Coda File System User and System Administrators Manual

M. Satyanarayanan

satya@cs.cmu.edu

Maria R. Ebling

mre@cs.cmu.edu

Joshua Raiff

jraiff@cs.cmu.edu

Peter J. Braam

braam@cs.cmu.edu

Jan Harkes

jaharkes@cs.cmu.edu

Coda File System User and System Administrators Manual

by M. Satyanarayanan, Maria R. Ebling, Joshua Raiff, Peter J. Braam, and Jan Harkes

Published August 1997

Copyright © 1995, 1996, 1997, 1998, 1999, 2000 by Coda Team

The *Coda File System* is a descendant of the *Andrew File System.* Like AFS, Coda offers location-transparent access to a shared Unix file namespace that is mapped on to a collection of dedicated file servers. But Coda represents a substantial improvement over AFS because it offers considerably higher availability in the face of server and network failures. The improvement in availability is achieved using the complementary techniques of *server replication* and *disconnected operation.* Disconnected operation proven especially valuable in supporting portable computers. This document is a reference manual for Coda users and system administrators.

Table of Contents

Preface **8**

Further Reading 8

Acknowledgments 9

1. Getting Started **10**

What is Coda? 10

Getting clued in with the Coda terminology. 10

Organization of the client 12

The kernel module and the cache manager 13

Utilities 13

Server organization 13

Authentication 13

Coda File Protection 14

Disconnected Operation 16

Hoarding 17

Repairing Conflicts 17

2. Common Scenarios **19**

Constructing a hoardfile 19

Hoarding for a Weekend 20

Re-integrating After Disconnection 21

Dealing With a Flaky Network 22

Re-integrating Over the Phone Line 22

Repairing an Inconsistent Directory 22

Server/Server Conflicts 23

Local/Global Conflicts 25

3. System Overview **27**

Machines 27

Processes 28

Data Location 29

4. Obtaining Coda **30**

5. Configuring Kernels for use with Coda clients **32**

VFS Interface 32

Configuring a Linux kernel 32

Configuring a FreeBSD kernel 32

Configuring a NetBSD kernel 33

6. Installing and Configuring a Coda Client **34**

Installing and Configuring the Coda Client Binaries and Documentation 34

Linux and the BSD's: running venus, the client cache manager 35
Windows 95: Starting and Configuring a Coda client 36
Upgrading from a previous Release 37
Configuration changes made by venus-setup 37
7. Installing a Coda Server 40
Introduction 40
Recoverable Virtual Memory 40
Server Disk Organization 41
Installation and Configuration 42
Installing the Coda Server Binaries and Documentation 42
Configuring A Coda Server 43
Before Beginning 43
Setting Up A Coda Server 44
Running Coda Server 48
Startup 48
Shutdown 51
Exploring replication 51
Underneath vice-setup 52
RVM Initialization 52
The Log Partition 53
The Metadata Partition 53
Update Monitor 54
Authentication Database 55
8. Troubleshooting 56
Basic Troubleshooting 56
Client Problems 57
Server Problems 59
Disconnections. 62
Advanced Troubleshooting 64
Troubleshooting on Windows 95 65
Common problems 66
Restrictions 66
9. Volume Administration 68
Concepts 68
Creating a Volume 68
Mounting a Volume 69
Deleting a Volume 70
Dumping and Restoring a Volume 70
Creating a dump of a replicated volume 70

Restoring volume dumps 71
Building the volume databases 71
Getting Volume Information 71
10. User Administration 73
Short introduction to pdbtool 73
Adding a new user 74
Upgrading existing user.coda and group.coda databases 75
Upgrading from the coda.pdb/name.pdb databases used in 5.2.0/5.2.2 75
11. The Backup System 77
Introduction: Design of the Coda Backup Subsystem 77
Installing a Coda Backup Coordinator Machine 78
Incremental Dumps 81
Tape files 81
Restoring a backup clone 82
Backup Scripts 83
12. Reinitializing Coda Servers after a Disaster 84
Obtaining Backup Dump Files 84
Reinitializing the Servers 85
References 87
A. Quick Reference 88
B. System Configuration Files 89

List of Tables

3-1. Server processes 28
7-1. Example of Partitions for a Coda Server 41
9-1. Coda volume types 68
A-1. Popular commands 88

List of Figures

1-1. Coda file-system view at a workstation 10
3-1. Typical Coda server organization. 27
3-2. Typical processes on file-server 28

Preface

Welcome to Coda! Whether you are at Carnegie Mellon or elsewhere, we hope that this manual will help you make effective use of the Coda File System. The manual is written from two different perspectives: end user and system administrator. If your organization is already running Coda, and you need to learn how to use it - Chapter 1 and Chapter 2 should fully answer your questions. For more adventurous among you, who wan't to maintain your own Coda cell, please read Chapter 3 through Chapter 6. The rest of the book is dedicated specifics of system administration.

In writing this manual, we have assumed that you are an experienced Unix user, familiar with the broad concepts of distributed file systems. If you are a system administrator, we assume that you are familiar with administering Unix systems. You will find Coda especially easy to use if you are already fluent in using AFS.

We have also assumed that you are familiar with the design goals and architecture of Coda. The best way to learn about these is to read the overview and design rationale papers on Coda [SATYANARAYANAN90Z], [SATYANARAYANAN90Y], [KISTLER92]. We urge you to obtain copies of these papers and to read them before attempting to use Coda. Those papers provide context and information upon which this manual relies and does not repeat.

Since Coda is an experimental system and not a commercial product, you will inevitably encounter rough edges. Within the limits of our resources, we will do our best to fix these problems and to improve the system over time. But we do ask that you try to characterize the problem as accurately as possible, and to try and obtain a repeatable and concise instance of it.

After you have taken the time to learn the system well, please give us your feedback. We would like to improve both the system and this manual.

Further Reading

Besides the three papers mentioned above, there are a number of papers that address specific aspects of Coda. These include:

- A description of the MiniCache, which allows the client manager to reside outside the kernel without excessive loss of performance [STEERE90].
- A description and performance analysis of the technique used in Coda for transparent directory resolution [KUMAR93].
- A manual on the remote procedure call, RPC2, and threading, LWP, mechanisms used in Coda [SATYA91].

- A paper and manual on the transactional facility, RVM, used on Coda clients and servers [SATYANARAYANAN93], [MASHBURN92].
- A detailed description of support for disconnected operation in Coda [KISTLER93]. Chapter 4 of this document offers an excellent overview of the implementation structure of clients and servers.
- A discussion of relevant security issues and mechanisms to address them [SATYANARAYANAN89C]. This paper discusses security in AFS-2, but the mechanisms in Coda are virtually identical.

Acknowledgments

Coda is the work of many individuals. Contributors to the design and implementation of various aspects of the system include: Jay Kistler, Puneet Kumar, David Steere, Lily Mummert, Maria Ebling, M. Satyanarayanan, Hank Mashburn, Brian Noble, Lu Qi, Josh Raiff, Ellen Siegel, Anders Klemets, and Kudo Masahi. Many of these individuals have also contributed to the writing of this manual. The system has improved considerably in response to feedback from users outside the Coda project. The earliest of these users (Tom Mitchell, Manuela Veloso, and Matt Zekauskas) deserve special thanks for their willingness to sail into uncharted waters!

Chapter 1. Getting Started

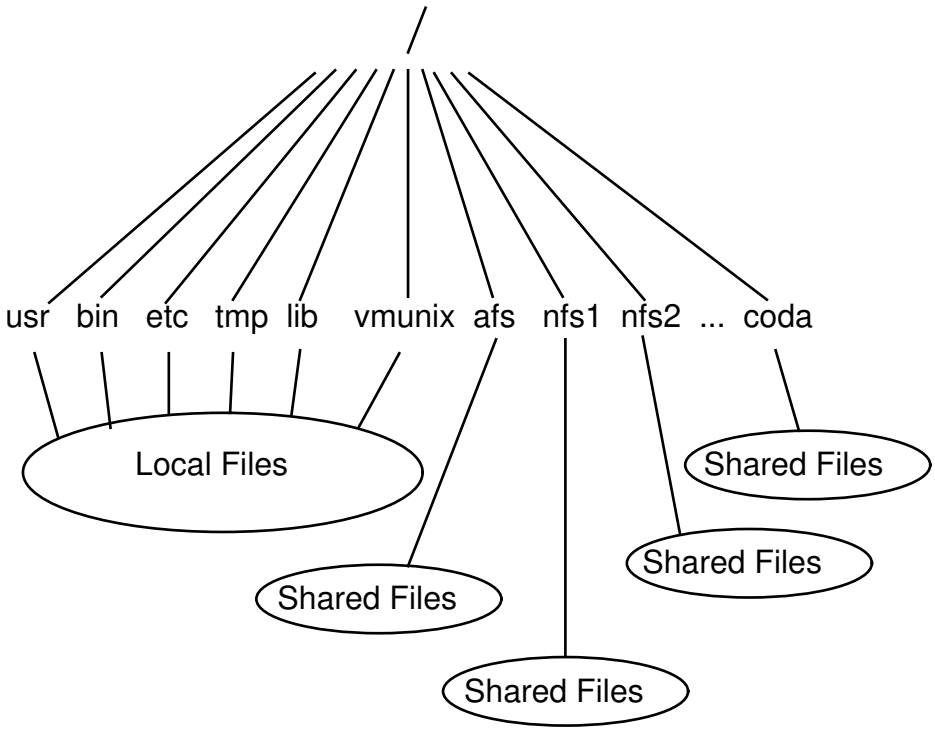
What is Coda?

Coda is a distributed file system, i.e. it makes files available to a collection of client computers as part of their directory tree, but ultimately maintains the authoritative copy of the file data on servers. Coda has some features that make it stand out: it supports *disconnected operation*, i.e. full access to a cached section of the file space during voluntary or involuntary network or server outages. Coda will automatically reintegrate the changes made on disconnected clients when reconnecting. Furthermore, Coda has read-write, failover server replication, meaning that data is stored and fetch from any of a group of servers and Coda will continue to operate when only a subset of all servers is available. If server differences arise due to *network partitions* Coda will resolve differences automatically to a maximum extent possible and aid users in repairing what can't be done automatically. Coda is very differently organized from NFS and Windows/Samba shares. Coda does have many similarities to AFS and DCE/DFS.

Getting clued in with the Coda terminology.

A single name space. All of Coda appears under a single directory `/coda` on the client (or under a single drive under Windows). Coda does not have different exports or shares as do NFS and Samba that are individually mounted. Under `/coda` the volumes (aka file sets) of files exported by all the servers (living in your Coda cell) are visible (See Figure 1-1). Coda automatically finds servers and all a client needs to know is the name of one bootstrap server that gives it information how to find the root volume of Coda.

Figure 1-1. Coda file-system view at a workstation.



Coda cell. Cell is a group of servers sharing one set of configuration databases. A cell can consist of a single server or up to hundreds of servers. One server is designated as the SCM - the System Control Machine. It is distinguished by being the only server modifying the configuration databases shared by all servers, and propagating such changes to other servers. At present a Coda client can belong to a single cell. We hope to get a cell mechanism into Coda whereby a client can see files in multiple cells.

Coda volumes. File servers group the files in volumes. A volume is typically much smaller than a partition and much larger than a directory. Volumes have a root and contain a directory tree with files. Each volume is "Coda mounted" somewhere under `/coda` and forms a subtree of the `/coda`. Volumes can contain mountpoints of other volumes. A volume mountpoint is not a Unix mountpoint or Windows drive - there is only one drive or Unix mountpoint for Coda. A Coda mountpoint contains enough information for the client to find the server(s) which store the files in the volume. The group of servers serving a volume is called the *Volume Storage Group* of the volume.

Volume Mountpoints. One volume is special, it is the root volume, the volume which Coda mounts on

`/coda`. Other volumes are grafted into the `/coda` tree using **cfs mkmount**. This command installs a volume mountpoint in the Coda directory tree, and in effect its result is similar to **mkdir mountpoint; mount device mountpoint** under Unix. When invoking the **cfs makemount** the two arguments given are the name of the mountpoint and the name of the volume to be mounted. Coda mountpoints are persistent objects, unlike Unix mountpoints which needs reinstating after a reboot.

Data storage. The servers do not store and export volumes as directories in the local disk filesystem, like NFS and Samba. Coda needs much more meta data to support server replication and disconnected operation and it has complex recovery which is hard to do within a local disk filesystem. Coda servers store files identified by a number typically all in a directory tree in `/vicepa`. The meta data (owners, access control lists, version vectors) is stored in an RVM data file which would often be a raw disk partition.

RVM. RVM stands for *Recoverable Virtual Memory*. RVM is a transaction based library to make part of a virtual address space of a process persistent on disk and commit changes to this memory atomically to persistent storage. Coda uses RVM to manage its metadata. This data is stored in an RVM data file which is mapped into memory upon startup. Modifications are made in VM and also written to the RVM LOG file upon committing a transaction. The LOG file contains committed data that has not yet been incorporated into the data file on disk.

Client data. Data on the client is stored somewhat similarly: meta data in RVM (typically in `/usr/coda/DATA`) and cached files are stored by number under `/usr/coda/venus.cache`. The cache on a client is persistent. This cache contains copies of files on the server. The cache allows for quicker access to data for the client and allows for access to files when the client is not connected to the server.

Validation. When Coda detects that a server is reachable again it will *validate* cached data before using it to make sure the cached data is the latest version of the file. Coda compares cached version stamps associated with each object, with version stamps held by the server.

Authentication. Coda manages authentication and authorization through a token. Similar (the details are very different) to using a Windows share, Coda requires users to log in. During the log in process, the client acquires a session key, or token in exchange for a correct password. The token is associated with a user identity, at present this Coda identity is the uid of the user performing the log in.

Protection. To grant permissions the cache manager and servers use the token with its associated identity and match this against priviliges granted to this identity in access control lists (ACL). If a token is not present, anonymous access is assumed, for which permissions are again granted through the access control lists using the `System:AnyUser` identity.

Organization of the client

The kernel module and the cache manager

Like every filesystem a computer enabled to use the Coda filesystem needs kernel support to access Coda files. Coda's kernel support is minimal and works in conjunction with the userspace cache manager Venus. User requests enter the kernel, which will either reply directly or ask the cache manager venus to assist in service.

Typically the kernel code is in a kernel module, which is either loaded at boot time or dynamically loaded when Venus is started. Venus will even mount the Coda filesystem on `/coda`.

Utilities

To manipulate ACL's, the cache, volume mountpoints and possibly the network behaviour of a Coda client a variety of small utilities is provided. The most important one is **cfs**.

There is also a **clog** to authenticate to the Coda authentication server. The **codacon** allows one to monitor the operatoin of the cache manager, and **cmon** gives summary information about a list of servers.

Server organization

The main program is the Coda fileserver codasrv. It is responsible for doing all file operations, as well as volume location service.

The Coda authentication server auth2 handles requests from clog for tokens, and changes of password from au and cpasswd. Only the the auth2 process on the SCM will modify the password database.

All servers in a Coda cell share the configuration databases in `/vice/db` and retrieve them from the SCM when changes have occurred. The updateclnt program is responsible for retrieving such changes, and it polls the updatesrv on the SCM to see if anything has changed. Sometimes the SCM needs a (non-shared) database from another server to update a shared database. It fetches this through an updatesrv process on that server using updatefetch.

On the server there are utilities for volume creation and management. These utilities consist of shell scripts and the volutil. There is also a tool to manipulate the protection databases.

Authentication

Once you are logged in to your workstation, you need to get a Coda authentication token by running the clog. Clog will prompt you for your Coda password and use it to get a token from the *authentication server*. This token will expire in about 25 hours. After the token expires, you must use clog to authenticate for another 25 hours.

The following in an example of running clog twice. The first time, the wrong password was entered:

```
% clog
Password:
Invalid login (RPC2_NOTAUTHENTICATED (F))
% clog
Password:
%
```

To see your newly acquired token, use ctokens. This will display the tokens and their expiration time for your UID.

```
% ctokens
Token held by the Cache Manager:

Local uid: 9010
Coda user id: 9010
Expiration time: Thu Apr  6 18:51:35 2000
```

Use the **cpasswd** to change your Coda password. As with passwd, cpasswd will prompt for your current password, then ask you to enter a new password twice.

```
% cpasswd
Changing password for raiff
Old password:
New password for raiff:
Retype new password:
Password changed, it will be in effect in about 1 hour
```

You can "log out" of Coda by using the **cunlog** to tell *venus* to forget your tokens. Once you run cunlog, you will have the same privileges as an anonymous Coda user until you acquire a new authentication token.

