Lock Manager (DLM)  
    project, 396  
distributed objects, 3, 361  
distribution kernels, 55  
distribution vendors, 57  
DLM (Distributed Lock Manager)  
    project, 396  
`dmesg` command, 59  
DNS (Domain Name System)  
    with Apache, 401, 405  
    with cluster node manager, 362  
    for iptables and ipchains, 41  
    monitoring, 321  
    reverse queries in, 404  
    round-robin, 155–157  
`DocumentRoot` entry, 398  
DoS (Denial of Service) attacks, 244, 248  
dotlock arbitration, 286  
dotlock files, 281  
downloading  
    FTP for, 369–371  
    Lynx for, 371–372  
    Perl modules, 264–266, 306  
    wget for, 372–373  
DRBD filesystem, 395  
dynamic executables, shared objects for, 388–390  
dynamic HTML, 367  
dynamic scheduling methods, 203–207

## E

`echo` entry, 268–269  
`eject` command, 382  
email, 3, 5  
    for iptables and ipchains, 42–43  
    monitoring, 264  
    sending and receiving, 353–354  
email alerts  
    in Heartbeat, 158  
    in Mon, 319

encryption keys in SSH, 72–73, 77  
    copying, 81–82  
    creating, 79–80  
    exchange process, 73–76  
    host, 79  
endián issues, 192  
EnGarde distribution, 257  
enterprise clusters. *See* clusters  
error logging, 29, 137, 316  
*/etc/anacrontab* file, 21  
*/etc/conf.modules* file, 119  
*/etc/dhcpd.conf* file, 101–102  
*/etc/gmetd.conf* file, 331  
*/etc/gmond.conf* file, 331  
*/etc/group* file, 31, 343–344  
*/etc/ha.d/authkeys* file, 130, 139  
*/etc/ha.d/conf* file, 267  
*/etc/ha.d/ha.cf* file  
    configuring, 134–136  
    for control messages, 115  
    for daemons, 130  
    for ipfail, 175  
    for primary server, 148  
    for respawning, 138, 162  
    for stonith, 170  
    for watchdog, 178  
*/etc/ha.d/haresources* file, 147–148  
    for batch job scheduler, 356  
    configuring, 136–148  
    for IP address takeover, 151–152  
    for IP aliases in, 148–149  
    for NIC selection process, 149–151  
    for primary server, 130, 148  
    for resources, 152–154  
    for Stonith, 170–171  
*/etc/ha.d/resource.d* file, 125, 138, 149  
*/etc/ha.d/resource.d/AudibleAlarm* file, 157  
*/etc/ha.d/resource.d/MailTo* file, 157  
*/etc/hosts* file, 100  
*/etc/httpd/conf/httpd.conf* file, 398  
*/etc/init.d* file, 20, 38, 50  
*/etc/init.d/lpd* file, 347  
*/etc/init.d/lvs* file, 221, 228  
*/etc/init.d/sshd* file, 79  
*/etc/init.d/systemimager-server-*  
    *rsyncd* file, 106  
*/etc/inittab* file, 16–18, 28, 161  
*/etc/lilo.conf* file, 62, 178  
*/etc/mail/sendmail.mc* file, 353–354  
*/etc/modules.conf* file, 119, 380  
*/etc/mon/mon.cf* file, 307–308, 317–318, 323  
*/etc/nsswitch.conf* file, 30  
*/etc/passwd* file, 30–31, 343–345, 353  
*/etc/printcap* file, 348–350  
*/etc/printcap.local* file, 25, 348–349  
*/etc/rc.d/init.d* file, 17  
*/etc/rc.d/init.d/dhcpd* file, 103  
*/etc/rc.d/init.d/sshd* file, 72  
*/etc/rc.d/rc* file, 17–18  
*/etc/sendmail.cf* file, 354  
*/etc/services* file, 307, 376  
*/etc/shadow* file, 31, 87, 343–345  
*/etc/snmp/snmpd.conf* file, 311, 314, 320  
*/etc/ssh/ssh\_host\_rsa\_key* file, 72, 79  
*/etc/ssh/ssh\_host\_rsa\_key.pub* file, 72, 79  
*/etc/ssh/sshd\_conf* file, 83  
*/etc/ssh/sshd\_config* file, 78, 83, 86, 355  
*/etc/stonith.cfg* file, 324  
*/etc/sudoers* file, 99  
*/etc/sysconfig* file, 24–25, 38, 379–380  
*/etc/sysconfig/ipchains* file, 385  
*/etc/sysconfig/iptables* file, 43, 385  
*/etc/sysconfig/network* file, 31, 107, 136, 219, 384  
*/etc/sysconfig/network-scripts* file, 379, 384  
*/etc/sysconfig/network-scripts/ifcfg-eth1* file, 108  
*/etc/sysctrl.conf* file, 261  
*/etc/syslog.conf* file, 29, 137, 269  
*/etc/systemimager/rsyncd.conf* file, 106  
*/etc/systemimager/systemimager.conf* file, 105  
*/etc/systemimager/updateclient.local*  
    *exclude* file, 108  
*/etc/xinetd.conf* file, 30

/etc/yum.conf file, 392

Ethernet

- backbones, 288
- for Heartbeat
  - cable connections in, 113
  - control messages in, 115
  - NIC device names for, 119

events in Stonith

- with Mon, 323–324
- multiple, 174
- sequences of, 167–168

exclusive BSD flocks, 281

executables, shared objects for, 388–389

exit command, 371

Expect programming language, 293

expire\_nodest\_conn variable, 269

expired persistence, 248

exponentially smoothed moving average function, 335

external lock arbitration daemon, 281

EXTRAVERSION number, 57, 60, 64

**F**

fallback configuration, 159–160

failed cluster nodes, removing, 188

failing devices, Stonith for. *See*

- Stonith (shoot the other node in the head)

failover, 10

- in Heartbeat, 161–162
  - license manager, 162
  - with Mon, 324–325
- IPAT for, 116
- stateful, 275–278

fallback entry, 272

fat clients, 362

fault-tolerant print spoolers, 345

- cluster nodes in, 345–347
- /etc/printcap file for, 349–350
- /etc/printcap.local file for, 348–349
- job ordering in, 345–346
- LPRng for, 346–348
- print jobs in, 346–347, 350–351

faxes, 362, 365

fcntl system, 282–283, 285

fdutils package, 157

fg option, 296

file lock arbitration, 281, 286

File Transfer Protocol (FTP)

- for downloading software, 369–371
- for iptables and ipchains, 40–41
- monitoring, 264
- persistent connections, 248

filesystems

- for clusters, 395–396
- root, 63–64

filters for packets, 34–38

FIN packets, 243–244

findif program, 149–150

finding locks held by kernel, 286–288

firewalls, 34, 92–93

firstboot script, 23

fixed scheduling methods, 202–203

floating licenses, 246–247

floppycontrol utility, 157

force-reload command, 129

forcing

- CPAN module installation, 265
- Heartbeat failover, 324–325
- standby mode, 160
- Stonith events, 323–324
- system reboots, 178

FORWARD hooks, 245–246

FORWARD rules, 35–37

fping.monitor script

- paths in, 319
- running, 308–309

fping package

- compiling and installing, 304
- path to, 319

FTP (File Transfer Protocol)

