Pilot Deployment of Internet Suspend/Resume on the CMU Campus

A Joint Project with Intel, IBM, and CMU

Casey Helfrich, Intel Research
David O'Hallaron, Carnegie Mellon University
Michael Kozuch, Intel Research
David Westfall, Intel Research

Version 0.93 – April 15, 2004

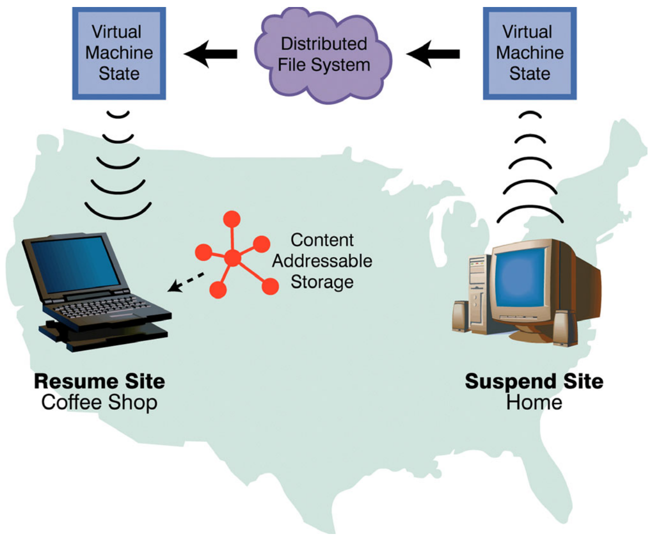


Table of Contents

Section 1: Introduction

IBM, Intel, and Carnegie Mellon are collaborating on a 2-year pilot deployment of the Internet Suspend/Resume (ISR) system on the Carnegie Mellon campus. ISR is a novel computing technology that encapsulates a user's computing environment and enables the management and migration of that environment. The pilot deployment is a proof of concept whose purpose is to help us better understand ISR as a new model of computing. In particular, the deployment team plans to collect data that will improve our understanding of the following four crucial issues:

1. ***Usage issues***. What creative ways do people use ISR in their daily lives? Do people fully exploit the available features of ISR, or only a subset?
2. ***System issues***. How well does the prototype system work? What components need to be improved or replaced? What are the performance bottlenecks? Are users willing to use the functionality provided by an already present distributed file system?
3. ***Resource issues***. What are the resource requirements of an ISR deployment? What is the TCO of an ISR system?
4. ***Management issues***. How does ISR improve/affect our ability to manage and administer systems?

Better understanding of these issues can only be obtained by placing the system in the hands of real users and empirically observing metrics such as system performance, resource consumption, and user satisfaction. Consequently, we have designed a pilot deployment of ISR that will involve a pool of users interacting with the system as part of their day-to-day routine. In return for access to the system, the users will agree to two stipulations: that they will provide periodic feedback regarding their satisfaction with the system, and that the system administrators can collect data regarding the operation of the system.

We have adopted a phased approach. Initially, the user pool will initially consist of a small group of sophisticated users. Over time, the user pool will grow and include less sophisticated computer users. The user pool will include students, faculty, and staff from CMU. After the first year we expect to involve 40 CS graduate students and 5 members of the CMU IT staff. In the second year, we will add approximately 10 members of the CMU financial services staff and another 45 or so more participants drawn from the campus community. Many of these users will be using ISR only on campus, suspending and resuming their environments on different machines on the CMU campus network. Other users will be using ISR between machines on campus and their home machines, which will be connected to the Internet through high-speed broadband links.

This document provides details regarding the ISR deployment including resource requirements, schedule, and milestones. This document also includes descriptions of the service agreement that will be provided to the users as well as the data collection consent form. The ISR architecture and current implementation are included as background material.

Section 2: Overview of ISR

2.1 The ISR Model

Internet Suspend/Resume (ISR) is a new approach to mobile computing in which a user's computing environment follows the user through the Internet as he or she travels. Today, when a laptop computer is closed, the user's execution state is suspended to disk. When the user moves the laptop to a new physical location, the user may re-open the laptop, and resume working in the same environment that was active at the time of suspend. The goal of the ISR project is to achieve the same effect without requiring the user to transport physical hardware.

For example, imagine the scenario in Figure 1 where telecommuter who works from home in the morning and at the office in the afternoon. After completing a morning's work, the user clicks "suspend" on the home machine and begins to travel to the office. While the user is en route, the state of the user's computing environment is also en route, through the network, to the machine in the user's office. When the telecommuter arrives at the office, the office machine is presenting the same environment that the user left at home: the same applications and files are open, the windows are all in the expected places, and the cursor is in the appropriate location.