Coda File Protection

Coda provides a close approximation to UNIX protection semantics. An *access control list* (ACL) controls access to directories by granting and restricting the rights of users or groups of users. An entry in an access list maps a member of the protection domain into a set of rights. Userrights are determined by the rights of all of the groups that he or she is either a direct or indirect member. In addition to the Coda access lists, the three owner bits of the file mode are used to indicate readability, writability, and executability. You should use chmod(1) to set the permissions on individual files. Coda rights are given as a combination of `rlidwka` where:

- `r` - *Read*. Allows the user to read any file in the directory.
- `l` - *Lookup*. Lookup allows the user to obtain status information about the files in the directory. An example is to list the directory contents.
- `i` - *Insert*. Allows the user to create new files or subdirectories in the directory.
- `d` - *Delete*. Allows the user to remove files or subdirectories.
- `w` - *Write*. Allows the user to overwrite existing files in the directory.
- `k` - *Lock*. The lock right is obsolete and only maintained for historical reasons.
- `a` - *Administer*. Allows the user to change the directory's access control list.

Coda also has *negative* rights, which deny access. Any of the normal rights listed above can also be negative.

Access control lists are managed with the **cfs** with the `listacl` and `setacl` options. They can be abbreviated as `la` and `sa` respectively. To see the access control list of any directory in a Coda file system, use **cfs la**. The following example displays the current directory's ACL:

```
% cfs la .
      System:AnyUser  rl
                raiff  rlidwka
```

The displayed list, shows that the user "raiff" has all of the access rights possible on the directory and that the group `System:AnyUser` has read and lookup privileges. `System:AnyUser` is a special Coda group, that includes all users.

A second example shows another group, `System:Coda`. Anyone who is a member of the group, will have the groups access rights:

```
% cfs la /coda
        System:coda  rlidwka
```

```
System:AnyUser  rl
```

Use **cfs sa** to change or set a directory's access control list. Options to **cfs sa** include `-negative` to assign negative rights to a user and `-clear` to clear the access list completely before setting any new access rights. You can also use `all` or `none` to specify all rights or no rights respectively.

To remove `System:AnyUsers` access to the current directory, you would issue the following command:

```
% cfs sa . System:AnyUser none
```

To give System:AnyUser read and lookup rights, use:

```
% cfs sa . System:AnyUser rl
```

To deny rights to a user, use the `-negative` switch:

```
% cfs sa -negative . baduser rl
```

This will deny `baduser` read and lookup rights, even though any other user has these rights. Note that negative rights are maintained separately from the normal rights, so to re-instate `badusers`' read and lookup access, you must use:

```
% cfs sa -negative . baduser none
```

If you omit the `-negative` switch, then `baduser` will still be denied read and lookup access.

Disconnected Operation

If all of the servers that an object resides on become inaccessible, then the client will use the cached copy of the object (if present) as a valid replica. When the client does this, it is operating in *disconnected mode*.

Disconnected mode may be the result of a network failure, or it could be the result of removing a laptop from the network. If you make sure all of the files you want to use are cached on your laptop, you can travel with it and access your files as if you were still on the network.

Unfortunately, a cache miss while operating in disconnected mode is not maskable, and you will get a connection timed out error message. Coda allows you to mark or *hoard* files with caching priorities to help keep the ones you want in the cache.

When you are in disconnected mode, you may want to checkpoint the modify log that Coda keeps of which directories have changed. Use **cfs checkpointml** to do this.

Checkpointing the modify log will ensure that changes you have made will not be lost if the cache manager crashes severely. The checkpointed log files are an identical copy of the in-memory logs that Coda uses when it re-integrates with the servers.

Coda adapts easily to low bandwidth connections like (PPP or SLIP modem links). You can use this to periodically reintegrate and cache new files when you are on a trip.

When you reintegrate after operating in disconnected mode, keep an eye on your codacon output or run the command:

```
% tail -f /usr/coda/etc/console
```

This file will let you know if the reintegration was successful. If it was not, then the files that you modified will be put in a tar file in `/usr/coda/spool/uid`. Reintegration fails, for example, when you modified a file in disconnected mode and someone else also modified that file on the servers. Read the section called *Re-integrating After Disconnection* in Chapter 2 for more information on reintegration.

Hoarding

Coda allows you to advise the cache manager, venus, of critical files that it should try to keep in the cache. You indicate the relative importance of the files by assigning priorities to them. This is known as *hoarding*. Venus maintains an internal *hoard database* of these files. Hoarding a file helps to ensure that it will be available when operating in disconnected mode. See the hoard(1) for information on hoard files and the section called *Constructing a hoardfile* in Chapter 2 and the section called *Hoarding for a Weekend* in Chapter 2 of this document for an example of how to set up your hoard database. A convenient way of setting up your hoard database is by creating a file with commands for the hoard. This file is known as a *hoard file*.

Repairing Conflicts

As a result of Coda's optimistic replica management, object replicas can conflict on different servers. A conflict arises when the same object is updated in different partitions of a network. For instance, suppose a file is replicated at two sites (say, serverA and serverB). If these two sites become partitioned and a user on each side of the partition updates the file (userA updates the file on serverA while userB updates the file on serverB), the file will be in conflict when the partition ends. Conflicts may also arise at the end of disconnected operation.

Coda guarantees conflict detection at the first request for that object when both servers are accessible. When a conflict is detected, Coda attempts to perform automatic conflict resolution. In simple cases, the conflict will be resolved automatically, a process which is transparent to the user except for a time delay in accessing the object. However, in more difficult cases, automatic conflict resolution fails and the object is marked *in conflict*. File system calls on an object which is in conflict fail with the same error code as if the object were a dangling, read-only symbolic link (usually, ENOENT). The conflict must be resolved by a user with appropriate access to the object. To help users resolve conflicts, Coda provides a repair tool which is discussed in the section called *Repairing an Inconsistent Directory* in Chapter 2.

Chapter 2. Common Scenarios

There are several common scenarios that you may encounter. This chapter attempts to list as many of them as possible and suggest how to handle each scenario.

Constructing a hoardfile

Coda allows you to give files priorities that affect the cache manager. The higher the priority, the lower the possibility that a file will get flushed from the cache to make space for a another file. These priorities are stored in a hoard database internal to Venus. This database is preserved across invocations of Venus, but will be erased when Venus is re-initialized.

The best way to set up a hoard database is by creating hoard files. Afteryou've created the files once, you do not need to do it again for that set of files. You can create a hoard file by hand or by using the spy program. See the hoard(1) man page for more details.

To run spy:

1. Run spy in the background, redirecting its output to a file.
2. Run all of the programs and access files you want to hoard.
3. Send a SIGTERM to spy (Do not use `^C`)
4. Sort the output file, removing duplicates
5. Remove unnecessary entries
6. On each line add `a` at the beginning and a priority at the end

The following is an example of creating a hoard file for gnu-emacs. Note that while running gnu-emacs, I explicitly enter *scribe mode*. This makes sure that the scribe-specific files are fetched.

```
% spy > gemacs.out &
[1] 316
%  gnu-emacs
%  kill %1
%
[1]    Done                  spy > gemacs.out
% sort -u gemacs.out > gemacs.hdb
% cat gemacs.hdb
/coda
/coda/i386_mach/omega/usr/local/emacs
/coda/i386_mach/omega/usr/misc/.gnu-emacs
```

```
/coda/misc/gnu-emacs/i386_mach/omega/bin/gnu-emacs
/coda/misc/gnu-emacs/i386_mach/omega/lisp/scribe.elc
/coda/misc/gnu-emacs/i386_mach/omega/lisp/term/x-win.el
/coda/misc/gnu-emacs/i386_mach/omega/lisp/x-mouse.elc
/coda/usr
```

Next you should delete the first and last line of the file as you do not need them. Then add the hoard specific commands. The final filelooks like:

```
a /coda/i386_mach/omega/usr/local/emacs 600
a /coda/i386_mach/omega/usr/misc/.gnu-emacs 600
a /coda/misc/gnu-emacs/i386_mach/omega/bin/gnu-emacs 600
a /coda/misc/gnu-emacs/i386_mach/omega/lisp/scribe.elc 600
a /coda/misc/gnu-emacs/i386_mach/omega/lisp/term/x-win.el 600
a /coda/misc/gnu-emacs/i386_mach/omega/lisp/x-mouse.elc 600
```

The `a` that starts each line tells hoard to **add** the named file to the database. The `600` that ends each line sets the file's priority. You may also specify additional attributes for each line. These attributes are separated from the priority by a `:` and are:

- `c` Current children of the given directory will inherit its hoard status.
- `c+` Current and future children of the given directory will inherit its hoard status.
- `d` Current descendents of the given directory will inherit its hoard status.
- `d+` Current and future descendents of the given directory will inherit its hoard status.

For example, to hoard all of the `emacs` directory, its descendents and any future descendents, I would include the following line in a hoardfile: `a /coda/i386_mach/omega/usr/local/emacs 600:d+`

This ensures you get all of the files you need, but you will use tens of megabytes of cache space to hoard many files that you do not need, so often you want to be more specific with respect to which files to hoard.

Other valid command to hoard are **clear**, **delete**, **list**, and **modify**. See the hoard(1) man page for more details on these commands.

Hoarding for a Weekend

One of the most common uses of a Coda laptop is to take it home overnight or for the weekend. Naturally, you want to be sure that all of the files that you need over the weekend are in the cache;

otherwise, there is little point in bringing the laptop home. The hoard(1) program helps you do this. Create a hoard file, as described in Section the section called *Constructing a hoardfile*, for each project you want to work on. You may also want hoard files for your personal files, such as your home directory if its in Coda, and other tools that you plan on using. It is best to clear the hoard database, whenever you switch projects to make sure you have enough space in your cache. You might consider having **clear** as the first command in your personal hoard file. If you do, make sure you always run hoard with this file before any other files. Once you've built hoard files for all of your tools and projects, its simply a matter of running hoard to build the hoard database you want. About fifteen minutes before you are ready to leave, force a hoard walk with the following command: **hoard** `walk`.

This will cause venus to attempt to cache all of the files in the hoard database. *Wait until the hoard command completes*. You are now ready to disconnect from the network. You are encouraged to try all of the commands you intend on using after you disconnect. If you are missing some files, it will be easy to reconnect and hoard them.

Re-integrating After Disconnection

When you reconnect to the network after a disconnected session, Coda will automatically try to reintegrate your changes with the Coda servers. You must be authenticated before reintegration occurs. Watch the file `/usr/coda/etc/console` by running: **tail** `-f /usr/coda/etc/console` or look at more detailed progress using the **codacon** command. The reintegration status will be written to this file.

If the reintegration was successful, the log entries would look like:

```
Reintegrate u.raiff, (1, 244) ( 13:33:43 )
coda: Committing CML for u.raiff ( 13:33:43 )
coda: Reintegrate: u.raiff, result = SUCCESS, elapsed = 2640.0 (15.0, 2609.0,
15.0) ( 13:33:43 )
coda:   delta_vm = 0, old stats = [0, 1, 0, 0, 0] ( 13:33:43 )
coda:   new stats = [   0,   0.0,     0.0,    1,   0.2], [   0,   0.0, 0.0,  0,   0.0] ( 13:
```

The following example is from a failed reintegration on the volume `u.raiff`.

```
Reintegrate u.raiff, (1, 244) ( 13:27:10 )
coda: Checkpoint-
ing u.raiff ( 13:27:10 )coda: to /usr/coda/spool/2534/u.raiff@@%
coda%usr%raiff ( 13:27:10 )
coda: Aborting CML for u.raiff ( 13:27:10 )
coda: Reintegrate: u.raiff, re-
sult = 198, elapsed = 2437.0 (15.0, 2265.0, 531.0)  ( 13:27:10 )
coda:   delta_vm = 1000, old stats = [0, 0, 1, 0, 0] ( 13:27:10 )
coda:   new stats = [   0,   0.0,     0.0,    1,   0.2], [   0,   0.0, 0.0,   0,   0.0] ( 13
```

Notice that the *change modify log* (CML) was check-pointed to `/usr/coda/spool/2534/u.raiff@@%coda%usr%raiff`. This file is a tar file containing the changes that were made on during the disconnected session. The files in the tar file are relative to the root of `u.raiff`.

The file which failed to reintegrate will be *in conflict* and will need to be repaired with the **repair** tool as discussed in the section called *Repairing an Inconsistent Directory*.

Dealing With a Flaky Network

When the network is acting up, you can use Coda to help isolate yourself from the networking problems. Set up your hoard database so that Venus will hoard the files you are working on. Then, disconnect from the Coda servers with the **cfs** `disconnect` command. To Coda, this is equivalent to physically disconnecting from the network.

Once the network becomes stable, you can use **cfs** `reconnect` followed by **cfs** `checkservers` to reconnect yourself to the Coda servers and re-integrate your work. Do not forget to clear your hoard database with **hoard** `clear` once you are done working on the set of files that you hoarded.

Re-integrating Over the Phone Line

If you are planning on taking a Coda laptop on an extended trip, you should consider using dial-up connection like SLIP or PPP to reintegrate with the Coda servers periodically. This will make your updates visible to other Coda users, protect against client crashes such as hard drive failure or theft, and allow you to work on multiple projects, even when your cache space is not large enough for all of the projects. By using the following instructions, you can reintegrate over the phone and change the files in your hoard database.

1. Dial to your network remote access server.
2. After network connection is established, tell Coda to see which servers it can communicate with **cfs** `checkservers` Your laptop will now behave as if it is on the network. Take into consideration that response time to commands will be much slower, than on the LAN.

Important: You can trigger immediate reintegration of a specific volume by issuing **cfs** `forcereintegrate` *`dir`* command.

Repairing an Inconsistent Directory

Occasionally, a directory entry will become inconsistent. This happens when there is a conflict between file server replicas that Coda cannot automatically resolve or a reintegration failed because of a local update the conflicts with the global state. The most common causes of a conflict are when the file servers are partitioned and a file is changed on more than one of the partitions or when a disconnected client updates a file that is also updated on the servers. When this happens, the directory containing the conflict will now look like a symbolic link and will be pointing to its *file identifier* (fid). For example, if a directory, `conflict`, is inconsistent, it will now appear as:

% ls -l conflict
lr–r–r– 1 root 27 Mar 23 14:52 conflict -> @@7f0000b3.00000005.0000011a

Most applications will return the error File not found when they try to open a file that is inconsistent. You need to resolve this conflict by using the repair(1) tool.

Server/Server Conflicts

Once you run repair, you need to do a **beginRepair** on the object that is inconsistent. After **beginRepair** is issued, the inconsistent directory will have an entry for each of the replicated volumes. You can look at all of these to decide which copy you want. Use repair to copy the correct version and clear the inconsistency. In the following example the file `conflict/example` is replicated on three servers. It has gone inconsistent.

```
% ls -lL conflict
lr-r-r-  1 root           27 Dec 20 13:12 conflict -
> @@7f0002ec.000000e3.000005d1
% repair
The repair tool can be used to manually repair files and directories
that have diverging replicas.  You will first need to do a "beginRepair"
which will expose the replicas of the inconsistent object as its children.


If you are repairing a directory, you will probably use the "com-
pareDir" and "doRepair" commands.

For inconsistent files you will only need to use the "doRepair" command.

If you want to REMOVE an inconsistent object, use the "removeInc" command.

Help on individual commands can also be obtained using the "help" facility.
* begin conflict
```

```
a server-server-conflict repair session started
use the following commands to repair the conflict:
comparedirs
removeinc
dorepair
* ^Z
Stopped
% ls conflict
gershwin.coda.cs.cmu.edu schumann.coda.cs.cmu.edu
% ls conflict/*
conflict/gershwin.coda.cs.cmu.edu:
example

conflict/schumann.coda.cs.cmu.edu:
example
% fg
repair
compare
Pathname of Object in conflict? [conflict]
Pathname of repair file produced?  [] /tmp/fix


NAME/NAME CONFLICT EXISTS FOR example

-rw-r-r-  1 raiff           0 Dec 20 13:10 gershwin.coda.cs.cmu.edu/example
-rw-r-r-  1 -101            0 Dec 20 13:11 schumann.coda.cs.cmu.edu/example


/coda/project/coda/demo/basic/rep/conflict/gershwin.coda.cs.cmu.edu/example
Fid: (0xb0.612) VV:(0 2 0 0 0 0 0 0)(0x8002f23e.30c6e9aa)
/coda/project/coda/demo/basic/rep/conflict/schumann.coda.cs.cmu.edu/example
Fid: (0x9e.5ea) VV:(2 0 0 0 0 0 0 0)(0x8002ce17.30d56fb9)
Should /coda/project/coda/demo/basic/rep/conflict/gershwin.coda.cs.cmu.edu/example be re-
moved?  [no] yes
Should /coda/project/coda/demo/basic/rep/conflict/schumann.coda.cs.cmu.edu/example be re-
moved?   [no]
Do you want to repair the name/name conflicts  [yes]
Operations to resolve conflicts are in /tmp/fix
* do
Pathname of object in conflict?  [conflict]
Pathname of fix file? [/tmp/fix]
OK to repair "conflict" by fixfile "/tmp/fix"?  [no] yes
SCHUMANN.CODA.CS.CMU.EDU  succeeded
GERSHWIN.CODA.CS.CMU.EDU  succeeded
* quit
```

```
% ls conflict
example
% exit
```

Local/Global Conflicts

Local/global conflicts are caused by reintegration failures, which means that the mutations performed while the client was disconnected are in conflict with the mutations performed on the servers from other clients during the disconnection. The objects involved in local/global conflict are represented in the same fashion as server/server conflicts, i.e., they become dangling symbolic links.