- for downloading software, 369–371
- for iptables and ipchains, 40–41
- monitoring, 264
- persistent connections, 248

functions script, 23, 126

fwmarked packets, 250–253

## G

Ganglia software package, 327  
    command utilities in, 328  
    for custom metrics, 338–340  
    gmetad and gmond  
        configuration for, 331–332  
    installing, 330  
    prerequisite packages for, 329–330  
    Web package, 333–337  
GARP (gratuitous ARP) broadcasts, 122–125  
gate option, 270  
gateways, default, 44  
get command, 371  
GETATTR operations, 290–291, 340  
getimage command, 100, 108  
GFS (Global File System), 396  
GLIBC packages, 388–391, 393  
gmetad daemon, 328  
    configuring, 331  
    starting, 332  
gmetric scripts, 338–340  
gmond daemon, 328  
    configuring, 331  
    starting, 332  
Golden Client  
    boot floppies for, 103–106  
    client software for, 98–99  
    cloning, 95  
    image installation for, 106–107  
    system images for, 99–101  
gpm script, 23  
graphs in Ganglia Web package, 334–337  
gratuitous ARP (GARP) broadcasts, 122–125  
grids in Ganglia, 328, 334  
GRUB bootloader, 62–64  
grub.conf file, 63  
gstat utility, 328, 338  
gunzip command, 372  
.gz file, 372

## H

halt script, 23  
Happy Tobi Project, 329  
hard option, 296  
haresources file. *See*  
    /etc/ha.d/haresources file  
hash buckets, 242  
hash table structure, 242  
hashing scheduling methods, 202–203  
hb\_standby command, 160, 180, 324  
Healthcheck web page, 272–273  
Heartbeat, 112  
    client computer resource access  
        in, 115–116  
    configuration, 130–131, 158–159  
    backup servers, 139–140  
    /etc/ha.d/authkeys for, 139  
    /etc/ha.d/ha.cf for, 134–136  
    /etc/ha.d/haresources for.  
        *See* /etc/ha.d/haresources  
        file  
    installation, 132–134  
    launching, 140–144  
    ldirectord in, 274–275  
    preparations for, 132  
    resource monitoring, 145  
    stopping and starting, 144–145  
    system time, 140  
    testing, 179–181  
control messages in, 114–115  
deadtime value for, 135, 161  
failover in, 161–162  
    license manager, 162  
    with Mon, 324–325  
GARP broadcasts in, 122–125  
for high-availability clusters, 263  
informational messages in logs, 161  
load sharing with, 154–157  
maintenance, 158–162  
network failures in, 175–176  
operator alerts in, 157–158  
physical paths in, 112–114

respawning with, 138, 161–162  
scripts for, 125–129  
secondary IP addresses and IP  
  aliases in, 116–125  
service offers in, 121–122  
Stonith for. *See* Stonith (shoot  
  the other node in the  
  head)  
watchdog for, 176–179  
hiding loopback interfaces, 261–263  
high-availability clusters, 10–11, 255  
  design goals for, 256–257  
  Healthcheck web page for,  
    272–273  
  Heartbeat installation for, 263  
ldirectord. *See* ldirectord  
  program  
loopback interface hiding in,  
  261–263  
LVS-DR, 257–258  
NAS server, 364  
redundant LVS Directors,  
  255–256  
stateful failover for  
  of IPVS table, 275–276  
  modifications for, 277–278  
highly available database servers,  
  365–366  
highly available serial devices,  
  364–365  
holdall command, 351  
hooks, Netfilter  
  for load balancer, 244–246  
  rules for, 34–35  
host encryption keys, 72, 79  
Host Report web page, 336–337  
hostbased authentication, 76  
hostname command, 136  
hosts  
  matching packets to, 45–47  
  virtual, 400–405  
HTTP  
  for iptables and ipchains, 43  
  with ldirectord, 264  
httpd script, 23  
hung systems, 176–179  
hwclock command, 58  
HylaFax program, 361–362, 365

i  
i/o fencing, 113  
I/O operations  
  measuring, 294–295  
  in NFS, 296  
I/P port entry, 383  
ICMP (Internet Control Message  
  Protocol)  
  with firewalls, 385  
  for iptables and ipchains, 43  
  ping packets in, 223, 385  
  tcpdump for, 377  
identd script, 24  
IEEE Task Force, 2  
ifconfig command  
  for cluster configuration, 224  
  for IP aliases, 120  
  for NICs, 119, 380, 383  
ifup command, 383  
inactive connections in LVS  
  scheduling, 203  
@INC file, 266  
informational log messages, 161  
init scripts, 16–17  
  on cluster nodes, 21–31  
  as Heartbeat resource scripts,  
    128–129  
  for Mon, 318  
  for respawning services, 18  
  symbolic links for, 18–21  
initdeadline entry, 135  
initiate.stonith.event script, 324  
inittab file, 16–17  
INPUT rules  
  for ipchains, 39–40  
  in Netfilter, 35–37  
insmod command, 177, 380  
installing  
  CPAN modules, 265, 305–306  
  fping programming, 304  
  Ganglia, 330  
  Heartbeat, 132–134, 139–140,  
    263  
  kernels, 56  
  ldirectord, 266  
  LPRng, 347–348  
  LVS software, 220

installing, *continued*  
    Mon software, 306–307  
    NICs, 381–382  
    object code and configuration  
        files, 60–62  
    rpm packages, 392–393  
    SNMP package, 304–305  
    System.map file, 61  
    SystemImager server software,  
        97–98

Integrated Services Digital  
    Networks (ISDN), 24

interactive user applications in NFS,  
    292–293

Intermezzo filesystem, 396

Internet Control Message Protocol  
    (ICMP)  
        with firewalls, 385  
        for iptables and ipchains, 43  
        ping packets in, 223, 385  
        tcpdump for, 377

IP address takeover (IPAT), 116

IP addresses  
    in Heartbeat, 116–125, 131  
    in linuxconf, 383  
    in LVS, 194–196  
    names for, 119–120  
    ownership of, 127–128  
    takeover of, 151–152  
    virtual, 194–195, 214–216, 241

IP aliases  
    creating and deleting, 120–121  
    failing over, 116  
    in haresources file, 148–149  
    in Heartbeat, 116–125  
    for source addresses, 117

IP-based virtual hosts, 400–401, 404

ip command  
    for packets, 50  
    for secondary IP addresses,  
        119–120

ip-request messages, 114

ip-request-resp messages, 114

ip route list command, 47

IP tunneling, 200–202

IP Virtual Server (IPVS) software,  
    193

ip\_vs\_timeout\_table tables, 244

IPAddr script, 150

IPAT (IP address takeover), 116

ipchains and iptables  
    clearing existing rules for, 39  
    default policies in, 38  
    DNS for, 41  
    email for, 42–43  
    firewall rules for, 92–93  
    FTP for, 40–41  
    HTTP for, 43  
    ICMP for, 43  
    INPUT policy for, 39–40  
    marketing packets with,  
        252–253  
    in Netfilter, 35–37  
    for NICs, 385  
    routing scripts for, 47–50  
    SSH for, 42  
    tcpdump for, 376  
    telnet for, 42

ipchains script, 24

ipfail plug-in, 175–176

ipfwadm command, 222

ipip option, 270

iproute command, 120

iptables script, 24

iptakeover script, 152–153

IPVS (IP Virtual Server) software,  
    193

IPVS table, stateful failover of,  
    275–276

ipvsadm command  
    for cluster nodes, 352–353  
    for connection tracking,  
        243–244  
    for Director, 221–223, 228  
    for timeouts, 247–248

irda script, 24

ISDN (Integrated Services Digital  
    Networks), 24

isdn script, 24

**J**

job management in print spoolers,  
    345–346, 350–351

job-scheduling system  
for clusters, 190  
with no single points of failure,  
354–357