To start a local/global repair session for an object `OBJ`, you need to invoke the repair tool and issue the **beginrepair** command with the pathname of `OBJ` as the argument. Once the repair session is started, both the local and global replicas of `OBJ` are visible at `OBJ`/local (read-only) and OBJ/global (mutable and serving as the workspace for storing the repair result for OBJ and its descendants). The central process of repairing the local/global conflicts on OBJ is to iterate the local-mutations-list containing all the local updates performed on OBJ or its descendants, which can be displayed by the **listlocal** command. Each operation in the list must be accounted for and the repair tool cooperates with Venus to maintain the current-mutation being iterated. The **checklocal** command can be used to show the conflict information between the current-mutation and the global server state. You can advance the iteration to the next operation using either the **preservelocal** or the **discardlocal** command with the former replaying the current-mutation operation on the relevant global replicas. You can also use the **preservealllocal** and **discardalllocal** commands to speed up the iteration. Because the global replica OBJ is mutable, existing tools such as emacs, etc. can be directly used to make the necessary updates. The **quit** command is used to either commit or abort the repair session. The man page on on the repair commands contains more detailed information, and the following simple example illustrates the main process of repairing a local/global conflict.

Suppose that during disconnection, a user creates a new directory `/coda/usr/luqi/papers/cscw/figs` and stores a new version for file `/coda/usr/luqi/papers/cscw/paper.tex`. However, during the disconnection his co-author also creates a directory `/coda/usr/luqi/papers/cscw/figs` and stores some PS files in it. Upon reintegration a local/global conflict is detected at `/coda/usr/luqi/papers/cscw`.

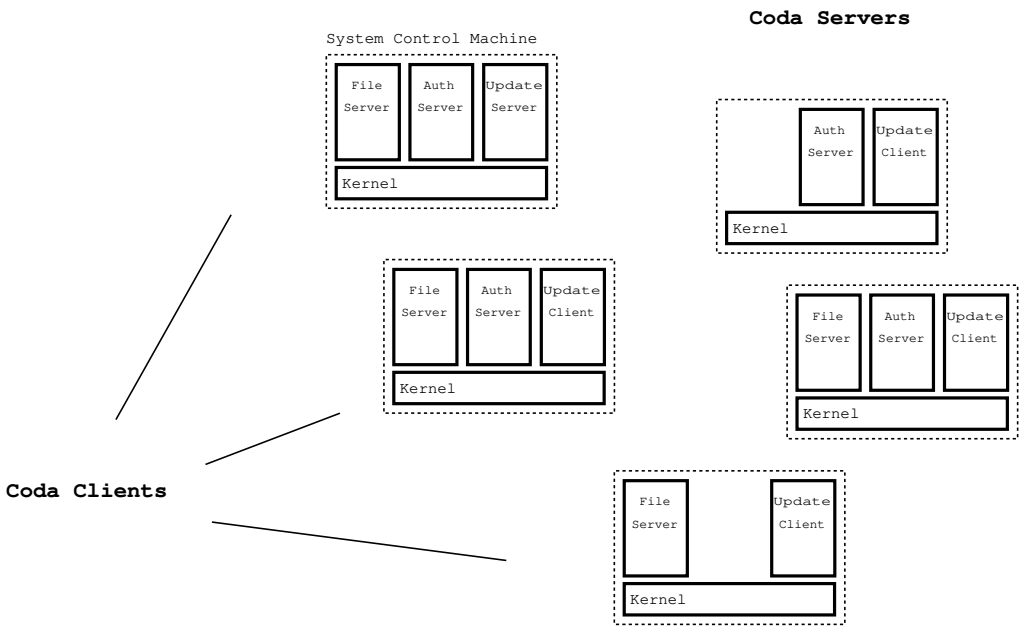
```
% ls -l /coda/usr/luqi/papers/cscw
lr-r-r-  1 root           27 Dec 20 00:36 cscw -
> @@7f000279.00000df3.0001f027
% repair
* begin
Pathname of object in conflict?  [] /coda/usr/luqi/papers/cscw
a local-global-conflict repair session started
```

```
the conflict is caused by a reintegration failure
use the following commands to repair the conflict:
        checklocal
        listlocal
        preservelocal
        preservealllocal
        discardlocal
        discardalllocal
        setglobalview
        setmixedview
        setlocalview
a list of local mutations is avail-
able in the .cml file in the coda spool directory

* ls -l /coda/usr/luqi/papers/cscw
total 4
drwxr-xr-x  3 luqi         2048 Dec 20 00:51 global
drwxr-xr-x  3 luqi         2048 Dec 20 00:51 local

* listlocal
local mutations are:

Mkdir   /coda/usr/luqi/papers/cscw/local/figs
Store   /coda/usr/luqi/papers/cscw/local/paper.tex (length = 19603)

* checklocal
local mutation: mkdir /coda/usr/luqi/papers/cscw/local/figs
conflict: target /coda/usr/luqi/papers/cscw/global/figs exist on servers

* discardlocal
discard local mutation mkdir /coda/usr/luqi/papers/cscw/local/figs

* checklocal
local mutation: store /coda/usr/luqi/papers/cscw/local/paper.tex
no conflict

* preservelocal
store /coda/usr/luqi/papers/cscw/global/paper.tex succeeded

* checklocal
all local mutations processed

* quit
commit the local/global repair session?  [yes]
```

Chapter 3. System Overview

Each Coda client sees the Coda File System as a single tree, `/coda`. In reality this tree is an illusion supported by *System Control Machine* (SCM), several dedicated file servers, and a local area network. One of the servers may double as the (SCM). Figure 3-1 illustrates the server organization.

Figure 3-1. Typical Coda server organization.



Machines

The Coda architecture recognizes three types of machines, *clients*, *servers* and a *system control machine* (or SCM). Client machines are typically single-user workstations used to access shared information. These machines are not trusted by servers except for the duration of a login session by an authenticated user. Server machines are secure, trusted machines whose purpose is to service client requests for shared data. As the custodians of shared information, servers must require authentication of each user before releasing shared data to the users client workstation. The third machine type is the system control machine (SCM). The purpose of the SCM is to provide a single point of control for ease of administration. Logically, the SCM is distinct from the servers, but, physically, the SCM can also act as a server.

Processes

Each of the dedicated file servers must run a number of processes. These processes are shown in Table 3-1 and are described below.

Table 3-1. Server processes

File Server Process	**The codasrv process interacts with the venus process on clients. Together they fulfill user requests for shared data stored on the server. When started, the server process will salvage the file system. The presence of a file called `SHUTDOWN` in the `/vice/srv` directory indicates that the server process exited normally.**
Auth Server Process	**The auth2 process runs on all servers. It validates user passwords and issues a token for that user if the password is correct. However, passwords may only be changed at the SCM. Hence, the password database is read-only replicated to all servers and the SCM maintains the read-write replica. Changes to the password file are updated automatically through the updateclnt/updatesrv processes described below. On all servers (except the SCM), auth2 is invoked with the `-chk` option.**
Update Client Process	**The updateclnt process works in conjunction with the updatesrv process (running on the SCM) to keep read-only copies of system files and databases in sync with their corresponding read-write copy. The updateclnt process checks with the updatesrv process on the SCM periodically to ensure that the read-only copy on this server is the latest copy. Thus, when the read-write copy is updated, the read-only copies will be automatically updated after some delay.**

Figure 3-2 lists the typical processes of a running file server.

Figure 3-2. Typical processes on file-server

```
  PID TT  STAT      TIME COMMAND
```

```
    0 ??  DLs    0:35.41 (swapper)
    1 ??  Is     0:00.26 /sbin/init
    2 ??  DL     0:39.89 (pagedaemon)
  115 ??  Ss     4:27.31 syslogd
  198 ??  Is    55:36.88 update 30
  200 ??  Ss     2:06.16 cron
 7297 ??  Ss     0:00.61 /usr/pkg/sbin/rpc2portmap
 7301 ??  Ss     0:12.36 /usr/pkg/sbin/updatesrv -p /vice/db
 7601 ??  Is     0:04.51 auth2
 9624 ??  Ss     0:05.78 updateclnt -h falla -q coda_udpsrv
 9907 ??  S<s    0:11.37 codasrv -t 1000 -trunc 5 -
rvm /dev/rwd1a /dev/rwd0g 94
 9899 p0  IW     0:00.02 /bin/sh /usr/pkg/sbin/startserver
```

Data Location

The information stored on Coda servers is organized into several directories. These directories are are described below.

- `/vice/auth2` This directory contains information related to the authentication process, including its log file.
- `/vice/bin` contains the Coda file system binaries for servers and the SCM.
- `/vice/db` contains the log file for the update processes as well as a number of databases important to servers.
- `/vice/srv` contains information related to the server process, including its log file.
- `/vice/vol` contains information related to the volumes contained in the Coda file system.
- `/vice/vol/remote` exists only on the SCM and contains volume information for all remote servers.
- `/vice/misc` is where the updateclnt and updatesrv processes live and store their logfiles.

A more detailed description of the files in the these directories is listed in Appendix Appendix B.

Chapter 4. Obtaining Coda

As of time of writing, Coda is supported on Linux, NetBSD and FreeBSD. Alpha quality Windows95/98 Coda client and WindowsNT Coda server are also available for testing purposes. We provide source code tarballs, precompiled Linux binaries in RPM packages, FreeBSD ports and NetBSD source packages.

> **Tip:** Always check the INSTALL and README files on (ftp://ftp.coda.cs.cmu.edu/pub/coda/*platform*/ (ftp://ftp.coda.cs.cmu.edu/pub/coda)) for last minute changes and updates that have not yet found their way into the manual.

Here are the specific steps to obtain packaged distributions for supported platforms:

Linux. Download RedHat RPMS for lwp, rvm, rpc2, coda-debug-client, and, if necessary, coda-debug-server from ftp://ftp.coda.cs.cmu.edu/pub/coda/linux/i386/ (ftp://ftp.coda.cs.cmu.edu/pub/coda/linux/i386/)

FreeBSD. Download FreeBSD ports for lwp, rvm, rpc2 and coda from ftp://ftp.coda.cs.cmu.edu/pub/coda/freebsd/ and untar them into `/usr/ports` directory on your system:

```
# cd /usr/ports
# tar xvzf path_to_downloaded_files/ports-lwp-version.tgz
# tar xvzf path_to_downloaded_files/ports-rvm-version.tgz
# tar xvzf path_to_downloaded_files/ports-rpc2-version.tgz
# tar xvzf path_to_downloaded_files/ports-coda-version.tgz
```

NetBSD. Download NetBSD source packages for lwp, rvm, rpc2 and coda from ftp://ftp.coda.cs.cmu.edu/pub/coda/netbsd/ and untar them into `/usr/pkgsrc` directory on your system:

```
# cd /usr/pkgsrc
# tar xvzf path_to_downloaded_files/pkgsrc-lwp-version.tgz
# tar xvzf path_to_downloaded_files/pkgsrc-rvm-version.tgz
# tar xvzf path_to_downloaded_files/pkgsrc-rpc2-version.tgz
# tar xvzf path_to_downloaded_files/pkgsrc-coda-version.tgz
```

Source tarballs. You can obtain source code tarballs from ftp://ftp.coda.cs.cmu.edu/pub/coda/src/. To compile Coda you mast get following tarballs: `lwp-`*`version`*`.tar.gz`, `rpc2-`*`version`*`.tar.gz`, `rvm-`*`version`*`.tar.gz` and `coda-`*`version`*`.tar.gz`.

> **Important:** FreeBSD ports and NetBSD packages are actualy only frameworks for compilation, they will download source tarballs and build Coda on your system. If you wish to perform Coda installation at later time, when Internet access would not be available, please download tarballs listed above and

put them into `/usr/ports/distfiles` (FreeBSD) or `/usr/pkgsrc/distfiles`(NetBSD) on your system.

Chapter 5. Configuring Kernels for use with Coda clients

The Coda servers may be run on unmodified kernels. The Coda codasrv process is user-level code that uses existing kernel services. The Venus client, however, requires some Coda-specific changes be made to the kernel. The kernel specific changes add Coda specific definitions to the *vnode* layer in the kernel which are needed to satisfy the requirements of the Coda file system to the VFS layer in the kernel.

VFS Interface

In Unix systems, the code which translates user-generated system calls into file system requests is called the *VFS Interface*. Coda provides a VFS Interface by providing the necessary support for systems calls such as `creat()`, `open()`, `stat()`, etc.

Communication between Venus and the kernel occurs through a character device `/dev/cfs0` (part of Coda) which provides the kernel with access to Venus for handling the vsf<->vnode kernel interface. Please see Chapter 6 for the major and minor device numbers specific to a supported platform.

The rest of this chapter will focus on integrating the Coda code that satisfies the Vnode interface into the kernels of supported platforms.

Configuring a Linux kernel

Configuring a FreeBSD kernel

All modern releases of FreeBSD (3.x and 4.x) already include all necessary to configure kernel for Coda client. Using Coda client under FreeBSD, you have two options: either to use Coda *kernel loadable module* (KLD), or to build custom kernel with Coda support. First option doesn't require any significant changes to your installation and can be used on systems without source code installed.

Loading Coda kernel module. To experiment with Coda client, you can load Coda KLD using following command: **kldload coda**. To verify that module is loaded run **kldstat**, `coda.ko` should be listed in its output, which could look similar to this:

```
Id Refs Address    Size     Name
```

```
1    6 0xc0100000 1cda44   kernel
2    1 0xc02ce000 15648    coda.ko
3    1 0xc0e1a000 7000     procfs.ko
4    1 0xc0e28000 7000     ipfw.ko
5    1 0xc0e4a000 15000    linux.ko
6    1 0xc0e80000 2000     green_saver.ko
```

Coda module can be unloaded with **kldunload coda**. To get FreeBSD load Coda module automaticaly on system startup, you should add `coda_load="YES"` line to `/boot/loader.conf`.

Compiling Coda support into the custom kernel. For guidelines on compiling and installing new FreeBSD kernels, please refer to Configuring the FreeBSD Kernel (http://www.freebsd.org/handbook/kernelconfig.html) chapter of the FreeBSD Handbook. Coda support requires including following lines into kernel configuration file:

```
options         CODA                #CODA filesystem.
pseudo-device   vcoda   4           #coda minicache <-> venus comm.
```

This is the only Coda-specific modification requred.

Configuring a NetBSD kernel

Chapter 6. Installing and Configuring a Coda Client

A Coda client can be run on any workstation that has kernel support for Coda in its kernel. Supported UNIX™-like platforms are currently Linux, FreeBSD and NetBSD.

Installing and Configuring the Coda Client Binaries and Documentation

Linux. Install RPMS using following commands:

```
# rpm -Uvh lwp-release.i386.rpm
# rpm -Uvh rvm-release.i386.rpm
# rpm -Uvh rpc2-release.i386.rpm
# rpm -Uvh coda-debug-client-release.i386.rpm
```

FreeBSD. Build and install coda5_client port from `/usr/ports/net/coda5_client`:

```
# cd /usr/ports/net/coda5_client
# make
# make install
```

NetBSD. Build and install coda5_client source package from `/usr/pkgsrc/net/coda5_client`:

```
# cd /usr/pkgsrc/net/coda5_client
# make
# make install
```

Windows95/98. Install Coda client by running ??????.

Currently, client files for the supported platforms are installed into the following directories:

Linux.

— user applications - `/usr/bin`;

— system binaries - `/usr/sbin`;

— other files - `/etc`.

FreeBSD.

— user applications - `/usr/local/bin`;

— system binaries - `/usr/local/sbin`;

— other files - `/usr/local/etc`.

NetBSD.

— user applications - `/usr/local/bin`;

— system binaries - `/usr/local/sbin`;

— other files - `/usr/local/etc`.

Windows95/98.

—All executables can be found in the directory `C:\usr\coda\bin`.

> **Note:** Please make sure your PATH environment variable is set properly for your platform. That is, ensure FreeBSD has `/usr/local/{bin,sbin}` and NetBSD has `/usr/pkg/{bin,sbin}` in its default search paths. Linux installs Coda into the default system paths, so Linux should not need its default search path modified.

Linux and the BSD's: running venus, the client cache manager

Once the binaries and documentation are installed, configuration of venus is identical for all platforms when venus-setup is used. To run the script, type:

venus-setup {comma,seperated,list,of,servers} {cache_size_in_kilobytes}

The venus-setup script does all the hard work, it will setup the coda control files, create `/dev/cfs0` to communicate with the kernel, ... It also initializes a directory for cache files. In your first Coda run we recommend a small cache, say 20MB. The cache size should be at least 10Meg, typically 60-200Meg is used. Do not go above 300Meg. All the files created will be placed under `/usr/coda`. You should make sure that there is enough space in the file system on which `/usr/coda` resides to hold a fully populated cache.

For example, **venus-setup** `micky,minnie,goofy 40000` will setup up venus to look for the set of servers micky, minnie and goofy and configure it to use 40MB of disk space for caching. This information is kept in `/etc/coda/venus.conf`. We strongly recommend that you initially try your client against testserver.coda.cs.cmu.edu first, and keep the cache size to 20000.

Start venus with **venus &**. An `-init` flag can be given when venus is started to flush the local cache contents. Installation procedure performed by venus-setup forces venus to inititialize cache upon first startup. The `-init` flag can be given if Coda cannot recover it's cache after a crash or after re-editing the `venus.conf` file manually.

Observe the venus log with **tail -f /usr/coda/etc/console**, it will tell you when venus has started. Use **codacon** to see the actions of the client in more detail, such as the communication between venus and the codaservers.

It is possible to see the upcalls from the kernel to venus by turning up logging, but they are not very interesting. (To turn on minimal debugging, type **vutil -d 1** and then **tail -f /usr/coda/coda.cache/venus.log**.)

To halt venus, type:

```
# vutil shutdown
# umount /coda  #(Linux only)
```

Or you can kill -9 venus, if you must.

Linux:

- Before restarting venus, `/coda` must be unmounted. If umounting `/coda` gives trouble, make sure to exit all process that hang on to the Coda filesystem, e.g. by having files open or being cd'd into /coda. A utility like lsof and fuser can help with this.
- `/proc/fs/coda` has interesting Coda statistics.
- You can enable kernel debugging with **vutil -kdebug 4095** and call tracing with **vutil ktrace 1**. The messages appear in `/var/log/messages`.

Windows 95: Starting and Configuring a Coda client

During installation you will be prompted for the IP address of your Coda server(s). Enter this as indicated.

CodaStart. The CodaStart is a Win32 windows-based application to control and observe the Windows 95 Coda client. It will be enhanced in the future. For now it provides a convenient way to start venus.exe.

It also displays the kernel-venus communication for debugging purposes. Printing the messages can be stopped by unticking the Monitor check box. The Reset button clears the display.

When you start venus for the first time, or you want to reinitialize its cache, tick the Init venus check box. This will start venus with the `-init` and `-cf 1500` flag set. To add or override flags use the Configure button. Start venus by clicking the left Start button. The 'Status' message will tell you 'Running' when venus starts.

Coda will be mounted automatically on the drive specified in the `C:\usr\coda\etc\vstab` file. `vstab` is created by the installer.

To unmount, type **cfs uk** in a DOS window. This will cleanly shut down venus as well.

You are now ready to browse through the Coda filesystem using the explorer!

> **Important:** In some installations the DPMI DOS Extender window suspends when it is not active. In this case untick the window property Properties→Misc→Background→Always Suspend. If it is unticked, ticking and unticking it again might help. Also untick the Termination flag, to allow Coda to automatically shutdown, when the system shuts down. For your convenience tick the Close on Exit check box in the Program tab.

Upgrading from a previous Release

If you are upgrading Coda from a previous release, do not run venus-setup again! Simply obtain the packages for your platform as indicated above. Then, shutdown venus by becoming root and issue the following command:

```
# vutil -shutdown
```

Next, install the binary packages following the installation instructions above. Make sure you have the appropriate kernel or module for *BSD systems or the correct module for Linux systems. The `INSTALL.`*`platform`* document available from the Coda ftp site will contain relevant notes about upgrading your system core components.

Once the new binaries are installed and the any necessary kernel updates are completed, you need to start the new venus for the first time by becoming root and typing:

```
# venus -init &
```

The `-init` flag is used to re-initialize the `/usr/coda/venus.cache/`.

Configuration changes made by venus-setup

This section describes the changes that must be made to various configuration files.

/etc/coda/venus.conf. The Coda client configuration file `venus.conf` in `/etc/coda` is copied from `/etc/coda/venus.conf.ex` if it doesn't yet exist. All configurable options in this file are extensively documented by comments in the file itself.

/etc/rc.*. To automatically start venus at boot time, the script rc.venus is provided in `/usr/local/etc` for FreeBSD systems and `/usr/pkg/etc` for NetBSD. venus.init is provided for RedHat Linux in `/etc/rc.d/init.d`.

RedHat Linux users should create the following symbolic link if it doesn't already exist:

```
# ln -sf /etc/rc.d/init.d/venus.init /etc/rc.d/rc3.d/S97venus.init
```

This will allow init to process the file and start venus at boot time.

FreeBSD. FreeBSD system users should create `/usr/local/etc/rc.d/venus.sh` (with `rwxr-xr-x` permitions), containing:

```
#!/bin/sh
echo -n " venus"
/usr/local/sbin/venus &
```

NetBSD. NetBSD system users should create `/usr/pkg/etc/rc.d/venus.sh` (with `rwxr-xr-x` permitions), containing:

```
#!/bin/sh
echo -n " venus"
/usr/pkg/sbin/venus &
```

/etc/services. The following port numbers are added to `/etc/services` by venus-setup:

```
coda_opcons     1355/udp                  # Coda opcons
coda_auth       1357/udp                  # Coda auth
coda_udpsrv     1359/udp                  # Coda udpsrv
coda_filesrv    1361/udp                  # Coda filesrv
coda_venus      1363/udp                  # Coda venus
coda_backup     1407/tcp                  # coda backup service
codacon         1423/tcp        venus.cmu       # Coda Console
coda_aux1       1431/tcp                  # coda auxiliary service
coda_aux1       1431/udp                  # coda auxiliary service
coda_aux2       1433/tcp                  # coda auxiliary service
```

```
coda_aux2      1433/udp                # coda auxiliary service
coda_aux3      1435/tcp               # coda auxiliary service
coda_aux3      1435/udp                # coda auxiliary service
```

/dev/cfs0. The following major and minor numbers are used for the various platforms:

- Character-special with Major 67 and minor 0 for Linux

  ```
  # mknod /dev/cfs0 c 67 0
  ```

- Character-special with Major 93 and minor 0 for FreeBSD

  ```
  # mknod /dev/cfs0 c 93 0
  ```

- Character-special with Major 46 and minor 0 for NetBSD 1.2

  ```
  # mknod /dev/cfs0 c 46 0
  ```

- Character-special with Major 51 and minor 0 for NetBSD 1.3

  ```
  # mknod /dev/cfs0 c 51 0
  ```

Please be sure to check the INSTALL.*platform* to make sure these numbers have not changed. For Linux and FreeBSD, the above numbers are officially registered. Coda on NetBSD at present does not have formally registered major device numbers so these are subject to change from release to release.

Where Data and Configuration Information is Stored.

- `/usr/coda/etc` Coda configuration files critical for venus startup.
- `/usr/coda/venus.cache` Cached copies of Coda files.
- `/usr/coda/spool` Saved CMLs from automatic or manual checkpointing and aborted reintegration.
- `/usr/coda/tmp` Temporary hoard related files.
- `/etc/services` Various port numbers used by Coda for client and server communication.

Chapter 7. Installing a Coda Server

Introduction

A Coda cell is a set of servers which all believe one member of the set to be the master, or SCM server. All modifications to important Coda databases should be made on the SCM, otherwise the SCM plays the role of an ordinary Coda server. The updateclnt and updatesrv daemons will then propagate changes from the SCM to the other servers.

The first server setup must be the SCM. This chapter is divided into three sections: installing the binaries, configuring the system using the configuration script vice-setup (for both SCM and non-SCM servers) and finally, a detailed look at what needs to be done to setup a server manually. Before beginning, we have provided some overview information to assist in setting up the server.

Recoverable Virtual Memory

To help ensure that data is not lost or left in inconsistent state between server restarts, Coda uses *Recoverable Virtual Memory* (RVM) which is a transaction system that maintains the system state of the Coda server meta data. RVM is a data logging transaction system which writes modifications into the RVM log and upon truncation or restart incorporates such modifications into the RVM data file. Both log and data file for RVM are ideally held on raw partitions.

Note: This should not be confused with Virtual Memory.

Upon startup, the Coda servers use RVM to restore the Coda system state. For optimal performance you should have dedicated disk partitions for metadata and log partitions, ideally log partition should reside on its own disk. However, a disk sharing the log partition with other disk partitions or the use of a log file may be used at corresponding loss of optimal performance.

RVM metadata storage. This is a file or raw partition for RVM metadata. You can use a file but it will be quite slow on a larger server. This partition must be around 4% of the total size of the files you wish to store under /vicepa (e.g. on a 2GB server we use around 80MB of rvm data). For first installations we recommend the default 22MB options, and using *files* for RVM log and data.

Note: Windows NT Setup creates the file c:\coda\rvm\DATA. Use this for RVM metadata.

Virtual memory. The metadata, held in the RVM data file, is memory mapped. You need that amount of space as virtual memory on your system, in addition to virtual memory to run the server (~6MB) and other software.

RVM transaction log. This is a LOG file, preferably a raw partition on a disk by itself. This needs not be large, a few MB's are fine.

Note: Windows NT Setup creates the file c:\coda\rvm\LOG. Use this for RVM transaction log.

Please refer to the section called *RVM Initialization* for more details on RVM.

Server Disk Organization

Coda servers require a minimum of 2 disk partitions for optimal performance (one raw partition for the RVM Log, one raw partition for RVM Data and one regular UNIX file system to hold file data store in Coda), data security and protection from accidental deletion. For additional performance gains the RVM Log partition should be on its own disk to avoid head movement latencies which reduces disk seek times on log operations. Optionally, `/vice` can be a separate partition for the same reasons it is advantages to have `/var` as a separate partition.

However, other configuration may be used such as having the RVM Data and Log information stored as regular UNIX files at a loss in performance and data security. Also, if more than one Storage Area Data is needed on a Coda Server (the default directory is called `/vicepa`), the additional storage areas must be separate partitions (different partition from `/vicepa` the default, initial storage area for data under Coda) and mounted as `/vicepb`, for example.

The table below shows a possible partitioning of disks on Coda servers with their respective purpose, mount points, typical sizes and consistency check program.

Note: Sizes of these partitions were taken from one of the Coda servers at CMU-SCS, the actual sizes may vary at other installations.

Table 7-1. Example of Partitions for a Coda Server

Partition	Storage Purpose	Mounted	Typical Size	Whether fscked
hda2	**RootandUser File System**	**/**	**650MB**	**Yes**
hda5	**Var file system**	**/var**	**100MB**	**Yes**
hda3	**Vice File System**	**/vice**	**300MB**	**Yes**
hdc1	**RVM Log**	**Not**	**12MB**	**No**
sda1	**RVM Data**	**Not**	**130MB**	**No**
sda2	**Coda FS Data0**	**/vicepa**	**1.6GB**	**Yes**

sda3	**Coda FS Data1**	**/vicepb**	**1.6GB**	**Yes**
sda5	**Coda FS Data2**	**/vicepc**	**1.6GB**	**Yes**

Installation and Configuration

Tip: Always check the INSTALL and README files on (ftp://ftp.coda.cs.cmu.edu/pub/coda/*platform*/ (ftp://ftp.coda.cs.cmu.edu/pub/coda)) for last minute changes and updates that have not yet found their way into the manual.

Installing the Coda Server Binaries and Documentation

Currently, server files for the supported platforms are installed into the following directories:

Linux.

— user applications - `/usr/bin`;

— system binaries - `/usr/sbin`;

— configuration files - `/etc/coda`.

FreeBSD.

— user applications - `/usr/local/bin`;

— system binaries - `/usr/local/sbin`;

— configuration files - `/usr/local/etc/coda`.

NetBSD.

— user applications - `/usr/pkg/bin`;

— system binaries - `/usr/pkg/sbin`;

— configuration files - `/usr/pkg/etc/coda`.

Note: Please make sure your PATH environment variable is set properly for your platform. That is, ensure FreeBSD has `/usr/local/{bin,sbin}` and NetBSD has `/usr/pkg/{bin,sbin}` in its

default search paths. Linux installs Coda into the default system paths, so Linux should not need its default search path modified.

There is one directory used by all platforms: `/vice` which is used to store the supporting Coda configuration and Coda databases. It does not need to be in the PATH.

Here are the specific steps to install the binaries and documentation on supported platforms:

Linux.

```
# rpm -Uvh lwp-release.i386.rpm
# rpm -Uvh rvm-release.i386.rpm
# rpm -Uvh rpc2-release.i386.rpm
# rpm -Uvh coda-debug-server-release.i386.rpm
```

FreeBSD. Build and install coda5_server port, other ports will be built and installed as dependencies:

```
# cd /usr/ports/net/coda5_server
# make
# make install
```

NetBSD. Build and install coda5_server source package, others will be built and installed as dependencies:

```
# cd /usr/pkgsrc/net/coda5_server
# make
# make install
```

Configuring A Coda Server

Server setup is similar for all platforms.

Before Beginning

The `/vice` directory will be created by the setup script automatically if it does not already exist. However, if your root partition does not have enough additional space (e.g. the `SrvLog` can become huge - over 10MB under certain circumstances), you will need to create a partition to be mount as `/vice`, format the partition as normal UNIX file system, create the mount point `/vice`, add the partition and mount information to `/etc/fstab` and mount it by hand *BEFORE* running the setup script.

Alternatively, `/vice` may be a symbolic link to a directory on an existing partition with enough space but it also must be created before running the setup script, vice-setup. Sub-directories needed under `/vice` will be created by vice-setup.

Also note that the setup of an SCM differs in some important ways from a non-SCM setup. So, when vice-setup asks: `Will this machine will be the SCM?` it is very important this question is answered correctly.

Only one SCM should be setup per Coda Cell and Coda does not support multiple-homing cells at this time.

While vice-setup sets up things that are common for both the SCM and non-SCM servers, different helper scripts will be called depending on whenever the answer to question `Is this the SCM?` is `yes` or `no`.

SCM Setup vice-setup.

```
vice-setup-scm
vice-setup-user
vice-setup-rvm
vice-setup-srvdir
```

non-SCM Setup vice-setup.

```
vice-setup-rvm
vice-setup-srvdir
```

Vice-setup invokes these scripts in the order listed above.

Setting Up A Coda Server

If this is the first (or only) server being setup for a Coda cell, it must be setup as the SCM by answering `yes` to the question when asked by vice-setup. The SCM coordinates the sync'ing of Coda database and global configuration files needed to keep track of data stored in Coda. The SCM also coordinates the authentication of a Coda user.

If you are adding a non-SCM server which keeps copies of databases, but does not distribute them to the rest of the machines comprising a Coda Cell answer `no` to the question.

Other than the distinctions indicated above, the SCM plays no special role in a Coda cell and can actually be down for a short time without denying service in a multiple-machine cell site. However, is extremely important to only have one SCM.

> **Note:** running a server seriously dips into system virtual memory. Running a Coda server, a Coda client *and* X11, we have observed that slightly over 64MB of available VM is needed. The **top**

command gives information on memory, CPU and process usage. Therefore, we do not recommend running a Coda client on a Coda server and only run X, when performing server maintenance, if X is convenient to use.

Whether setting up an SCM Coda server or adding a server to an existing Coda cell, make sure the following items are taken care before running the setup script:

1. Create an empty directory (e.g. `/vicepa`) where the file server will keep user files. If more disk space is required than the size of single partition, additional directories on separate partitions can be created (e.g. `/vicepb`).
2. Allocate a raw partition for RVM meta-data (you can use a file but this will be slow on a medium to large server). This partition must be around 4% of the total size of the files you wish to store under `/vicepa` (e.g. on a 3.3GB server we use around 130M of rvm data). Consider 10M to be the minimum.
3. Allocate a LOG partition, preferably on a separate disk. This needs not be large (12MB is large enough for a 3.3GB partition which is the largest default configuration provided by vice-setup-rvm script).
4. Prepare two different secret tokens of up to 8 characters long (e.g. `elephant`). The RVM files involve the journaling/transactional aspects of Coda and communicating this information between servers and via the loop-back device when the SCM updates itself. The tokens are used to secure server-server communications.
5. If the root partition is not large enough to hold `/vice`, having alternative measures setup as outlined in the section called *Before Beginning*.

Once you have the above items available, run **vice-setup**. This command provides several useful “canned” configuration for setting up a Coda server. It does a number of things "behind the scenes" as well (such as setting up the directory `/vice`, if it does not exist and populating it with the sub-directories `backup db srv vol bin`) while asking the following questions:

```
Enter a random password for auth2 authentication :
```
This must be up to eight characters in length.

```
Enter a random password for volutil authentication :
```
This must be up eight characters in length and different from the auth2 password.

```
Enter a random password for update authentication :
```
There are no length limitations, preferable it is different from the auth2 and volutil passwords.

```
Do you want to start the server at boot time? (y/n)
```

Answering `yes` to this question creates the file `/vice/srv/STARTFROMBOOT`, which must be present for either rc.vice or vice.init to start the server at boot time. Removal of this file will prevent the Coda server from starting automatically at boot time.

```
Is this the master server, aka the SCM machine? (y/n)
```

For the first Coda machine being setup in a Coda cell, the answer must be `yes`. If you are adding a server to an existing Coda cell, the answer must be `no`.