## K

kdcrotate script, 24  
keepalive entry, 135  
Kerberos security, 24  
kernel lock arbitration, 280  
methods, 281–283  
and NLM, 284–285  
kernels  
for clusters, 187  
compiling, 53  
boot loader configuration,  
62–64  
object code and  
configuration files for,  
60–62  
options for, 56–59  
process, 59–60  
requirements for, 54  
source code for, 54–56  
finding locks held by, 286–288  
hanging, 178  
installing, 56  
modular, 57–59, 119, 380  
for NICs, 380  
for packets, 44–50  
upgrading and patching, 56–57  
keys, encryption, 72–73, 77  
copying, 81–82  
creating, 79–80  
exchange process, 73–76  
host, 79  
keytable script, 24  
kill command, 17, 158  
killall command, 24, 179  
klockd daemon, 25  
known\_hosts database  
on cluster node manager,  
355–356  
for SSH, 74–75  
kudzu script, 24

## L

LAN Information Server service, 24  
latency, NFS, 293  
launching Heartbeat, 140  
backup primary servers,  
142–143  
log files for, 143–144  
on primary servers, 140–142  
LBLC (locality-based least-  
connection) scheduling  
method, 206–207  
LBLCR (locality-based least-  
connection with  
replication) scheduling  
method, 207  
LC (least-connection) scheduling  
method, 204  
1cd command, 370–371  
LDAP systems, 342–343  
ldapsearch command, 344  
1dd command, 390  
ldirectord program  
cluster node monitoring by,  
258–261  
configuration file for, 267–272  
in Heartbeat configuration,  
274–275  
installing, 266–267  
Perl modules for, 263–266  
starting and testing, 273–274  
least-connection (LC) scheduling  
method, 204  
least-loaded cluster nodes, 338  
legacy account administration  
methods, 342–343  
legacy applications, converting, 192  
/lib/modules file, 57, 59, 61, 64  
libraries, shared, 390–391  
license manager failover, 162  
licenses, floating, 246–247  
LIDS (Linux Intrusion Detection  
System), 77, 257, 355  
LILO bootloader, 62–64  
lilo.conf file, 62  
links  
library, 388

links, *continued*  
    symbolic  
        chkconfig for, 18–20  
        ntsysv for, 20–21  
        for source code, 56

Linux Intrusion Detection System (LIDS), 77, 257, 355

Linux Standard Base (LSB)  
    Specification, 128–129

Linux Terminal Server Project (LTSP), 362–363

Linux Utility for cluster Installation (LUI) project, 108

Linux Virtual Server. *See* LVS (Linux Virtual Server)

Linux Virtual Server Direct Routing clusters. *See* LVS-DR (Linux Virtual Server Direct Routing) clusters

Linux Virtual Server Network  
    Address Translation clusters. *See* LVS-NAT (Linux Virtual Server Network Address Translation) clusters

linuxconf command, 379–384

lisa script, 24

listen entry, 399–400

little-endian byte ordering, 192

lmgd daemon, 162

load averages, 334–335

load balancer, 4, 239  
    Director connection tracking table for, 242–243  
    LVS for  
        with Netfilter, 239–241  
        with persistence, 247–253  
        without persistence, 246–247  
    return packets and Netfilter hooks for, 244–246  
    timeout values for, 243–244

load sharing in Heartbeat, 154–157

local.cfg file, 104–105

LOCAL\_IN hook, 240–241, 251

local passwords in authentication, 343–345

locality-based least-connection (LBLC) scheduling method, 206–207

locality-based least-connection with replication (LBLCR) scheduling method, 207

LocalNode configuration, 227–229

locate command, 391

lock arbitration, 5, 280–281  
    kernel, 281–283  
    NLM for, 284–285  
    potential methods, 395–396

lock arbitrators, 280–283

lock files, 281

lockd daemon, 281, 283–284

lockdemo command, 287

lockf method, 281, 285

locks  
    finding, 286–288  
    managing, 291  
    NLM for, 283–285  
    performance issues in, 288–290

log files  
    in Heartbeat, 141  
        informational messages in, 161  
    on primary servers, 143–144  
    for Mon package, 309

logfile command, 141

logfile entry, 269

logger command, 137

loopback devices, 233–235

loopback interface, hiding, 261–263

lpc command, 350–351

lpd script, 25, 108

LPRng package  
    child processes in, 284  
    for fault-tolerant print spoolers, 346–348  
    installing, 347–348

ls command, 59, 371

LSB (Linux Standard Base)  
    Specification, 128–129

lsmod command  
    for loaded modules, 59  
    for NICs, 380  
    for watchdog, 177, 179

- `lsof` command, 287
- `lspci` command, 19
  - for loaded modules, 59
  - for NICs, 119, 380–381
- LTSP (Linux Terminal Server Project), 362–363
- LUI (Linux Utility for cluster Installation) project, 108
- Lustre filesystem, 396
- LVS (Linux Virtual Server), 193–194
  - Apache configuration files for.
 *See* Apache
  - IP address name conventions in, 194–196
  - for load balancer, 239–241
  - monitoring, 258–261
  - with persistence, 247–253
  - without persistence, 246–247
  - redundant Directors, 255–256
  - scheduling methods in
    - dynamic, 203–207
    - fixed, 202–203
  - types of, 196–202
- LVS-DR (Linux Virtual Server Direct Routing) clusters, 198, 231–232
  - ARP broadcasts for, 236–238
  - building, 188
  - client computer access to services, 232–236
  - high-availability, 257–258
  - properties of, 199–200
- LVS-NAT (Linux Virtual Server Network Address Translation) clusters, 196–198, 209
  - building, 187, 216
    - Apache configuration and starting for, 218–219
    - default routes for, 219–220
    - LVS configuration for, 221–224
    - LVS installation for, 220
    - operating system installation for, 217
    - testing configuration for, 224–227
- client computer access to resources, 209–216
- LocalNode configuration for, 227–229
- virtual IP addresses for, 214–216
- LVS-TUN (IP tunneling), 200–202
- Lynx program
  - for downloading software, 371–372
  - for web pages, 228

## M

- `m4` macro processor, 308
- MAC addresses, 22, 233–234
- `mail.alert` script, 308–309
- `mail` command, 354
- `MailTo` script, 158
- `make bzImage` command, 59
- `make clean` command, 56, 59
- `make dep` command, 59
- `make menuconfig` command, 57–58
- `make modules` command, 59
- `make modules_install` command, 59–60
- `make mrproper` command, 56
- `make oldconfig` command, 56–57
- `make process`, failures in, 61
- Management Information Base (MIB)
  - in Mon, 303
  - for SNMP, 312–313
- marked packets, 250–253
- `masq` option, 270
- matching packets
  - to destination hosts, 45–47
  - to networks, 45
- MD5 checksum process, 264
- measuring
  - I/O operations, 294–295
  - NFS latency, 293
- `meatclient` command, 170
- meatware devices, 168–173
- Message Passing Interface (MPI), 3
- messages
  - Heartbeat, 114–115
  - superfluous, 260

MIB (Management Information Base)  
in Mon, 303  
for SNMP, 312–313  
`mkautoinstalldiskette` command, 103–104  
`mkdhcpserver` utility, 101  
`mkintrd` command, 64  
MLDBM module, 97  
`/mnt/floppy` file, 104  
modems, high-availability, 365  
`modprobe` command, 177  
modular kernels, 57–59, 119, 380  
Mon package, 301–302  
compiling and installing, 304  
CPAN modules for, 305–306  
debugging mode for, 309–310  
email alerts in, 319  
`/etc/mon/mon.cf` file for, 307–308  
Heartbeat failover with, 324–325  
init script for, 318  
logs for, 309  
for network failures, 175  
running, 302–303  
SNMP operation in  
package installation for, 304–305  
proof of concept, 310–315  
real word, 315–319  
scripts for, 320–322  
software installation for, 306–307  
Stonith events with, 323–324  
testing installation of, 308–309  
`mon` script, 318  
monitoring  
cluster nodes  
with `ldirectord`, 258–261  
performance, 189  
Heartbeat resources, 145  
Mon scripts, 302  
SNMP, 325  
monitoring agents for, 313–314  
scripts, 321–322  
monolithic kernels, 380  
mounting NFS directories, 22

MPI (Message Passing Interface), 3  
multicasts, 115, 276  
multiple IP aliases, failing over, 116  
multiple NAS servers, 292–293  
multiple NFS transactions, 290  
multiple Stonith events, 174  
`mv` command, 56

## N

N-Tier applications, 361  
name-based virtual hosts, 401–405  
named service, 41  
names  
clusters, 187  
IP addresses  
in LVS, 194–196  
secondary, 119–120  
NIC devices, 119  
primary servers, 148  
`NameVirtualHost` directive, 402–403  
NAS servers  
for clusters, 186–187  
in data center example, 363–364  
for high availability, 256  
multiple, 292–293  
performance of, 295  
NAT (network address translation).  
See `LVS-NAT` (Linux Virtual Server Network Address Translation) clusters  
`ncftp` utility, 369–371  
`ncurses` package, 54  
`negotiate` method, 271  
net-SNMP client software  
installation, 310  
`net-snmp` package, 134  
`net-snmp-devel` package, 304  
`net-snmp-version.tar` file, 304  
Netfilter system  
for clusters, 187  
history of, 35–38  
for load balancer, 239–241, 244–246  
Marked Packets in, 250–253  
for packets, 34–35  
`netfs` script, 25

netmasks, 123, 151, 383  
`netsnmp-exec.monitor` script, 321–322  
`netsnmp-freespace.monitor` script,  
314–315  
`netsnmp-proc.monitor` files, 317–318  
`netstat` command  
    for Apache configuration, 400  
    for HTTPD, 226  
    for network queues, 107  
    for routing rules, 47  
    for sshd, 83  
    with `tcpdump`, 376  
network address translation (NAT).  
    *See* LVS-NAT (Linux Virtual  
    Server Network Address  
    Translation) clusters  
Network File System. *See* NFS  
    (Network File System)  
network interface aliases. *See* IP  
    aliases  
Network interface cards. *See* NICs  
    (network interface cards)  
Network Lock Manager (NLM),  
    283–284  
    and kernel lock arbitration,  
        284–285  
    performance issues of, 288–290  
network script, 25  
Network Time Protocol daemon, 26  
networks  
    failures in, 175–176  
    matching packets to, 45  
never queue (NQ) scheduling  
    method, 205  
NFS (Network File System)  
    attribute caching in, 291  
    batch jobs in, 292–293  
    client configuration in, 296–297  
    developing, 297  
    directory mounting for, 22  
    I/O operations in, 294–296  
    interactive user applications in,  
        292–293  
    latency in, 293  
    locks in, 280–283  
        finding, 286–288  
        managing, 291  
    NLM for, 283–285  
    NAS performance in, 295  
    performance issues in, 288–290,  
        338–339  
    resources for, 297–298  
NFS Extensions for Parallel Storage  
    project, 297  
`nfs` script, 25  
`nfslock` script, 25  
`nfsstat` command, 294  
`nice` command, 292  
`nice_fallback` entry, 159–160  
NICs (network interface cards),  
    379–380  
    configuration for, 381–384  
    device names for, 119  
    existing configuration for, 380  
    in `haresources`, 149–151  
    installing, 381–382  
    for load balancer, 245–246  
    for LVS-DR clusters, 233–235  
    monolithic vs. modular kernels  
        for, 380  
    testing, 383–385  
NIS/LDAP gateways, 30  
NIS systems  
    for cluster account  
        authentication, 343–345  
    for cluster node manager, 361  
    for legacy Unix account  
        administration, 342–343  
    for `ypbind`, 30–31  
    for `ypasswd`, 31  
NLM (Network Lock Manager),  
    283–284  
    and kernel lock arbitration,  
        284–285  
    performance issues of,  
        288–290  
No Single Point of Failure (NSPoF)  
    in batch job-scheduling systems,  
        354–357  
    cluster architecture for, 6–7  
    in Heartbeat, 111  
    in high-availability  
        configuration, 10, 255  
    in print spoolers, 345  
nodes. *See* cluster nodes  
`nohup.out` file, 60

non-dynamic scheduling methods, 202–203  
NQ (never queue) scheduling method, 205  
ns\_rexmit messages, 115  
nscd script, 25  
ntpd script, 26, 316, 319  
ntsysv program, 20–21  
nw\_rpc100s device, 168

## 0

objdump command, 389  
object code, installing, 60–62  
Oeriker, Tobi, 329  
Open Distributed Lock Manager (DLM) project, 396  
Open Global File System (GFS), 396  
Open SSH2. *See* SSH (secure shell)  
operating system for LVS-NAT clusters, 217  
operator alerts in Heartbeat  
    audible alarms, 157  
    email alerts, 158  
Oracle Cluster File System (CFS), 396  
OUTPUT rules in Netfilter, 35–37  
overhead values in scheduling, 204–205  
ownership of Heartbeat resources, 126–128

## P

packages for kernel compilation, 54  
packets and packet routing, 33–34  
    for clusters, 187  
    default chain policies, 38  
    ip for, 50  
    iptables and ipchains, 39, 252–253  
    kernel for, 44–50  
    with load balancer, 240–241, 244–246  
    marked, 239, 250–253  
    matching, 45–47  
    Netfilter for, 34–38