If you have answered `no` to the above question, you will be asked for the HOSTNAME of an existing SCM:

```
Enter the hostname of the SCM machine :
```

This question is only asked if the machine is not the SCM. The host name entered must be that of the SCM which distributes the Coda cells database and global configuration files.

If you have answered `yes` to the question about setting up master SCM machine, you will see the message: `Now installing files specific to the SCM...` and get following questions:

```
serverid:
```

All servers in a coda cell need to have a unique number to identify them. The servername to identifier mappings have to be defined by the administrator in the file `/vice/db/servers` on the SCM according to the following format:

servernameX.domain.name 1
servernameY.domain.name 2
...

There are currently several limitations to which identifiers are actually usable:

- all numbers must fit in a single byte;
- 0 and -1 (255) are used in error conditions;
- 127 is used to identify ‘replicated volumes’.

This leaves us with a usable range of 1-126 and 128-245 for server identifiers.

```
Enter the name of the rootvolume (< 32 chars) :
```
Enter the name of your root volume (e.g. `coda.root`)

After creating root volume system will create privileged user for managing Coda servers. You will see following message:

```
You need to give me a uid (not 0) and username (not root)
for a Coda System:Administrator member on this server,
(sort of a Coda super user)
Enter the uid of this user:
```

We recommend you to enter uid of your local account, using which you will administer Coda servers.

```
Enter the username of this user:
```
Enter your username

If operation was successful, you should see following message:

```
An initial administrative user username (id uid) with Coda pass-
word changeme now exists.
```

```
Are you ready to set up RVM? [yes/no]
```

RVM is the Recoverable Virtual Memory used to keep a copy of Coda Virtual memory on local disk. Answer `yes` to this question or the setup will abort.

```
What is your log partition?
```

While RVM can be placed in regular file, it is strongly recommended to allocate a small partition on a dedicated disk for this purpose. If you have set aside, for example, `/dev/sdc1` under Linux as the log partition, enter the partitions' device name at the prompt.

Important: If you are setting up a *BSD system, you must enter the raw partition name. For example, if you are going to use `/dev/sd2e` you must enter the raw device name `/dev/rsd2e`.

The maximum template size we provide for and have tested is 30MB.

```
What is your log size? (enter as e.g.  '12M')
```

The log partition keeps transaction records for Coda volume updates that may not have been written to the Coda data partition yet. We have not extensively tested values above 30MB and do not recommend more than this. 12MB is a good value for a small to medium sized server. The value may be entered as, `12M` or `12288` for 12 Megabytes. Read the section called *The Metadata Partition* for a more detailed explanation.

```
What is your data partition (or file)?
```

This allows you to specify a partition or file. We strongly recommend a partition which can reside with other system partitions. Remember, if you use a partition, you are using it "raw" . it does not contain a file system of any kind. Again, if you are using Linux, you may enter a block device name such as `/dev/sdc1`.

Important: If you are using a *BSD system, you must enter a raw device such as `/dev/rsd2e`.

```
What is the size of you data partition (or file) [22M,44M, 90M, 130M]:
```

This specifies the the size of the RVM data partition. It must be typed exactly as `22M`, `44M` , `90M` , or `130M` . These are associated with default parameters that must be feed to codasrv when it is started. The following table provides an approximate RVM data size to total data storage conversion:

- `22M` is good for up to 500MB;

- `44M` is good for up to 1.0GB;
- `90M` is good for up to 2.2GB;
- `130M` is good for up to 3.3GB.

For a detailed explanation of the trade off between different RVM data sizes vs. data storage space, please see the section called *The Log Partition*.

```
Proceed, and wipe out old data? [y/N]
WARNING
if you have entered erronous entries for RVM log
and RVM data partitions, you could damage your system.  Be sure you
entered correct information!
This is the last chance before any action is taken by the setup script.
Enter y to commit or N to abort.
Where shall we store your data [/vicepa]?
```

Older development versions of coda require `/vicepa` to be present. This version does not. You give a name other than the default. However, we strongly advise not calling it `/coda` as that can be confused with venus.

Once you successfully complete vice-setup you can start your Coda file-server.

Running Coda Server

Startup

Start the rpc2portmap server, update server, client and the auth server, as well as the fileserver by typing:

Linux.

```
#/etc/rc.d/init.d/auth2.init start
#/etc/rc.d/init.d/update.init start
#/etc/rc.d/init.d/codasrv.init start
```

FreeBSD.

```
#/usr/local/etc/rc.vice start
```

NetBSD.

```
#/usr/pkg/etc/rc.d/rc.vice start
```

Windows NT.

```
> codastart
```

Note: Coda file-server process is started using following command:

codasrv -rvm *log_partition data_partition size_of_log_partition*

To facilitate automatic setup procedure and easy changes of configuration we provide a script called startserver, which reads these command-line arguments from the file `/vice/srv.conf` and provides them for codasrv.

The vice-setup-rvm creates `/vice/srv.conf` with values derived from information you have entered during setup process. Should you deviate from one of these canned default values, you will need to either create or modify the `/vice/srv.conf` file to indicate the correct data size of the data partition. For example, one of CMU's Coda servers contains:

```
-rvm /dev/hdc1 /dev/sda1 119070700
```

You should be careful to specify the correct partitions for the RVM log, data segment, and size of the data segment.

The `SrvLog` should show `File Server started`. If not, you have a problem.

Log files

Each process running on a server maintains a log file of its activities. This file may be monitored by running **tail -f** on the particular file. For instance, an operator might want to run this command in a small window on the file server console. The table below shows the absolute pathname of the log file for each server process.

Server Process	Log File
File Server	`/vice/srv/SrvLog`
Authentication	`/vice/auth2/AuthLog`
Update	`/vice/misc/UpdatexxxLog`

The processes append new log entries to the end of the log file. To control the growth of the log files over extended periods of operation, the files should be removed periodically. `SrvLog` may be removed by the `swaplog` option of the **volutil**; this will move the current `SrvLog` to `SrvLog-1` and begin a new log. Swaplog also keep the last 6 log files, by removing, `SrvLog-7` and renaming the remaining log files. a swaplog is also performed during file-server startup. The log files of the other processes should be removed only when their respective processes are not running.

For a non-SCM server this startup sequence will cause the new server to first obtain the current server files such as `/vice/db/ROOTVOLUME` and `/vice/db/VRList` as well as others needed for this server to participate in the Coda cell.

If the codasrv process started correctly, you should see `File Server started` in the SrvLog, othervise you will need to look at the error messages in `SrvLog` and `SrvErr` to determine what went wrong. If the messages are not helpful, you should restart the server with higher debugging level. Consult Chapter 8 about problem solving.

Note: In case of trouble, it is worth to first check with **ps** whether auth2, rpc2portmap, updatesrv and updateclnt are running.

Changing debugging level of Coda server

The codasrv allows you to specify the level of debugging information which is logged. Level 0 is the lowest level of debugging available (i.e. no extra information is printed), while the highest level of debugging available is 1000 (i.e. all debugging information is printed). As with all other server output, debugging output will appear in `SrvLog`. Be careful to turn off debugging when its output is not in use since the log will grow rapidly and can easily fill the partition with debugging information.

There are two ways to change debugging level of the file-server:

1. re-starting server with `-d` *`debug_level`* (e.g. following command **startserver -d 10 &** will provide reasonable amount of additional information);
2. changing debugging level on the fly via **volutil `setdebug` *`debug_level`***.

One final step is needed to make the SCM server function. Use the root volume you specified in vice-setup in place of *`your-root-volume`*. This step creates the volume, which will be mountable by the client software.

```
#createvol_rep your-root-volume E0000100 /vicepa
```

Once you have done this, check if it was successfully created by viewing the log:

```
#tail -f  /vice/srv/SrvLog &
```

Shutdown

To shutdown the file server gracefully, use the volume utility client procedure, volutil. Issue the command **volutil shutdown** and then monitor the log file for the server process (`/vice/srv/SrvLog`) until it reflects that the server has shutdown completely. For example, here are the last few lines before file server shutdown:

```
VShutdown: Taking volume usr.grajen(0xcc0006d1) offline..... Done
VShutdown: Taking volume synrgen.rep(0xcc00057c) offline..... Done
VShutdown: Taking volume i386.omega(0xcd0004fc) offline..... Done
VShutdown: Taking volume trace.rep.0(0xcc00057d) offline..... Done
VShutdown: Taking volume usr.mmv.rep.0(0xcc00057e) offline..... Done
VShutdown: Taking volume trace.backup(0xcc00057f) offline..... Done
VShutdown:  complete.
```

Exploring replication

First you will have to add a second server to your Coda cluster. Install the software and use vice-setup again. This time your server is *not* going to be the SCM. Proceed answering the questions until done.

Adding server to Coda cell

> **Note:** All actions should be performed on SCM, unless other server is mentioned.

1. Your server needs a unique server number, to be added to the `/vice/db/servers` file.
2. Make two new entries in the `/vice/db/VSGDB` file. One for your new server by itself, one of the form: `E0000104` *`scm-server second-server`*. `E0000104` is an example of unique id assigned to the *Storage Group* containing our two servers. Each set of servers that contains replicated volumes should be assigned its unique id.
3. Start updatesrv and updateclnt on the second server.
4. Start codasrv on the second server
5. Make a new volume using **createvol_rep** giving the address of the volume as `E0000104`.
6. Mount the volume.

You can now use this volume and your files will automatically be stored on multiple servers. To temporarily disable a server, and see that things continue to function normally, either shut the server down with **volutil shutdown** or disconnect its network. You can also isolate the server using our network filters with **filcon isolate -s server-name**. Using **filcon clear *`server-name`*** clears the filter.

Modifications made to Coda files during the server outage will be resolved when the files are first accessed. You see message of the form `Resolve path` in the codacon output. By typing **cfs checkservers** you can see if the server is available again.

Underneath vice-setup

The sections contained here describe what vice-setup does for you. This information is useful for those who either wish to customize our default vice-setup script or wish to have a custom server setup outside the scope of Coda's setup script..

RVM Initialization

RVM initialization requires the selection of several parameters, each of which involve tradeoffs. Although the RVM log and data can be kept as regular UFS files, *this is not recommended*. Raw partitions have much stronger data consistency guarantees as well performance advantages. It is probably best to plan out the RVM partitions on paper first, taking into consideration both the effect on RVM performance as well as overall disk usage if you choose not to use the values provided in the vice-setup script.

The Log Partition

The size of the log device is based on available space and issues involving truncation. Larger logs provide a longer accessible history of operations, are truncated less frequently, but each truncation will take a longer period of time. Shorter logs truncate more often, but each truncation takes less time. Log size is also strongly related to server startup time as well. We use a 90M log size, on a storage size roughly 3.2GB spread between two UNIX partitions. (We suggest leaving a little space at the end of the RVM log partition for safety, as RVM automatically adds about one extra page to the amount you specify). Our vice-setup script provides for several default values, the largest of which is 130MB. If you decide to follow the recommendation and use a dedicated partition, creating a 130MB partition will leave plenty of room to expand even if your initial use only suggests a size of 22M, for example.

The log is initialized with rvmutl. At rvmutls prompt, use the command **i**, and then specify the size of the log segment. In specifying the size, you can use `M` for megabyte and `K` for kilobyte. For example, to initialize a log on partition 0g to eight megabytes:

```
#rvmutl
*i
Enter name of log file or device:/dev/sdc1
Enter length of log data area:22M
*q
```

The Metadata Partition

The data segment contains the meta-data of the system such as volume headers, Coda directories, resolution logs, etc. The size of the data segment depends on the amount of disk space for file data, i.e. the size of the `/vicep?` partitions. As a rule of thumb, you will need about 3-5% of the total file data space for recoverable storage. We currently use 4% as a good value under most circumstances. In our systems the data segment is 90Meg for approximately 3.2 gigabytes of disk space. By making it smaller, you can reduce server startup time. However, if you run out of space on the RVM Data partition, you will be forced to reinitialize the system, a costly penalty. So, plan accordingly.

In initializing (or reinitializing), you need to pick several parameters. The first is the starting address of the recoverable segment in your address space. On our servers we start the RVM segment at `0x50000000` for both Linux and BSD based systems on Intel architecture. The second is the amount of space to give the recoverable heap. The heap will obviously grow over use, so plan accordingly. Our heap space is `0x1000000`. We suggest that for the last parameters you use 1Meg (`0x100000`) for the static area, use `80` free lists (or nlists), and a chunksize of `32`. These numbers will work well with the internal structure of the file server and are provided as the default vales in the setup script vice-setup.

To perform the data initialization, run the program rdsinit. rdsinit takes two parameters, the names of the RVM log and data devices. For example, to initialize one of our Intel based servers:

```
# rdsinit /dev/hdc1 /dev/sdb1
Enter the length of the device /dev/hdc1:119070700
Going to initialize data file to zero, could take awhile.
done.
rvm_initialize succeeded.
starting address of rvm: 0x50000000
heap len: 0x1000000
static len: 0x100000
nlists: 80
chunksize: 32
rds_zap_heap completed successfully.
rvm_terminate succeeded.
```

Note: Use of the decimal value for the length of the device and the use of hex values for the address and lengths of the next three values.

Update Monitor

The update monitor is used to propagate changes to the Coda server databases to all servers from the SCM. Client update processes run on all the Coda servers and connect to a server update process running on the SCM. The server process uses the file `/vice/db/files` to determine which files should be kept consistent on all the servers. See the updateclnt(8) and updatesrv(8) man pages for more details.

Create the file `/vice/db/files` on the SCM. Currently our `/vice/db/files` looks like this:

```
ROOTVOLUME
VLDB
auth2.pw
auth2.tk
```

```
auth2.tk.BAK
auth2.lock
servers
prot_users.db
prot_index.db
volutil.tk
update.tk
VRDB
files
VSGDB
dumplist
```

Authentication Database

Coda uses an authentication database that is separate from the UNIX password file. This database is maintained by the SCM. When someone authenticates to Coda, their password is checked against this database and that person is issued a token if they successfully authenticate.

On servers other than the SCM, the Update monitor will ensure that the proper database files are propagated from the SCM. However, you must copy the file `/vice/db/update.tk` to the new server for the Update monitor to work.

Chapter 8. Troubleshooting

The Coda filesystem is still under development, and there certainly are several bugs which can crash both clients and servers. However, many problems users observe are related to semantical differences of the Coda filesystem compared to well-known NFS or SMB network filesystems.

This chapter will point out several logs to look at for identifying the cause of problems. Even if the source of the problem cannot be found, the information gathered from Coda's logging mechanisms will make it easier for people on the coda mailinglist codalist@coda.cs.cmu.edu (mailto:codalist@coda.cs.cmu.edu) to assist in solving the problem(s).

Some of the more common problems are illustrated in detail. At the end of this chapter some of the more involved debugging techniques will be addressed. This will be helpful to developers to isolate problems more easily. At the end there is a whole section describing how to solve some problems with using Coda client on Windows95.

Basic Troubleshooting

Most problems can be solved, or at least recognized by using the information logged by the clients and servers. The first step in finding out where the problems stems from is doing a **tail -f** on the logfiles.

It must also be noted that, when coda clients and servers crash they do not 'dump core', but start sleeping so that we developers can attach debuggers. As a result, a crashed client or server still shows up in the **ps auxwww** output, and only the combination of lack of file-service and error messages in logfiles indicate that something is really wrong.

Since release 5.3.4 servers now actually exit when they crash, create a file `/vice/srv/ZOMBIFY` to force a server to go into an endless sleep again.

Client debugging output.

- codacon is a program which connects to venus and provides the user with run-time information. It is the initial source of information, but cannot be used to look back into the history. It is therefore advisable to always have a codacon running in a dedicated xterm.

  ```
  client$ codacon
  ```

- `/usr/coda/etc/console` is a logfile which contains mostly error or warning messages, and is a place to look for errors which might have occured. When assertions in the code fail, it is logged here.
- `/usr/coda/venus.cache/venus.log` contains more in-depth information about the running system, which can be helpful to find out what the client is or was doing.

Server logs.

- cmon is an ncurses program that can be run on a client to gather and display statistics from a group of servers. When a server goes down it will not respond to the statistics requests, which makes this a simple method for monitoring server availability.

```
client$ cmon server1 server2 server3 ...
```

- `/vice/srv/SrvLog` and `/vice/srv/SrvErr` are the server logfiles.