Panasas filesystem, 396  
partitioned clusters, 113–114  
passive FTP, 41  
passwords in authentication, 343–345  
patch command, 30, 344  
patching kernel, 56–57  
path names for links, 403  
paths in Heartbeat, 112–114  
PCCs (persistent client connections), 249  
pcmcia script, 26  
perceptions of NFS performance, 288–290  
performance  
    cluster node, 189–191  
    Ganglia for. *See* Ganglia  
        software package  
    gmetric for, 338–340  
    NAS, 295  
    NFS, 288–290, 388  
Perl modules  
    for ldirectord, 263–266  
    for SystemImager, 97–98  
persistence  
    in LVS, 247–253  
        connection templates for, 247  
        connection types for, 248–249  
    Netfilter Marked Packets, 250–253  
    PCC, 249  
    port affinity, 250  
    PPC, 249–250  
    LVS without, 246–247  
persistent client connections (PCCs), 249  
persistent connection templates, 247–248  
persistent port connections (PPCs), 249–250  
PHP for Ganglia, 329  
physical paths in Heartbeat, 112–114  
pidof program, 126  
PIDs (process identifiers)  
    determining, 126  
    with locks, 287–288

ping command and packets  
    for ICMP, 223, 385  
    for NICs, 384–385  
ping entry for ipfail, 176  
pipes in NFS performance, 288  
Plone clusters, 342, 351  
points of failure. *See* No Single Point of Failure (NSPoF)  
Polyserve filesystem, 396  
port affinity, 250  
Port entry, 399  
portmap script, 26  
Posix fcntl system, 282–283  
Posix kernel lock arbitration, 285  
POST\_ROUTING hook, 240–241  
power control, Stonith for. *See* Stonith (shoot the other node in the head)  
PPCs (persistent port connections), 249–250  
pppoe script, 26  
PRE\_ROUTING hook, 240–241  
prepareclient command, 99, 108  
presentation tiers, 361  
preventative maintenance, rebooting nodes for, 352–353  
primary Directors  
    Heartbeat installation on, 263  
    LVS, 255–256  
primary IP addresses, 116  
primary names in linuxconf, 382  
primary servers in Heartbeat, 112  
    configuration, 131  
    in haresources file, 148  
    launching Heartbeat on, 140–142  
    log files on, 143–144  
    standby mode for, 160  
primaryserver entry, 171  
print job management, 346–347  
    on central print servers, 350  
    on cluster nodes, 351  
print servers, 5  
    custom scripts on, 351  
    /etc/printcap on, 349–350  
    LPRng installation on, 347–348  
    print job management on, 350  
print spoolers, fault-tolerant, 345  
    cluster nodes in, 345–347  
    /etc/printcap file for, 349–350  
    /etc/printcap.local file for, 348–349  
    job ordering in, 345–346  
    LPRng for, 346–348  
    print jobs in, 346–347, 350–351  
printing systems for clusters, 190  
private encryption keys, 72  
/proc/cpuinfo file, 59  
/proc/ksyms file, 61  
/proc/locks file, 286  
/proc/net/ip\_vs\_lblcr file, 207  
/proc/net/route file, 149  
/proc/net/rpc/nfs file, 339  
/proc/sys/net/ipv4/conf/all/arp\_announce file, 262  
/proc/sys/net/ipv4/conf/all/arp\_ignore file, 262  
/proc/sys/net/ipv4/conf/lo/arp\_announce file, 262  
/proc/sys/net/ipv4/conf/lo/arp\_ignore file, 262  
/proc/sys/net/ipv4/ip\_forward file, 45, 49–50, 221  
/proc/sys/net/ipv4/vs/expire\_nodest\_conn file, 268  
/proc/sys/net/ipv4/vs/secure\_tcp file, 244  
process identifiers (PIDs)  
    determining, 126  
    with locks, 287–288  
processes, 10  
procinfo utility, 335  
products in Zope, 367  
prompt, cluster node examinations  
    from, 337–338  
protocol entry, 271  
proxy servers, 206–207  
public encryption keys  
    copying, 81–82  
    in SSH, 72  
purchasing cluster nodes, 190–192  
PuTTY client, 367  
pxe script, 26  
Python programs, 367

## Q

queries, DNS reverse, 404  
quiescent entry, 268  
quorum elections, 164

## R

random script, 26  
rankings in weighted round-robin scheduling, 202  
rawdevices script, 26  
rc program, 17  
reads in fcntl locking, 282  
real entry, 269–270  
real IP (RIP) addresses, 195  
real servers  
    Apache configuration and starting on, 218–219  
    default route setting on, 219–220  
    Directors as, 227–229  
    ldirectord for, 258–261  
    for LVS clusters, 194  
    stateful failover modifications for, 277–278  
    virtual IP addresses on, 214–216  
rebooting  
    for preventative maintenance, 352–353  
    with watchdog, 178  
red cluster nodes, 335  
Red Hat init scripts, 21–31  
redirection URLs, 403–404  
redundant LVS Directors, 255–256  
refreshing Ganglia Web package, 333  
rehash command, 382  
relative path names for links, 403  
reload command, 129  
removing  
    cluster nodes, 188, 352  
    services, 21  
resource groups, file, 153  
resources  
    in Heartbeat  
        client computer access to, 115–116  
    in haresources file, 152–154  
monitoring, 145  
scripts for, 125–129  
LVS-NAT cluster, accessing, 209–216  
respawn entry, 162  
respawn hacluster entry, 176  
respawning  
    in Heartbeat, 138, 161–162  
    services, 18  
restart command, 129  
retransmission request messages, 115  
return packets, 244–246  
reverse DNS queries, 404  
rexmit-request messages, 115  
rhnsd script, 27  
RIP (real IP) addresses, 195  
rm command, 56  
root filesystems, 63–64  
round-robin DNS, 155–157  
round-robin (RR) scheduling methods, 202  
route command, 47  
routes and routing  
    default, 44  
    for iptables, 47–50  
    for LVS-NAT clusters, 219–220  
    packets. *See* packets and packet routing  
    rules for, 47  
routing start command, 50  
routing status command, 50  
rpm command, 77  
rpm files, 373  
.rpm.bz file, 372  
.rpm.bz2 file, 372  
.rpm.gz file, 372  
rpm packages  
    installing, 392–393  
    shared libraries for, 390–391  
rps10 device, 168  
RR (round-robin) scheduling methods, 202  
RRDtool, 329  
rstatd script, 27  
rsync command  
    for batch jobs, 292  
    for copying files, 89, 91–92

- over slow WAN connections, 89–90
- snapshots with, 90–92
- with SSH. *See* SSH (secure shell)
- synchronizing servers with, 69–70
- in SystemImager servers, 106
- rules for iptables and ipchains, 39
  - DNS, 41
  - email, 42–43
  - FTP, 40–41
  - HTTP, 43
  - ICMP, 43
  - INPUT policy, 39–40
  - reviewing and saving, 43–44
  - SSH, 42, 92–93
  - telnet, 42
- runlevel command, 17
- rup command, 27
- rusersd script, 27
- rwall script, 27
- rwhod script, 27

**S**

- Samba package, 28
- sample Heartbeat in Stonith, 166
- SASL (Simple Authentication and Security Layer), 26
- saslauthd script, 27
- saving
  - iptables rules, 43–44
  - kernel configuration files, 62
- /sbin/ifconfig file, 123
- /sbin/ip file, 128
- /sbin/route file, 124
- scheduler entry, 271
- schedules and scheduling methods
  - batch jobs with no single points of failure, 354–357
  - in LVS, 194
    - dynamic, 203–207
    - fixed, 202–203
  - rsync snapshots, 90–92
  - scripts for, 21–23
- scp command, 139–140
- scripts
  - arguments for, 153–154
  - on central print servers, 351
  - for cluster account
    - authentication, 344–345
  - for Heartbeat, 125–129
  - init, 16–17
    - on cluster nodes, 21–31
    - for Heartbeat resources, 128–129
    - for Mon, 318
    - for respawning services, 18
    - symbolic links for, 18–21
  - Mon, 302
  - SNMP
    - creating, 320–321
    - monitoring, 321–322
  - SCSI disks, booting from, 58, 64
  - secondary IP addresses
    - in Heartbeat, 116–125
    - names for, 119–120
  - secondary LVS Directors, 255–256
  - secondaryserver entry, 171
  - secure data transport, 71–72, 86
  - secure shell. *See* SSH (secure shell)
  - Secure Socket Layer (SSL)
    - encryption, 21
  - security control messages, 115
  - security in SSH, 86–87
  - SED (shortest expected delay)
    - scheduling method, 205
  - self-referential URLs, 403–404
  - send\_arp program, 122
  - sendmail daemon, 354
  - sendmail script, 27
  - sequences, event, in Stonith, 167–168
  - serial cable connections, 113
  - serial devices, highly available, 364–365
  - serial entry, 135
  - serial-to-IP communication devices, 365
  - ServerName entry, 398
  - ServerPath directive, 402, 404

servers  
  DHCP, 101–103  
  failures in, Stonith for. *See* Stonith (shoot the other node in the head)  
  in Heartbeat, 112  
    configuration on, 131  
    in `haresources` file, 148  
    installation on, 139–140  
    launching on, 140–143  
    log files on, 143–144  
    maintenance of, 159–160  
    standby mode for, 160  
  highly available, 365–366  
  in N-Tier applications, 361  
  print, 5  
    custom scripts on, 351  
    `/etc/printcap` on, 349–350  
    LPRng installation on, 347–348  
    print job management on, 350  
    proxy, 206–207  
    software updates on, 189  
    synchronizing, 69–70  
    for SystemImager, 97–98  
  service command, 21  
  service entry, 270  
  services, 10  
    Heartbeat  
      offers, 121–122  
      status of, 125–126  
    LVS-DR clusters, client  
      computer access to, 232–236  
    removing, 21  
    respawning, 18  
    starting, 15–21  
  session keys in SSH, 73  
  shared BSD flocks, 281  
  shared data in N-Tier applications, 361  
  shared libraries, 390–391  
  shared objects, 388–389  
  shared secrets, 73  
  shared storage devices, 5  
  shell prompt, cluster node  
    examinations from, 337–338  
shoot the other node in head. *See* Stonith (shoot the other node in the head)  
shortest expected delay (SED) scheduling method, 205  
`show ip arp` command, 179–180  
shutdown program, 17  
`SIGHUP` command, 158  
Simple Authentication and Security Layer (SASL), 26  
Simple Network Management Protocol. *See* SNMP (Simple Network Management Protocol)  
single node failures, 191  
single points of failure. *See* No Single Point of Failure (NSPoF)  
single script, 27  
single transactions in NFS performance, 289–290  
`sk_buff` structure, 233, 239, 246, 250–251  
skew, clock, 58  
slow WAN connections, `rsync` over, 89–90  
`smb` script, 28  
snapshots  
  for Ganglia Web page, 334–337  
  `rsync`, 90–92  
SNMP (Simple Network Management Protocol), 301, 315  
  alerts in, 28  
  package installation for, 304–305  
  proof of concept operations, 310–315  
  real word operations, 315–319  
  scripts for  
    creating, 320–321  
    monitoring, 321–322  
  software installation for, 306–307  
`snmpd.conf` file, 315–317  
`snmpd` daemon, 28, 301, 303, 317  
`snmptranslate` command, 312  
`snmptrapd` script, 28

`snmpvar.monitor` script, 325  
`snmpwalk` command, 312–313, 322  
socket buffer structure, 239,  
    250–251  
source addresses, IP aliases for, 117  
source code  
    acquiring, 54–56  
    compiling, 59–60  
source hashing scheduling methods  
    in LVS, 203  
split-brain. *See* Stonith (shoot the  
    other node in the head)  
spoolers, fault-tolerant, 345  
    cluster nodes in, 345–347  
    `/etc/printcap` file for, 349–350  
    `/etc/printcap.local` file for,  
        348–349  
    job ordering in, 345–346  
    LPRng for, 346–348  
    print jobs in, 346–347, 350–351  
SQL servers, 365–366  
squid script, 28, 206  
SSH (secure shell), 71  
    encryption keys in, 72–73, 77  
    copying, 81–82  
    creating, 79–80  
    exchange process, 73–76  
    host, 79  
    for iptables and ipchains, 42  
    for least-loaded clusters, 338  
    rsync over, 87–89  
    for secure data transport, 71–72  
    in Stonith, 168  
two-node client-servers. *See* two-  
    node SSH client-servers  
two-way trust relationships in,  
    73–77  
version 1, 83  
`ssh-agent` daemon, 77  
`ssh-keygen` command, 72, 80, 355  
`sshd_config` file, 92–93, 355  
`sshd` daemon, 28, 355, 360–361  
SSHD file, 78  
SSL (Secure Socket Layer)  
    encryption, 21  
Stallman, Richard, 9  
standby mode, 160  
`start` command, 129  
`start` script, 17  
starting  
    Heartbeat, 144–145  
    services, 15–21  
`statd` daemon, 283–284  
stateful failover  
    for high-availability cluster,  
        277–278  
    of IPVS table, 275–276  
`status` command, 129  
status of Heartbeat services  
    determining, 125–126, 129  
    messages for, 114  
stock kernels, 54–55  
Stonith, 164  
Stonith (shoot the other node in  
    the head), 113–114,  
        163–164  
background, 164–165  
configuration of, 166  
device support in, 168–174  
events in  
    with Mon, 323–324  
    multiple, 174  
    sequences of, 167–168  
    single devices in, 165–166  
`stonith_host` entry, 170  
`stop` command, 129  
stopping Heartbeat, 144–145  
`strace` command, 338  
Super Sparrow program, 157  
superfluous Apache messages, 260  
symbolic links  
    `chkconfig` for, 18–20  
    `ntsysv` for, 20–21  
    for source code, 56  
synchronizing servers, 69–70  
`syslog` script, 29, 137, 316  
system configuration changes for  
    clusters, 187  
System Configurator, 109  
system hangs, watchdog for,  
    176–179  
`System.map` file, 61  
system reboots, forcing, 178  
system time, 140  
System V lockf kernel lock  
    arbitration, 281, 285

SystemImager package, 95–96  
boot floppies for, 103–107  
client software for, 98–99  
client updates in, 108–109  
DHCP servers with, 101–103  
for Golden Client cloning,  
    95–96  
installer program in, 98  
post-installation notes, 107–108  
rsync in, 106  
server software for, 97–98  
system images on, 99–101,  
    106–107  
SystemInstaller project, 108–109