Other logfiles that could be helpful in discovering problems are:

- `/vice/auth2/AuthLog`
- `/vice/srv/portmaplog`
- `/vice/srv/UpdateClntLog`
- `/vice/srv/UpdateLog`

Client Problems

1. Client does not connect to testserver.coda.cs.cmu.edu.

When you have set up your client for the first time, and it can not connect to the testserver at CMU, there are a couple of possible reasons. You might be running an old release of Coda, check the Coda web-site to see what the latest release is.

Another common reason is that your site is behind a firewall, which blocks, or allows only outgoing, udp traffic. Either try Coda on a machine outside of the firewall, or set up your own server.

The third reason is that the testserver might be down, for maintenance or upgrades. That does not happen often, but you can check whether it is up, and how long it has been running using cmon.

```
client$ cmon testserver.coda.cs.cmu.edu
```

2. Venus comes up but prints `cannot find RootVolume`.

All of the reasons in the previous item could be the cause. It is also possible that your `/etc/services` file is not allright. It needs the entries:

```
# IANA allocated Coda filesystem port numbers
rpc2portmap     369/tcp
```

```
rpc2portmap     369/udp    # Coda portmapper
codaauth2       370/tcp
codaauth2       370/udp    # Coda authentication server

venus           2430/tcp   # codacon port
venus           2430/udp   # Venus callback/wbc interface
venus-se        2431/tcp   # tcp side effects
venus-se        2431/udp   # udp sftp side effect
codasrv         2432/tcp   # not used
codasrv         2432/udp   # server port
codasrv-se      2433/tcp   # tcp side effects
codasrv-se      2433/udp   # udp sftp side effect
```

3. Trying to access a file returns `Connection timed out (ETIMEDOUT).`

The main reason for getting `Connection timed out` errors is that the volume where the file is located is disconnected from the servers. However, it can also occur in some cases when the client is in write-disconnected mode, and there is an attempt to read a file which is open for writing. See the section called *Disconnections.* for more information.

4. Commands do not return, except by using `^C`.

When command are hanging it is likely that venus has crashed. Check `/usr/coda/etc/console` and `/usr/coda/venus.cache/venus.log`.

5. Venus fails when restarted.

If venus complains in `venus.log` about not being able to open `/dev/cfs0`, it is because `/coda` is still mounted.

```
# umount /coda
```

Another reason for not restarting is that another copy of venus is still around, and venus is unable to open it's network socket. In this case there will be a message in `venus.log` stating that RPC2_CommInit has failed.

6. Venus doesn't start.

A reason is that you do not have the correct kernel module. This can be tested by inserting the module by hand, and then listing the available modules. `coda` should show up in that listing. Otherwise reinstall (or recompile) a new module.

```
# depmod -a
# insmod coda.o
# lsmod
```

```
Module                  Size  Used by
coda                   50488   2
```

If the kernel-module can be loaded without errors, check `venus.log`. A message `Cannot get rootvolume name` indicated either a misconfigured server or the `codasrv` and/or `codasrv-se` ports are not defined in `/etc/services`, which should contain the following entries. See above for the entries needed.

7. I'm disconnected and Venus doesn't start.

Put the hostnames of your servers in `/etc/hosts`.

8. I cannot get tokens while disconnected.

Take a vacation until we release a version of Coda which uses it's telepathic abilities to contact the auth2 server. We will add this feature.

9. Hoard doesn't work.

Make sure you have version 5.0 of Coda or later. Before you can hoard you must make sure that:

- You started Venus with the flag `-primaryuser` *`youruid`*
- You have tokens

10. File System Full

You may get a message of the form, `/usr: file system full` when you have not put any extra data on that disk. If the `venus.cache` directory is on that disk, that is probably your problem. You can execute either **cfs flushobject *`object`*** or **cfs flushvolume *`dir`*** to try to free up some disk space.

> **Warning**
>
> Be careful when flushing anything while disconnected. You will lose any changes that you made.

In the long term, the cache size as specified in `/etc/coda/venus.conf` is probably too large for the disk and should be made smaller.

Server Problems

1. The server crashed and prints messages about `AllocViaWrapAround`.

This happens when you have a resolution log that is full. In the `SrvLog` you will usually be able to see which volume is affected, take down it's volume id (you may need to consult `/vice/vol/VRList` on the SCM to do this. Kill the dead (zombied) server, and restart it. The moment it is up you do:

```
# filcon isolate -s this_server
We need to prevent clients from overwriting the log again
# volutil setlogparms volid reson 4 logsize 16384
# filcon clear -s this_server
```

Unless you do "huge" things 16k will be plenty.

2. server doesn't start due to salvaging problems

If this happens you have several options. If the server has crashed during salvaging it will not come up by trying again, you must either repair the damaged volume or not attach that volume.

Not attaching the volume is done as follows. Find the volume id of the damaged volume in the `SrvLog`. Create a file named `/vice/vol/skipsalvage` with the lines:

```
1 <10122>
0xdd000123 <10123>
```

<10122>indicates that a single volume is to be skipped

<10123>volume id of the replica that should not be attached. If this volume is a replicated volume, take all replicas offline, since otherwise the clients will get very confused.

You can also try to repair the volume with norton. Norton is invoked as:

norton [LOG] [DATA] [DATA-SIZE]

These parameters can be found in /vice/srv.conf. See norton(8) for detailed information about norton's operation. Built-in help is also available while running norton.

Note:

1. Often corruption is replicated. This means that if you find a server has crashed and does not want to salvage a volume, your other replicas may suffer the same fate: the risk is that you may

have to go back to tape (you do make tapes, right?). Therefore *first* copy out good data from the available replicas, *then* attend to repairing or skipping them in salvage.

2. Very often you have to take both a volume and its most recent clone (generated during backup) offline, since corruption in a volume is inherited by the clone.

3. How to restore a backup from tape

Tuesday I have lost my email folder - the whole volume `moose:braam.life` was corrupted on server moose ant it wouldn't salvage. Here is how I got it back.

1. First I tried mounting `moose.braam.life.0.backup` but this was corrupted too.
2. On the SCM in `/vice/vol/VRList` I found the replicated volume number `f0000427` and the volume number `ce000011` (fictious) for the volume.
3. I logged in as root to bison, our backup controlller. I read the `backuplog` for Tuesday morning in `/vice/backuplogs/backuplog.DATE` and saw that the incremental dump for August 31st had been fine. At the end of that log, I saw the name `f0000427.ce000011` listed as dumped under `/backup` (a mere symlink) and`/backup2` as spool directory with the actual file. The backup log almost shows how to move the tape to the correct place and invoke restore:

```
# cd /backup2
# mt -f /dev/nst0 rewind
# restore -b 500 -f /dev/nst0 -s 3 -i
Value after -s depends upon which /backup[123] volume we pick to restore backup.
restore> cd 31Aug1998
restore> add viotti.coda.cs.cmu.edu-f0000427.ce000011
restore> extract
Specify volume #: 1
```

4. In `/vice/db/dumplist` I saw that the last full backup had been on Friday Aug28. I went to the machine room and inserted that tape (recent tapes are above bison). This time `f0000427.ce000011` was a 200MB file (the last full dump) in `/backup3`. I extract the file as above.
5. Then I merged the two dumps:

```
# merge /restore/peter.mail /backup2/28Aug1998/f0000427.ce000011 \
> /backup3/31Aug1998/f0000427.ce000011
```

6. This took a minute or two to create /restore/peter.mail. Now all that was needed was to upload that to a volume:

```
# volutil -
h moose restore /restore/peter.mail /vicepa vio:braam.mail.restored
```

7. Back to the SCM, to update the volume databases:

```
# bldvldb.sh viotti
```

8. Now I could mount the restored volume:

```
# cfs mkm restored-mail vio:braam.mail.restored
```

and copy it into a read write volume using cpio or tar.

4. createvol_rep reports RPC2_NOBINDING.

When trying to create volumes, and createvol_rep reports `RPC2_NOBINDING`, it is an indication that the server is not (yet) accepting connections.

It is useful to look at `/vice/srv/SrvLog`, the server performs the equivalent of **fsck** on startup, which might take some time. Only when the server logs `Fileserver Started` in `SrvLog`, it starts accepting incoming connections.

Another reason could be that an old server is still around, blocking the new server from accessing the network ports.

5. `RPC2_DUPLICATESERVER` in the rpc2portmap/auth2 logs

Some process has the UDP port open which rpc2portmap or auth2 is trying to obtain. In most cases this is an already running copy of rpc2portmap or auth2. Kill all running copies of the program in question and restart them.

6. Server crashed shortly after updating files in `/vice/db`.

Servers can crash when they are given inconsistent or bad data-files. You should check whether updateclnt and updatesrv are both running on the SCM and the machine that has crashed. You can kill and restart them. Then restart codasrv and it should come up.

7. Users cannot authenticate or created volumes are not mountable.

Check whether auth2, updateclnt, and updatesrv are running on all fileservers. Also check their logfiles for possible errors.

Disconnections.

As most common problems are related to the semantical differences arising as a result of 'involuntary' disconnections, this section contains some background information of why volumes become disconnected or write-disconnected. And how to get them to reconnect again.

1. Volume is fully disconnected.

There are several reasons why a coda client may have disconnected some or all volumes from an accessible server.

- **Pending reintegration.** When modifications have been made to the volume in disconnected mode, the client will not reconnected the volume until all changes have been reintegrated. Also, reintegration will not occur without proper user authentication tokens. Furthermore, reintegration is suspended as long as there are objects in conflict.

  The most important item here is to have a codacon process running, since it will give up-to-date information on what venus is doing. Venus will inform the user about missing coda authentication tokens, `Reintegration: pending tokens for user` *`uid`*. In this case the user should authenticate himself using the clog command.

  Conflicts, which require us to use the repair tool, are conveyed using the `local object` *`pathname`* `inconsistent` message. Otherwise codacon should show messages about `backfetches`, and how many modifications were successfully reintegrated.

- **Access permissions.** The client may also disconnect when a servers reports an error to an operation, when according to the client this is a valid operation. Causes for this are authentication failure; check tokens using ctokens and optionally obtain new tokens using clog. Or inconsistencies between the data cached on the client and the actual data stored on the server; this will reveal itself as an inconsistent object during subsequent reintegration.
- **Lost connections.** Sometimes the client does not receive a prompt reply from an accessible server, and marks the server as dead. This will ofcourse disconnect the volume if the last server is lost. Once every five minutes, the client automatically verifies connectivity with all known servers, and can thus recover from lost connections. However, this action can also be triggered by the user by excecuting the **cfs checkservers**. If **cfs checkservers** reports that servers are unreachable, it might be interesting to check with cmon, if the server is responding at all, since we might be faced with a crashed server. When a server was considered unreachable, but is successfully contacted after **cfs checkservers**, reintegration will automatically start (when a user has tokens, and there are no inconsistencies).

2. Volume is write-disconnected.

Write-disconnected operation is used as often as *weakly connected mode* to describe this volume state, and they are effectively the same. This is the special situation where a client observes a weak

connectivity with a server, and therefore forces the associated volumes in weakly connected mode. Weakly connected volumes postpone writing to the server to significantly reduce waiting on a slow network connection. Read operations are still serviced by the local cache and the servers, as in fully connected mode. Which is why this mode of operation is also called write-disconnected operation.

The write operations are effectively a continuous reintegration (*trickle-reintegration*) in the background. This mode, therefore, requires users to be authenticated and gives more chance for possible file conflicts. The following points are several reasons for write-disconnected operation.

- **Weak network connectivity.** Venus uses bandwidth estimates made by the RPC2 communication layer to decide on the quality of the network connection with the servers. As soon as the connectivity to one of the servers drops to below the weakly connected treshhold (currently 50 KB/s), it will force all volumes associated with that server into weakly-connected mode. The **cfs wr** can be used to force the volumes back into fully connected mode, and immediately reintegrate all changes.

  To avoid switching to weakly connected mode, use **cfs strong**. This way venus ignores bandwidth estimates. **cfs adaptive** will make venus revert to interpreting bandwidth estimates.

  When the user was not authenticated, or conflicts were created during the write-disconnected operation, the user must first obtain proper authentication tokens or repair any inconsistent objects before the volume becomes fully connected again. Here again codacon is an invaluable tool for obtaining insight into the client's behaviour.

- **User requested write-disconnect mode.** Users can ask venus to force volumes in write-disconnected mode, exchanging high consistency for significantly improved performance. By using the `-age` and `-time` flags in the **cfs wd** commandline, some control is given about the speed at which venus performs the trickle-reintegration. For instance, to perform the trickle-reintegrate more quickly than the default, where only mutations to the filesystem older than 15 minutes are reintegrated. You could use **cfs wd -age 5**, which will attempt to reintegrate all mutations older than 5 seconds.
- **Pending reintegration.** When a volume is write-disconnected, it will stay write-disconnected until a user properly authenticates using clog.

Advanced Troubleshooting

1. Troubleshooting network problems with rpc2tcpdump

rpc2tcpdump is the regular tcpdump, which is modified to decode rpc2 protocol headers. This makes it a very useful tool for analyzing why programs fail to work.

All traffic between venus and the Coda servers can be viewed using the following command:

```
# tcpdump -s120 -Trpc2 port venus or port venus-se
```

To identify problems with clog, for instance which server it is trying to get tokens from.

```
# tcpdump -s120 -Trpc2 port codaauth
```

2. Debugging with gdb

To be able to debug programs that use RVM, most coda related application will go into an endless sleep when something goes really wrong. They print their process-id in the log (f.i. `venus.log` or `SrvLog`), and a user can attach a debugger to the crashed, but still running, program.

```
# gdb /usr/sbin/venus `pidof venus`
```

This makes it possible to get a stack backtrace (**where**), go to a specific stack frame (**frame *x***), or view the contents of variables, (**print *varname***). By installing the coda sources in same place as where the binaries were initially built from, it is possible to view the surrounding code fragment from within the debugger using the **list**.

When using RedHat Linux rpms, you can install the sources in the right place by installing the coda source rpm file.

```
# rpm -i coda-x.x.x.src.rpm
# rpm -bp /usr/src/redhat/SPECS/coda.spec
```

On other platforms look at the paths reported in the backtrace and unpack the source tarball in the correct place.

```
(gdb) where
#0  CommInit () at /usr/local/src/coda-5.3.6/coda-src/venus/comm.cc:175
#1  0x80fa8c3 in main (argc=1, argv=0xbffffda4)
    at /usr/local/src/coda-5.3.6/coda-src/venus/venus.cc:168
(gdb) quit
# cd /usr/local/src
# tar -xvzf coda-5.3.6.tgz
```

Troubleshooting on Windows 95

Common problems

1. Unable to shutdown Windows95.

Check the DOS Windows settings of Venus. The check box in Properties→Misc→Termination must be unticked.

2. I cannot reboot Windows95 and I think it is due to the VXDs loaded for Coda.

Boot your System in DOS mode by pressing **F8** on boot time. **Cd** to the Windows directory and type **edit system.ini**. In the section `[enh386]` you will find following entries:

```
device=c:\usr\coda\bin\mmap.vxd
device=c:\usr\coda\bin\mcstub.vxd
```

Comment them out by using a `;` in front of the lines. Try to restart Windows again.

3. How can I find out why `venus.exe` crashed.

See the section called *Client Problems*. When this happens it might not be possible to restart Venus, if it is still mounted. In this case try to unmount by typing **unmount *`drive`***. If it does not work, you want to reboot the machine.

4. How can I find out more about what has happend

Look in the file `c:\vxd.log`. The file system driver codadev.vxd prints information about all requests and answers in this file. the information is only stored if the debug level has been turned on. the debug level is specified in the registry key `HKLM/System/CurrentControlSet/Services/VxD/Codadev/Debuglevel`. Set the debug level higher than `0` to receive messages in the debug file.

5. I hook my running machine off the network and the explorer blocks.

Venus switches to disconnected mode after a short timeout. After that it should work fine. If it doesn't, check if you have 'network connections' set up in the explorer (e.g. windows shares). Sharing such a network drive blocks your system when no network is available because the SMB filesystem doesn't handle disconnected operations.

Restrictions

- Most command line tools, that talk to Venus through the ioctl interface of the Coda kernel module seem to work even when they print error messages.
- Handling large files (in particular executables) does not work well in a low bandwidth scenario.
- cfs.exe and hoard.exe use absolute pathnames so far.
- Short filenames are not supported under DOS environment yet. You can access files, but you need to use the long filenames. Or was that the other way around?

Chapter 9. Volume Administration

Concepts

Although users view the Coda file system as a hierarchy of directories and files, system administrators view the Coda file system as a hierarchy of volumes. Each volume contains a subtree of related directories and files which is a subtree of the entire file system. Volumes, then, parallel traditional Unix file systems. Like a Unix file system, a volume can be mounted. Thus, the root of a volume can be named within another volume at a mount point. The Coda file system hierarchy is built in this manner and is then mounted by each client using a conventional Unix mount point within the local file system. Since all internal mount points are invisible, the user sees only a single mount point for the entire Coda file system.