## T

tail command, 60  
TAL program, 367  
tar (tape archive) files, 373–374  
.tar.bz file, 372  
.tar.bz2 file, 372  
.tar.gz file, 372  
.tar.Z file, 372  
.tbz file, 372  
.tbz2 file, 372  
tcp option, 296  
tcp\_states table, 244  
tcpdump utility, 143, 375–377  
telnet  
    for cluster node access, 353  
    for iptables and ipchains, 42  
    persistence for, 249  
    PuTTY for, 367  
    for user sessions, 203  
    xinetd for, 16, 29  
telnet.monitor script, 323  
telnetd daemon, 360–361  
templates, persistent connection,  
    247–248  
testing  
    email, 354  
    Heartbeat configuration,  
        179–181  
    ldirectord configuration,  
        273–274  
    ldirectord installation, 266–267

LVS-NAT cluster configuration,  
    224–227  
Mon package installation,  
    308–309  
NICs, 383–385  
for resource ownership,  
    126–128  
resource scripts, 128  
two-node SSH client-server  
    connections, 82–86  
Thin Clients, 362–363  
time command, 293  
time in Heartbeat configuration,  
    140  
time to live (TTL) setting, 156  
timeouts  
    for connection tracking  
        records, 243–244  
        persistence, 247–248  
timers, watchdog, 178  
title section for Ganglia Web  
    package, 333–334  
top command, 338  
total I/O operations, measuring,  
    294–295  
touch command, 218  
tracking connection records  
    for Director, 242–243  
    timeout values for, 243–244  
transactions in NFS performance,  
    289–290  
traps, SNMP, 28  
troubleshooting  
    NICs, 384–385  
    tcpdump for, 375–377  
trust relationships in SSH, 73–77  
TTL (time to live) setting, 156  
tunneling, 200–202  
tux script, 29  
two-node SSH client-servers, 77–78  
    connection tests in, 82–86  
    copying files in, 89  
    firewall rules in, 92–93  
    host key configuration in, 79  
    rsync over slow WAN  
        connections, 89–90  
    rsync over SSH in, 87–89

scheduled snapshots in, 90–92  
secure data transport in, 86  
security improvements in,  
86–87  
user accounts in, 78–79  
user encryption keys for, 79–82  
two-way trust relationships, 73–77

## U

ucd-snmp package, 310  
ucd-snmp-utils package, 310  
udpport entry, 135  
uname command, 55, 59, 137  
uncompress command, 372  
Unison project, 70  
Unix  
    account administration  
        methods, 342–343  
    converting applications from,  
        192  
    updateclient command, 108, 187  
    updates  
        cluster node software, 189  
        SystemImager client, 108–109  
    upgrading kernel, 56–57  
    uptime utility, 335  
    URLs, self-referential, 403–404  
    UseCanonicalName directive, 404  
    user account administration  
        without Active Directory,  
            342–343  
        for clusters, 189–190  
        in SSH, 78–79  
    user applications, interactive,  
        292–293  
    user authentication  
        for cluster accounts, 343–345  
        in SSH, 76–77  
    user encryption keys in SSH, 72  
        copying, 81–82  
        creating, 79–80  
    user perceptions of NFS  
        performance, 288–290  
    useradd command, 78  
    /usr/lib/heartbeat/hb\_standby file,  
        160

/usr/lib/heartbeat/ResourceManager  
    file, 140  
/usr/lib/heartbeat/send\_arp file, 122  
/usr/lib/mon/logs file, 309  
/usr/sbin/checkpc file, 347  
/usr/sbin/getimage file, 99  
/usr/sbin/httpd file, 405  
/usr/sbin/prepareclient file, 99  
/usr/sbin/stonith file, 173  
/usr/share/doc/packages/heartbeat  
    file, 134  
/usr/src/linux file, 56  
/usr/src/linux/arch/i386/boot/bzImage  
    file, 60  
/usr/src/linux/.config file, 56  
/usr/src/linux/System.map file, 61

## V

V9fs filesystem, 396  
/var/lib/heartbeat/rsctmp file, 126  
/var/lib/systemimager file, 99  
/var/lib/systemimager/scripts file,  
    105  
/var/log/ha-log file, 141, 161  
/var/log/ldirectord.log file, 269  
/var/log/messages file, 161  
/var/log/secure file, 86  
/var/log/systemimager/rsyncd file,  
    106–107  
/var/spool/cron file, 357  
verifying Apache configuration,  
    404–405  
vers option, 296  
versions, kernel, 55–56  
virtual entry, 269  
virtual host configuration, 400–405  
virtual IP (VIP) addresses  
    with load balancer, 241  
    in LVS, 194–195  
    on LVS-NAT real servers,  
        214–216  
virtual services with load balancer,  
    241  
vs\_tcp\_states\_dos table, 244  
vs\_timeout\_table\_dos tables, 244

## W

w utility, 335  
WAN connections, rsync over,  
    89–90  
warntime value, 161  
watch command, 273, 291  
watchdog, 176–177  
    configuration for, 178–179  
    enabling, 177  
    rebooting with, 178  
Web package, Ganglia, 333–337  
weight of cluster nodes, 352  
weighted least-connection (WLC)  
    scheduling method, 204  
weighted round-robin (WRR)  
    scheduling method, 202  
wget utility, 372–373  
whoami command, 80  
wide-area load balancing, 157  
winbindd script, 29  
wireless infrared communication,  
    24  
WLC (weighted least-connection)  
    scheduling method, 204  
writes in fcntl locking, 282  
WRR (weighted round-robin)  
    scheduling method, 202  
wti\_nps device, 168

## Z

.z file, 372  
ZEO program, 367  
zero port connections, 248  
Zope, 342, 351, 367

## X

xfs script, 29  
xinetd script, 29–30  
XML-Simple module, 97

## Y

Yellow dog Updater, Modified  
    (Yum) program, 392–393  
ypbind script, 30–31  
yplapd daemon, 30  
yppasswd script, 31  
ypserv script, 31  
Yum (Yellow dog Updater,  
    Modified) program,  
    392–393



# Electronic Frontier Foundation

## Defending Freedom in the Digital World

**Free Speech. Privacy. Innovation. Fair Use. Reverse Engineering.** If you care about these rights in the digital world, then you should join the Electronic Frontier Foundation (EFF). EFF was founded in 1990 to protect the rights of users and developers of technology. EFF is the first to identify threats to basic rights online and to advocate on behalf of free expression in the digital age.

**The Electronic Frontier Foundation Defends Your Rights!**  
**Become a Member Today!**  
**<http://www.eff.org/support/>**

### Current EFF projects include:

*Protecting your fundamental right to vote.* Widely publicized security flaws in computerized voting machines show that, though filled with potential, this technology is far from perfect. EFF is defending the open discussion of e-voting problems and is coordinating a national litigation strategy addressing issues arising from use of poorly developed and tested computerized voting machines.

*Ensuring that you are not traceable through your things.* Libraries, schools, the government and private sector businesses are adopting radio frequency identification tags, or RFIDs – a technology capable of pinpointing the physical location of whatever item the tags are embedded in. While this may seem like a convenient way to track items, it's also a convenient way to do something less benign: track people and their activities through their belongings. EFF is working to ensure that embrace of this technology does not erode your right to privacy.

*Stopping the FBI from creating surveillance backdoors on the Internet.* EFF is part of a coalition opposing the FBI's expansion of the Communications Assistance for Law Enforcement Act (CALEA), which would require that the wiretap capabilities built into the phone system be extended to the Internet, forcing ISPs to build backdoors for law enforcement.