All system administration tasks are performed relative to a volume or a set of volumes. So, adding new users requires creating new volumes for their files and directories; quotas are enforced on volumes; and backups are performed on a per-volume basis. The volume abstraction greatly simplifies the administration of large systems.

The Coda file system provides four different types of volumes. The simplest of these is the *non-replicated volume*. Non-replicated volumes reside on a single server and are in the custody of the Coda file server they reside on. The Coda servers work with the venus processes on client workstations to provide a single, seamless view of the file system. However, if a custodian crashes or is otherwise inaccessible, its non-replicated volumes are inaccessible as well.

Thus, the main volume type that is used in typical Coda environments are the *read-write, replicated volumes*. Read-write, replicated volumes are logical volumes which group together multiple read-write, non-replicated volumes. Coda provides protocols which allow read-write, replicated volumes to reside on a number of servers and to be accessed even when some servers are inaccessible.

Table 9-1. Coda volume types

Volume Type	Where Reads are performed	Where Writes are performed	Conflicts
Non-replicated	**Only Custodian**	**Only Custodian**	**No**
Read-Write Replicated	**Any VSG Member**	**Any VSG Member**	**Yes**
Backup	**Only custodian**	**Nowhere**	**No**

Creating a Volume

Typically, volumes consist of a single users data objects or other logically connected groups of data objects. Four factors should be used in dividing the file system tree into volumes.

- The volume is the unit of quota enforcement and, potentially, accounting.
- The rename command is prohibited across volume boundaries. Thus, manipulation of the tree structure at a granularity other than a whole volume (e.g. moving the mount point) or less than a volume (e.g. moving directories or files within a volume) is expensive. Moving a subtree of a volume to another volume requires copying every byte of data in the subtree.
- The size of the volume should be small enough that moving volumes between partitions is a viable approach to balancing server disk utilization and server load. Thus, volumes should be small relative to the partition size.
- Finally, the size of a volume must not exceed the capability of the backup media.

A volume naming convention should also be used by those administrators who create volumes. Volume names are restricted to `32` characters and should be chosen so that given a volume name, a system administrator (who knows the naming conventions) can determine its correct location in the file system hierarchy. The convention used by the Coda project is to name volumes by their function and location. Thus, a replicated volume named `u.hbovik` is mounted in `/coda/usr/hbovik` and contains hboviks data. A project volume is prefixed by `p.` and a system volume is prefixed by `s.` Similarly, volumes containing machine specific object files are prefixed by the machine type. For instance, `p.c.alpha.pmax.bin` contains project coda binaries for our current alpha release and is mounted on `/coda/project/coda/alpha/pmax_mach/bin`.

Use the command createvol_rep(8) to create read-write replicated volumes respectively. This command is actually a script which ultimately invokes the volutil(8) command with the create_rep option at the appropriate server. The volume will contain an access list initialized to `System:Administrators rlidwka` and `System:AnyUser rl`. Creating the volume does not mount the volume within the file system hierarchy. Mounting the volume as well as changing the access list or the quota must be done using the cfs(1) command from a client. A new volume may not be visible at client workstations for some time (see the section called *Building the volume databases* below).

A few concrete examples should clarify the use of some of these commands. On the SCM, the **createvol_rep** `u.hbovik E0000107 /vicepa` will create a replicated volume named `user.hbovik` on each server in the Volume Server Group (VSG) identified by `E0000107`. The file `/vice/db/VSGDB` contains the mapping between VSGs and servers. The names of the replicas will be `user.hbovik.`*n*, where *n* is a number between `0` and `|VSG| - 1`.

Mounting a Volume

In order to use a volume which you have created and added to the appropriate databases, you must mount the volume. Although Unix file systems must be mounted upon reboot in Unix, Coda volumes are mounted only once. To mount a Coda volume, you must be using a Coda client and be authenticated as a user (use the **clog**) who has write access to the directory in which the mount point will be created.

Mount the volumes using the command **cfs** `mkmount` *`filename volname`*. Note that cfs creates *`filename`* automatically. For example, **cfs** `mkmount /coda/usr/hbovik u.hbovik` will create `/coda/usr/hbovik` and then mount the `u.hbovik` volume created in the example in the section called *Creating a Volume*. The volume is now visible to all users of the Coda file system. More information about the cfs command can be found in Chapter 1 and in man page.

> **Warning**
>
> When mounting a volume, avoid creating multiple mount points for it. Coda cannot check for this.

Deleting a Volume

When a volume is no longer needed, it may be purged by running the purgevol or purgevol_rep scripts on the SCM. Before removing a volume, you should probably create a backup for offline storage (see the section called *Dumping and Restoring a Volume*). The volume's mount point should be removed with the cfs(1) command (see the `rmmount` option) before purging the volume (if possible). Note that purging the volume will not purge related backup volumes. Backup and should be purged with the purgevol script or the **volutil** `purge` command.

Dumping and Restoring a Volume

Creating a dump of a replicated volume

On the SCM, you need to clone the read-write copy of the volume. You can use the command **volutil** `clone` *`VolumeId`*. This command will create a read-only volume with the name `binaries.readonly` (assuming that your volume is called `binaries`). Next, you will need to dump this cloned volume to a file with the command **volutil** `dump` *`VolumeId filename`*.

The next section explains how to restore a dumped volume.

Restoring volume dumps

For complete details on the backup/restore process, see Chapter 11. In short, one first needs to get the correct dumpfile, possibly merging differential dumps to an older full dump to get the desired state to be restored. Once this file is obtained, use the volutil(8) restore facility.

volutil restore {*filename*} {*partition*} [*volname* [*volid*]]

The *partition* should be one of the `/vicep?` partitions on the server. You may optionally specify the volumeId and the volume name for the restored volume. This is useful when creating read-only replicated volumes. Note that currently dump files are not independent of byte ordering – so volumes cannot be dumped across architectures that differ in this respect.

After the volume has been restored the volume location databases have to be updated before client can see the restored volume. This is explained in the section called *Building the volume databases*.

Building the volume databases

VLDB. The *Volume Location Data Base*, VLDB, is used to provide volume addressing information to workstations. Copies of the VLDB reside on the servers and are updated periodically. The VLDB lists the latest known location or locations of all on-line volumes in the system. A human readable version of the VRDB is maintained on the SCM in the file `/vice/vol/VRList`.

The VLDB is maintained on the SCM. When you wish to update it, run the bldvldb.sh(8) from `/vice/bin/` on the SCM. The script gathers a copy of the `/vice/vol/VolumeList` file from all the servers, merges it into a single list, and builds a new VLDB. The updateclnt and updatesrv daemons will then propagate the new VLDB to all the servers.

Note: The createvol_rep and purgevol_rep scripts automatically invoke bldvldb.sh.

Building the VRDB. The *Volume Replication Data Base*, the VRDB, is used to provide information about replicated volumes to client workstations. Copies of the VRDB reside on all servers and are updated periodically. The VRDB maps each logical volume to its corresponding set of physical volumes. A human readable version of the VRDB is maintained on the SCM in the file `/vice/vol/VRList`. The `makevrdb` option to the volutil(8) command will create a new VRDB which will automatically be distributed to the other servers.

Building the VSGDB. The *Volume Storage Group Data Base*, VSGDB, is created during setup process and, currently, is maintained by hand. Each valid volume storage group has an entry in this data base containing an identification number and the names of the servers in the group.

Getting Volume Information

The cfs provides information on volumes. cfs can only be used on a machine which has a running venus (such as a client workstation). cfs is described in Chapter 1 as well as in the manual page.

Chapter 10. User Administration

As of 5.2.x the user and group administration tools and databases have been replaced. Now, instead of editing `user.coda` and `group.coda`, and then converting them with **pwdtopdb** and **pcfgen** has been replaced by the new pdbtool program.

Short introduction to pdbtool

The pdbtool is an interactive command to manipulate and query user and group information. The most commonly used commands in the pdbtool are:

- **nui** {username} {userid}

  – create a new user, with the specified id.

- **ng** {groupname} {ownerid}

  – create a new group, with the specified owner.

- **ci** {user/groupname} {newid}

  – change the id of an existing user or group.

- **ag** {group-id} {user/groupid}

  – add a user/group to a group.

- **n** {user/groupname}

  – list all information about the user or group.

Note: User-ids are supposed to be positive integers, group-ids are negative integers.

For more information read the manualpage (pdbtool(8)), which is included in the `coda-doc-`*`version`* rpms and tarball, or simply play around with the pdbtool's **help** command.

Adding a new user

This is a step-by-step example of the administrative steps involved in adding a new user to Coda. Here we assume that the host named scm is one that has the read-write copy of the databases. The new user is going to be added to the groups Users and Developers.

```
root@scm# pdbtool
pdbtool> nui jan 768
pdbtool> n jan
USER jan
  *  id: 768
  *  belongs to no groups
  *  cps: [ 768 ]
  *  owns no groups
pdbtool> n Users
GROUP Users OWNED BY System
  *  id: -221
  *  owner id: 777
  *  belongs to no groups
  *  cps: [ -221 ]
  *  has members: [ 22 178 184 303 545 697 822 823 835 894 712 738
  *  777 901 902 ]
pdbtool> n Developers
GROUP Developers OWNED BY System
  *  id: -225
  *  owner id: 777
  *  belongs to no groups
  *  cps: [ -225 ]
  *  has members: [ 122 835 ]
pdbtool> ag -221 768
pdbtool> ag -225 768
pdbtool> n jan
USER jan
  *  id: 768
  *  belongs to groups: [ -221 -225 ]
  *  cps: [ -221 -225 768 ]
```

```
  *  owns no groups
pdbtool> q
```

This sequence has created the new user account, and added the account to the appropriate groups. Now in order to activate the account, we need to set an initial password with the authentication server.

```
admin@anymachine$ au -h scm nu
Your Vice Name: codaadmin
Your Vice Password: ********
New User Name: jan
New User Password: newpassword
```

To finish up, we can create a home volume, mount it, set the ACLs, and the user is set up. So all the user needs to do is change his password.

```
root@scm# createvol_rep users:jan E0000100 /vicepa
admin@anymachine$ cfs mkm /coda/usr/jan users:jan
admin@anymachine$ cfs sa /coda/usr/jan jan all

jan@anymachine$ cpasswd -h scm
```

Upgrading existing user.coda and group.coda databases

To upgrade the user and group databases to the new format, a script called pwdtopdbtool.py has been added to the distribution. You might need to alter the pathnames of `/vice/db/user.coda` and `/vice/db/group.coda`. Simply run this script on the SCM and it will parse out all entries from the old user and group files, and uses the pdbtool to create the new `prot_users.db` and `prot_index.db` files in `/vice/db`.

To distribute these files to the other servers, you should add the following lines to `/vice/db/files`:

```
prot_users.db
prot_index.db
```

As long as there are pre-5.2 servers in the group, it is advised to stick to the old way of adding users, and use the pwdtopdbtool.py script to keep the new files in sync.

Upgrading from the coda.pdb/name.pdb databases used in 5.2.0/5.2.2

The initial release using the pdbtools used gdbm as the underlying data store. However we were unable to

reliably use the resulting databases across heterogeneous servers. So we decided to switch to libdb 1.85.

There is no script available to automatically upgrade your gdbm databases to libdb 1.85 format. If you still have existing `users.coda`/`groups.coda`, simply run pwdtopdbtool.py.

Otherwise, you must extract all information using the old pdbtool *before* upgrading your SCM.

```
# echo list | pdbtool > full_pdb_dump
```

However, it is not possible to easily use this dump to recreate the database. Everything needs to be re-added to the new pdb database by hand.

Chapter 11. The Backup System

Introduction: Design of the Coda Backup Subsystem

As the use of the Coda file system increased, the need for a reliable backup storage system with a large capacity and a minimal loss of service became apparent. A one operation backup system was determined to be infeasible given the volume of data in Coda, the nature of a distributed file-system, and the long down-time that would normally be required to backup the system in one operation.

In order to meet the goals of high availability and reliability inherent in Coda design and to make efficient use of backup hardware and materials, the volume was chosen as the unit of data, and 24 hours was chosen as the time unit for system management and administration. The result of these design considerations is a volume by volume backup mechanism that occurs in three phases:

Cloning. The cloning phase consists of freezing the (replicated) volume, creating a read-only clone of each of the replicas, and then unfreezing the volume. This allows mutating operations on the replicated volume to occur while maintaining a snapshot to backup. Once the cloning phase has been completed, normal read-write services can be resumed without fear of data corruption due to mutating operations on an active file system.

Dumping to disk files on a backup spool machine. The dumping phase consists of converting the read-only volume clones to disk images stored as regular disk files, on a spool machine. A dump can either be full (level 0), in which all files are dumped; or incremental (level 1 through 9), in which only those files or directories which have changed since the last successful backup of the lower level are included in the dump. So a level 1 dump will include changes from last full dump, a level 2 - from last level 1 dump, and so on. This allows for a system in which only a subset of volumes need a full backup at anyone time (with incremental backups done between full backups), thus reducing the amount of off-line storage and network bandwidth needed at any one time. However, it allows the re-creation of data a granularity of 24 hours when combined with incremental dumps. Incremental dumps, however, are only supported for replicated volumes. Since there is little need for non-replicated volumes, only full dumps are supported for non-replicated volumes.

Saving to media. The last phase consists of writing all the dump files from the backed up volumes dumped on local partitions to an archival media such as tape. Any standard backup system can be used for this phase. At CMU, we use the BSD dump and restore utilities to write and retrieve the disk images of Coda volumes to tape.

Practically, this system has been implemented as series of tasks. The first two tasks are carried out by the backup program the latter by a Coda independent Perl script (tape.pl).

1. Backup

   a. create a read-only clone
   b. dumping the read-only clone to a local disk
   c. backing up the dumped data to a suitable archive media

2. Restore

   a. Retrieving appropriate full and incremental dumps from the archive media.
   b. Merging the full and incremental dumps to the time line of restoration.
   c. restoring the fully integrated backup to the Coda file system.

Remember, in practice, many restores are a result of a user accidently deleting or corrupting their own files. In this case, users may use the the **cfs** mechanism to retrieve files for the last 24 hour time line. For example: **cfs** `mkmount OldFiles u.hmpierce.0.backup`, will mount the hmpierce's user backup volume from replica 0 to the `OldFiles` mount point. The file can then be copied out (backup volumesare read-only). Only if restoration needs are older than 24-hours or some catastrophic event outside of the users control occurs do restores from tape normally need be done

Several tools have been developed to help in the creation, analysis, and restoration of data backups. Some of these tools have been developed by the Coda team (those tools concerning Coda FS to local disk conversion) such as backup and tape.pl (used to coordinate the efforts of backup, dump, etc), others employ off-the-shelf software such as the traditional UNIX dump or tar to transfer the disk images created from the dump phase to the backup media. Coda, however, provides a Perl script front-end to dump.

Installing a Coda Backup Coordinator Machine

The Coda Backup coordinator should be a trusted machine. It should be able get all the files that exist in the `/vice` subtree on the servers, although it is not necessary to run a file-server on the backup coordinator (nor is it recommended that the backup machine be a file-server).

Assuming the Backup Coordinator has been setup with an appropriate operating system, the steps are as follows:

1. On the Coda File Server designated as the SCM, verify that `/vice/db/dumplist` contains correct backup schedule for your volumes. If you were using createvol_rep for volume creation, you should have been prompted about the backup schedule for new volume. The `dumplist` contains three fields: volume id specified as its hex value, the full/incremental backup schedule, and a comment which is generally the human readable volume name. For example:

```
7f000001    IFIIIII     s:coda
7f000002    IIIIIFI     u:satya
7f000003    0325476       u:jan
```

The first column specifies the volume id to be backed up, the second column specifies the backup schedule by the day of the week beginning Sunday, and the third column is a comment, usually the volume name in human readable form. So, the volume id `7f000001` is scheduled for a *F*ull backup every Monday, and a level 1 *I*ncremental - Tuesday through Sunday and from the comment, we know this is a system volume called `coda`. Likewise, the second volume `7f000002` is scheduled for a Full backup on Friday with a level 1 incremental being done Saturday through Thursday and is a user volume called `satya`. The last entry shows that it is also possible to numerically identify which dumplevels are to be made on each day.

2. On the SCM, modify `/vice/db/vicetab` to indicate which host is acting as the backup coordinator and which partitions on the backup server are to be used by the backup coordinator to store the dump files. On a triply replicated system, `vicetab` might look like this:

```
tye             /vicepa         ftree  width=8,depth=5
taverner        /vicepa         ftree  width=8,depth=5
tallis          /vicepa         ftree  width=8,depth=5
dvorak          /backup1        backup
dvorak          /backup2        backup
dvorak          /backup3        backup
```

`vicetab`, in addition to listing information on the servers providing replicated data, must also include information on the backup coordinator with backup coordinator's name in the first column, the backup partitions in the second column, and the designation `backup` in the third column. The 4th column is not used for the backup sub-system. Please see the vicetab(5) man page for additional information.