*Providing you with a means by which you can contact key decision-makers on cyber-liberties issues.* EFF maintains an action center that provides alerts on technology, civil liberties issues and pending legislation to more than 50,000 subscribers. EFF also generates a weekly online newsletter, EFFector, and a blog that provides up-to-the minute information and commentary.

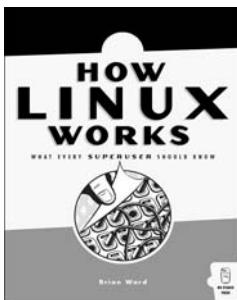
*Defending your right to listen to and copy digital music and movies.* The entertainment industry has been overzealous in trying to protect its copyrights, often decimating fair use rights in the process. EFF is standing up to the movie and music industries on several fronts.

**Check out all of the things we're working on at <http://www.eff.org> and join today or make a donation to support the fight to defend freedom online.**

More No-Nonsense Books from



**NO STARCH PRESS**



## **HOW LINUX WORKS**

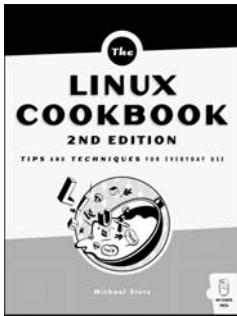
### **What Every Superuser Should Know**

by BRIAN WARD

*How Linux Works* describes the inside of the Linux system for systems administrators, whether they maintain an extensive network in the office or one Linux box at home. While some books try to give copy-and-paste instructions for how to deal with every single system program out there, *How Linux Works* shows the reader how the Linux system functions so that they will have the knowledge to solve any problem that comes up.

MARCH 2004, 368 PP., \$37.95 (\$55.95 CAN)

ISBN 1-59327-035-6



## **THE LINUX COOKBOOK 2ND EDITION**

### **Tips and Techniques for Everyday Use**

by MICHAEL STUTZ

*The Linux Cookbook 2nd Edition* shows you the best ways to do things on Linux, so that you can get your work done, quickly and easily. Organized by the general things that you use your computer for, the book gives you "recipes" for each task, each one with simple, step-by-step instructions, and an example that shows how to use the technique in practice. Covers the major Linux distributions.

APRIL 2004, 576 PP., \$39.95 (\$59.95 CAN)

ISBN 1-59327-031-3



## **WICKED COOL SHELL SCRIPTS**

### **101 Scripts for Linux, Mac OS X and Unix Systems**

by DAVE TAYLOR

The UNIX shell is the main scripting environment of every Linux, Mac OS X and UNIX system. This cookbook of useful and fun scripts gives you the tools to solve many common problems and personalize your computing environment. Among the more than 100 scripts included are an interactive calculator, a spell checker, and a disk backup utility.

JANUARY 2004, 488 PP., \$29.95 (\$44.95 CAN)

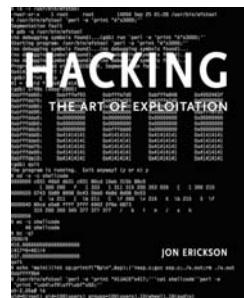
ISBN 1-59327-012-7

# HACKING

## The Art of Exploitation

by JON ERICKSON

A comprehensive introduction to the techniques of exploitation and creative problem-solving methods commonly referred to as “hacking.” Shows how hackers exploit programs and write exploits, instead of just how to run other people’s exploits. Explains the technical aspects of hacking, including stack-based overflows, heap-based overflows, string exploits, return-into-libc, shellcode, and cryptographic attacks on 802.11b.



“From all the books I’ve read so far, I would consider this the seminal hacker’s handbook.”—Security Forums.com

NOVEMBER 2003, 264 PP., \$39.95 (\$59.95 CAN)

ISBN 1-59327-007-0

# THE BOOK OF™ POSTFIX

## State-of-the-Art Message Transport

by RALF HILDEBRANDT and PATRICK KOETTER

Developed with security and speed in mind, Postfix has become a popular alternative to Sendmail. *The Book of Postfix* is a complete guide to Postfix whether used by the home user or as a company mailserver. Practical examples show how to deal with daily challenges like protecting mail users from SPAM and viruses, managing multiple domains, and offering roaming access.



MARCH 2004, 328 PP., \$39.95 (\$59.95 CAN)

ISBN 1-59327-001-1

**PHONE:**  
800.420.7240 OR  
415.863.9900  
MONDAY THROUGH FRIDAY,  
9 A.M. TO 5 P.M. (PST)

**FAX:**  
415.863.9950  
24 HOURS A DAY,  
7 DAYS A WEEK

**EMAIL:**  
SALES@NOSTARCH.COM

**WEB:**  
[HTTP://WWW.NOSTARCH.COM](http://WWW.NOSTARCH.COM)

**MAIL:**  
NO STARCH PRESS  
555 DE HARO ST, SUITE 250  
SAN FRANCISCO, CA 94107

No Starch Press Copyright © 2005 by Karl Kopper  
USA

## **UPDATES**

Visit [\*\*http://www.nostarch.com/cluster.htm\*\*](http://www.nostarch.com/cluster.htm) for updates, errata,  
and other information.

## **A B O U T T H E C D - R O M**

The CD-ROM includes copies of the stock Linux 2.4 and 2.6 kernels with the LVS kernel modules; the ldirectord software and all of its dependencies; the Mon monitoring package, monitoring scripts, and dependencies; the Ganglia package; OpenSSH; rsync; SystemImager; and Heartbeat. The latest versions of all these software packages can be downloaded from the Internet for free.

The CD-ROM also includes copies of all the figures and illustrations used in the book so you can use and modify them to suit your needs.

# CD LICENSE AGREEMENT

This CD-ROM bundled with the book *The Linux Enterprise Cluster*, ISBN 1-59327-036-4, contains electronic versions of all figures and illustrations (“Figures”) that appear in the book, as well as a collection of relevant software. See the specific license for each software package located in the source files on the CD. All software is published under the GNU General Public License (GPL), <http://www.gnu.org/copyleft/fdl.html>, except Ganglia, which is released under the BSD license.

The Figures on the *Linux Enterprise Cluster* CD-ROM are released under the Creative Commons Attribution-NonCommercial-ShareAlike License (<http://creativecommons.org/licenses/by-nc-sa/2.0>). This license applies *only* to the Figures on the CD-ROM.

You are free to copy, distribute, display, and perform the Figures contained on the CD-ROM and to make derivative works from them but you must give the original author credit, you may not use this work for commercial purposes, and if you alter, transform, or build upon the Figures, you may distribute the resulting work only under a license identical to this one.

The penguin art is owned by Karl Kopper and released under the Creative Commons license.

For any reuse or distribution, you must make clear to others the license terms of this work. Any of these conditions can be waived if you get permission from the copyright holder.

## No Warranty

The software provided is free of charge and without warranty. The software is provided “as is” without warranty of any kind, either expressed or implied. The software is provided without warranty for merchantability or fitness for any purpose. The entire risk as to the quality and performance of the software is with the user. If any software provided proves to be defective or insecure or to cause damages of any kind, the user assumes all cost of repair.