The number of partitions available for dumping may be controlled by the system administrator. Because the volume of data may be both large and variable, the backup program intelligently decides where to store individual dump files based on size across the specified backup partitions. The directories in the sample `vicetab`, are assumed to be separate local disk partitions. An organized central symbolic link tree is created by the backup.sh script in the directory `/backup` that points to the actual files scattered across the `/backup1`, `/backup2`, and `/backup3` given in this example.

3. On the Backup Coordinator, the backup binaries and shell scripts need to be installed. The following directories should be have been created under `/vice` upon the installation of the Coda backup package:

   /backup
   /vice/backup
   /vice/backuplogs

/vice/db
/vice/vol
/vice/lib
/vice/bin
/vice/spool
/vice/srv

The primary binaries that should be installed under /vice/bin to get started are:

backup
backup.sh
bldvldb.sh
merge
updateclnt
updatesrv
updfetch
tape.pl

Once it has been verified that the backup system is installed, the files `/vice/db/hosts`, `/vice/db/files` should be manually copied from the SCM to the same location on the Backup Coordinator. These are needed the first time by the updateclt daemon the when it runs. Also, Coda currently relies on the BSD dump and restore command to manipulate the tapes. A copy of dump package should be installed on the backup coordinator. BSD dump is available for all UNIX and UNIX like operating systems we have successfully run Coda on. Please check with your OS Vendor if you need help obtaining a copy.

Upon completion backup will print which volumes were successfully backed up, the volumes on which backup failed, and those volumes which were not specified for backup.

The merge program allows a system administrator to update the state seen in a full dump by the partial state in an incremental dump. This is useful, when a user wishes to restore to a state that was captured by a full and some number of incremental dumps, for instance, in the middle of the week. merge applies an incremental to a full dump, producing a new full dump file.

An incremental is a partial snapshot *with respect to* the previous dump. The Coda backup facility maintains an order on dumps for a volume. merge will only allow an incremental to be applied to its predecessor in the order. This predecessor may be a full dump or the output of the merge.

Once the administrator has created or retrieved the full dump which contains the desired state of a volume, she can create a read-only copy of that state by using the **volutil** `restore` facility. This volutil command creates a new read-only volume on a server. The new volume can be mounted as any other Coda volume. Regular Unix file operations can then be used to extract the desired old data. The obvious exception is that mutating options will fail on files in a read-only volume.

Incremental Dumps

In every dump (full or incremental) produces a file containing the version vectors and *StoreIds* of every vnode in the volume. These files have names of the form `/vice/backup/`*`groupid`*`.`*`volid`*`.newlist` for replicated volumes and `/vice/backup/`*`volid`*`.newlist` for non-replicated volumes. When the backup coordinator is convinced that the backup of a volume has completed, the `*.newlist` file is renamed to be `*.ancient` via the **volutil** `ancient` call. These files are stored in a human-readable format for convenience.

When creating an incremental dump, the server looks for the .ancient file corresponding to the volume. If it doesnt exist, a full dump is created. If it does exist, it is used to determine which files have changed since the last successful backup. The server iterates through the vnode lists for the volume and the version vector lists from the `*.ancient` file comparing the version vectors and storeIds. A discrepancy between the two implies that the file has changed and should be included in the incremental dump. Since version vectors are not maintained for non-replicated volumes, incremental dumps are not supported for non-replicated volumes by the coda backup facility. By comparing the sequence numbers in the vnodes lists, it can also be determined if a file or directory had been deleted (since the vnode is no longer in use). Vnodes that are freed and then reallocated between dumps look like vnodes which have been modified, and so are safely included in the incremental dump.

It is also important to maintain an ordering on the incremental dumps. To correctly restore to a particular day each incremental dump must be applied to the appropriate full dump. To ensure that this happens, each dump is labeled with a uniquifier, and each incremental is labeled with the uniquifier of the dump with respect to which it is taken. During merge, the full dumps uniquifier is compared with the uniquifier of the dump used to create the incremental. If they do not match, the incremental should not be applied to the full dump.

Tape files

Once the dump files have been created, they must be written to tape. This is due to the fact that disk space is usually a limited commodity. The basic mechanism for the writing is the Unix tar(1) facility.

Each tape contains a series of tar files, the first and last of which are labels. The start and end labels are identical, and contain version information, the date the backup was taken, and an index which maps individual dump files into offsets into the tape. Thus the Coda backup tapes are self identifying for easy sanity checks. The label is a tar file which only contains a simple Unix file called `TAPELABEL`.

The dump files are first sequenced by size. They are then broken down into groups, where the total size of the group must be larger than a certain size, currently .5 Megabytes. Each group is stored in a single tar file on the tape. These data tar files are the 2nd through n-1st records on the tape, the first and nth being a tar files containing just the tape label.

This structure was chosen for several reasons. The first is that it is easy to implement. TAR has been used for many years, and has been proven to be reliable. The second is easy access of information on the tape. Using a single monolithic tar file would often require hours of waiting to retrieve a single dump file. This way you can skip over most of the data using mt(1) and its fast-forward facility. Finally, it provides a simple and effective end-to-end check to validate that all the information has made it to tape.

At CMU, we have created a convention for capturing sufficient information for reliability, while trying to avoid excess use of tapes. Full backups are taken once a week. However, since our staging disks are not large enough to hold full dumps for all the replicas of all the volumes, we stagger the full backups across the week.

There are three kinds of requests for restorations: users who have mistakenly trashed a file, users who lost data but didnt know it, or bugs which require us to roll back to a substantially earlier state. The first class of restores can be typically handled by yesterdays state, which we keep on-line in the form of read-only backup clones. Thus almost all forms of requests never reach the system administrator at all. To give users easy access to the previous days backup, create a directory, `OldFiles`, in their `coda` directory, and mount each of the backups in the `OldFiles` directory.

If the user did not catch the loss of data immediately, its reasonable to expect that they will catch it before a week has passed. We keep all incremental and full backups to guarantee we can restore state from any day in the last week. This requires 14 tapes, or two weekly sets. One weeks worth is not sufficient, because state from later incremental relies on earlier incremental backups at a lower dumplevel in order to be restored. Thus as soon as the first incremental tape is over-written (say Mondays), the state from the remainder of the last week is lost (last Tuesdays, Wednesdays, etc).

The third class of data loss is either due to infrequently used files or to catastrophe. (We've actually been forced to rely on the backup system to restore *all* Coda state, while developing server software). Since its unreasonable to keep all the tapes around, we only save tapes containing full dumps. Weekly tapes are saved for a month, and monthly tapes are saved for eternity.

Restoring a backup clone

A basic assumption of performing backups is that eventually someone will need to restore old state of a volume. To do this they should contact the system administrator, specifying the volume (groupid and repid for replicated volumes or just the volid for non-replicated volumes) and the date of the state they wish to restore.

The system administrator must then determine which dump files contain the state. There could be more than one involved since the state may have been captured by a full and some incremental dumps. Once the administrator knows the dates of the backups involved, she must get the appropriate tapes and extract the dump files (via the extract.sh script).

The administrator then creates the full state to be restored by iteratively applying the incremental backups to the full state via the merge program. Once the state for the date in question has been restored, a read-only clone is created by choosing a server to hold the clone, and invoking the **volutil** `restore` operation, directing the call to the chosen server. Once the clone has been restored, the administrator should build a new VLDB, and mount the volume in the Coda name space so the user can access it. When the user has finished with it, she should notify the administrator in order for the clone to be purged.

Backup Scripts

Although the backup program handles all the tricky details involved in Coda backup, there still remains some issues to be handled, most notably the saving of the dump files to tape. This is done by a series of scripts, backup.sh, and tape.pl.

backup.sh takes the name of the directory in which to run backups. It creates a subdirectory who's name indicates the date that backup was run. It then runs the backup program, using the dumplist file in the directory specified in the arguments, saving the output of backup in a logfile in the newly created subdirectory. It copies in the current Coda databases (so they will be saved to tape along with the dump files). It then invokes tape.pl to write and verify that the files have been safely recorded.

tape.pl performs the work of saving the files on tape. It takes the directory in which the backup was taken (the subdirectory generated by backup.sh), and the device name of the tape drive. It first checks to see that the tape to be used is either empty or has the correct label. For Coda at CMU this means checking that the tape was last used on the same day of the week. It then gathers the dump files and databases into groups and generates the tape label for this backup. Finally it writes the tape label and all the groups to the tape via the dump(1) facility, marking the end of the tape with another copy of the tape label.

tape.pl then verifies that it did its job correctly. It first reads off the tape label, comparing it with one stored in the backup directory. It then scans all the dump files, listing the contents of the dumps and the commandline parameters for restore to stdout which has been redirected by backup.sh to the backup logfile.

restore(1) is used to extract a dump file from a Coda backup tape. The backuplogs contain the commandline that is needed to jump to the correct part of the tape.

Chapter 12. Reinitializing Coda Servers after a Disaster

As Coda is still an experimental system, the state of a Coda File System may sometimes get to the point where you will want to reinitialize all of the servers.

> **Note:** This is a fairly drastic step and should be treated as a last resort. It will typically take an entire day or more to perform and the system will be unavailable during that time. You will *not* be able to reintegrate after a re-initialization.

This chapter explains the steps necessary to reinitialize your system and restore your data on "clean" servers. The basic steps are as follows:

1. Obtain backup dump files for all of the volumes in the system.
2. Reinitialize RVM on the servers.
3. Recreate the volumes that existed before the re-initialization.
4. Restore and mount the backup clones created in step 1.
5. Mount the volumes created in step 3.
6. Copy the data from the restored volumes to the newly created volumes.
7. Unmount the new volumes and backup clones.
8. Flush client caches.
9. Test the system.

Obtaining Backup Dump Files

If the system is in a state that backups can be successfully run, we suggest running the **backup.sh** script to capture the most recent state. You may, however, use the tapes from your last successful backup and skip this step. See Chapter 11 and backup(8) for more details on creating backups. You will need to create new `dumplist` files for the backups. If you include the volumes that you do not normally back up in the newly created dumplists, you will be able to restore those volumes rather than re-create their contents. See dumplist(5) for details on the format of these files. To create the dumplists, use the following steps:

1. Use `/vice/vol/VolumeList` and `/vice/vol/VRList` to create a list of all of the volumes on the servers you want to reinitialize.
2. Sort the volumes by size. Use the **cfs** `lv` command to get an estimate on the size that each volume will take. Remember that if a volume is replicated, each replica will need space on the tape.
3. Divide the volumes into groups that will fit onto a single backup tape (at CMU this is about 4 GB)
4. Run **backup** on the dumplist files.

Reinitializing the Servers

—Reinitializing RVM on the Servers

Use **rvmutl** and **rdsinit** to initialize RVM. Unless you are reinitializing in order to change the size of the RVM heap, be sure to use the same sizes that you used previously. See the section called *RVM Initialization* in Chapter 7 for the details on RVM initialization process.

—Recreate the Backed Up Volumes

Use createvol(8) and createvol_rep(8) to create all of the volumes that you backed up in the section called *Obtaining Backup Dump Files*.

—Restore the Backup Clones

First, restore the read only backup of the `coda_root` volume. You should have created the new `coda_root` volume in the previous step. This will be automatically be mounted. Next create a directory `/coda/tmp/restore` and use the **volutil** `restore` to restore each of the backup clones to `/coda/tmp/restore`. See the volutil(8) man page for more details on restoring the volumes.

—Mount the Recreated Volumes

Create a directory `/coda/tmp/inited` and mount all of the volumes that were created earlier into this directory.

—Populate the New Volumes

Run the script restorevolumes(8) to restore all of the volumes that were backed up in the section called *Obtaining Backup Dump Files*.

—Unmount the Volumes

Use **cfs** `rmmount` to unmount all of the backup volumes in `/coda/tmp/restored` and the new replicated volumes in `/coda/tmp/inited`. Then remove these directories. See the man page cfs(8) for more details on unmounting volumes.

—Flush Client Caches

You need to flush the client caches as they will not have the correct volume information anymore. Use the **cfs** `flushcache` command to do this.

—Test the system

After youve flushed the client caches the system is ready to run. Do some testing to make sure Coda is functioning correctly and allow the users back into the system.

References

M. Satyanarayanan, “Integrating Security in a Large Distributed System”, *ACM Transactions on Computer Systems*, 7, 3, August, 1989.

M. Satyanarayanan, J. J. Kistler, P. Kumar, M. E. Okasaki, E. H. Siegel, and D. C. Steere, “Coda: A Highly Available File System for a Distributed Workstation Environment”, *IEEE Transactions on Computers*, 39, 4, April, 1990.

M. Satyanarayanan, “Scalable, Secure, and Highly Available Distributed File Access”, *IEEE Computer*, 23, 5, May, 1990.

D. C. Steere, J. J. Kistler, and M. Satyanarayanan, “Coda: A Highly Available File System for a Distributed Workstation Environment”, *Summer Usenix Conference Proceedings*.

M. Satyanarayanan, *RPC2 User Guide and Reference Manual*, October, 1991, School of Computer Science, Carnegie Mellon University.

H. Mashburn and M. Satyanarayanan, *RVM: Recoverable Virtual Memory User Manual*, June, 1992, School of Computer Science, Carnegie Mellon University.

H. Mashburn and M. Satyanarayanan, “Disconnected Operation in the Coda File System”, *ACM Transactions on Computer Systems*, 10, 1, February, 1992.

P. Kumar and M. Satyanarayanan, “Log-based Directory Resolution in the Coda File System”, *Proceedings of the Second International Conference on Parallel and Distributed Information Systems*, January, 1993.

J. J. Kistler, *Disconnected Operation in a Distributed File System*, May, 1993, Computer Science Department, Carnegie Mellon University.

M. Satyanarayanan, J. J. Kistler, P. Kumar, M. E. Okasaki, E. H. Siegel, and D. C. Steere, *Lightweight Recoverable Virtual Memory*, March, 1993, School of Computer Science, Carnegie Mellon University.

Appendix A. Quick Reference

Table A-1. Popular commands

cfs {checkpointml}	**Checkpoint Coda modify log**
cfs {la} [*dir*]	**Display a Coda access control list**
cfs {sa} [*dir*]	**Set a Coda access control list**
clog	**Authenticate to Coda servers**
cpasswd	**Change your Coda password**
ctokens	**List Coda tokens**
hoard	**Set your hoard database**
hoard {list}	**List your hoard database**
repair	**Repair inconsistent files and directories**
spy	**List all Coda files that are referenced**

Appendix B. System Configuration Files

File	Where	Created By	Edited By	Purpose
auth2.pw	/vice/db	initpw	auth2	User password database of in binary format used by auth2.
auth2.tk	/vice/db	vice-setup		Used for secure communication among Coda servers.
dumplist	/vice/db	vice-setup	createvol_rep	Specifies which volumes to backup and when.
files	/vice/db	vice-setup	hand	Update uses the files in files as a list of files to keep up to date.
servers	/vice/db	vice-setup	hand	Server number. Used as part of a volume number.
VLDB	/vice/db	buildvldb.sh		Volume Location Database. Used by clients to locate volumes.
prot_users.db, prot_index.db	/vice/db	pdbtool	pdbtool	User and group database used by Coda file-servers
volutil.tk	/vice/db	vice-setup		Used for secure communication among Coda servers.

`VRDB`	`/vice/db`	volutil, makevrdb	volutil, makevrdb	Map group volume names and numbers from a replicated volume to a VSG and the read-write volumes that make up the replicated volume.
`VSGDB`	`/vice/db`	vice-setup	hand	By createvol_rep to map a VSG to a set of servers
`AllVolumes`	`/vice/vol`	bldvldb.sh		Human readable version of VLDB
`maxgroupid`	`/vice/vol`	vice-setup	createvol_rep	By creatvol_rep when allocating group ids
`partitions`	`/vice/vol`	bldvldb.sh		Lists vice partitions for all servers.
`RWlist`	`/vice/vol`	createvol_rep		Lists all read write volumes.
`VolumeList`	`/vice/vol`	createvol_rep		List of volumes on server
`VRlist`	`/vice/vol`	volutil	volutil	Human readable version of VRDB.
`SrvLog`	`/vice/srv`	codasrv		Log of server activity.
`SrvLog-??`	`/vice/srv`	volutil		Old SrvLog.
`update.tk`	`/vice/db`	vice-setup		Used for securing the communication between the update daemons.
`UpdateClntLog`	`/vice/misc`	updateclnt	updateclnt	Log of updateclntactivity.
`UpdateSrvLog`	`/vice/misc`	updatesrv	updatesrv	Log of updatesrvactivity.

`AuthLog`	`/vice/auth2`	auth2		Log of authentication server activity.
`venus.conf`	Client: `/etc/coda`	venus-setup	hand	Venus configuration file. Several other programs use it to get server names.