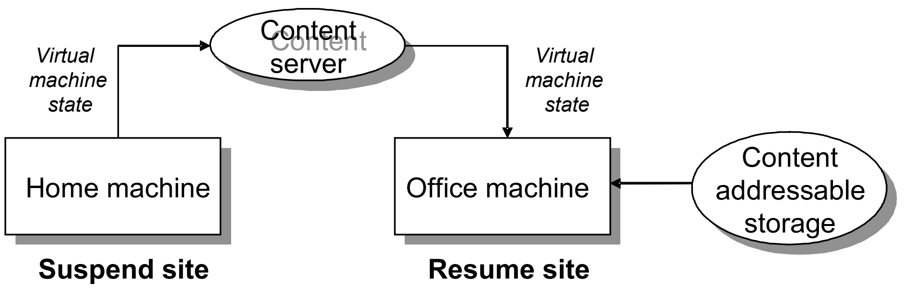


Figure 1: An Internet Suspend/Resume cycle.

The capabilities of ISR are realized through a combination of two well-studied technologies: virtual machine technology and distributed files systems. Virtual machine technology is used to capture the user's computing environment at the time of suspend. This environment consists of the user's operating system, applications, data files, customizations and the current execution state. By leveraging virtual machines, this state is re-instantiated directly on the machine at the resume site. This is not a thin-client approach, all process execution and computation takes place on hardware near the user. Therefore, ISR is able to provide the low-latency interactivity that users have come to expect.

One of the great challenges of ISR is that the state associated with a user's computing environment may be large: tens of gigabytes in size. ISR research at Intel Research Pittsburgh has focused primarily on reducing the time required to move this state through the network. We employ distributed file system technology to manage the transport of this state and have investigated a number of novel techniques for improving the operation of the distributed file system.

For example, the ISR team is currently exploring the augmentation of distributed file systems with Content Addressable Storage (CAS) technology. With CAS technology, users may leverage the storage associated with small devices and nearby computers to improve the performance of the ISR system without sacrificing security or robustness.

In addition to serving mobile computer users, ISR technology may also be of particular interest to Information Technology (IT) departments for medium to large organizations. Leveraging the ISR encapsulation of user state, IT departments may provide enhanced services by centrally managing the computing environments for all the users in the organization. As Internet Suspend/Resume becomes more pervasive, users will find themselves able to access their computing environment from any connected computer.

2.2 ISR Implementation

2.2.1 Architecture and Operation

ISR uses a combination of hardware virtualization and distributed file systems to present a new model for user mobility. The idea is to add a virtual machine layer between the physical hardware and the user's operating system. This allows ISR to completely encapsulate the user's computing environment and isolate it from the hardware. The computing environment can then be moved to a different hardware platform. As long as the virtualization layer presents the same virtual machine interface, the guest OS does not know or care that the physical hardware underneath it has changed. The hardware must also have access to some form of distributed file system, so that the files that represent the user's computing environment can be exported and shared privately.

Figure 2 shows the layered architecture of the current ISR implementation. A commodity *virtual machine monitor* (VMware) supports a collection of *guest machines*, each of which runs their own *guest operating system* and *guest applications*. Guest machines are also referred to as *virtual machines* (VMs). The complete state of each guest machine is encapsulated as a collection of files called a *parcel*. The ISR layer sits between the virtual machine monitor and the *host machine* and its *host operating system,* interposing on certain interactions between the virtual machine monitor and the host operating system, reading and writing files in a distributed file system. For the proof of concept in the pilot deployment, we are using the Coda distributed file system.

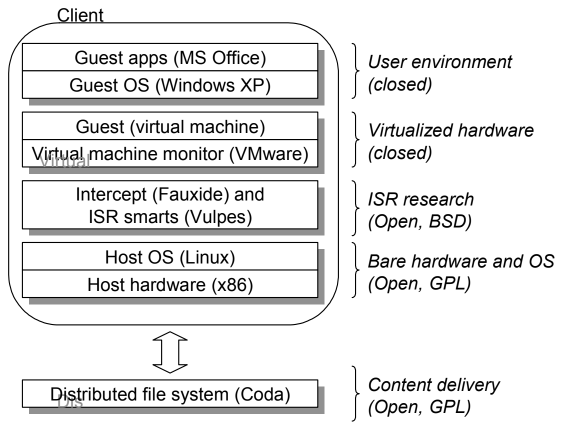


Figure 2: ISR layered architecture.

Figure 3 shows the interactions between these different components in more detail. A loadable kernel module called *Fauxide* serves as the device driver for a Linux pseudo-device called `/dev/hdk`. VMware is configured to use this pseudo-device as its sole virtual disk in “raw” mode. All VMware disk I/O requests to `/dev/hdk` are redirected by Fauxide to a user-level process called *Vulpe*s. It is Vulpes that implements state transfer policy, as well as the mapping of VM state to files in Coda. Vulpes also controls the caching of those files and their encryption and compression (more on this in the next section). Vulpes represents each parcel as a tree of Coda files rather than a single large Coda file. Virtual disk state is divided into 128KB or 256KB (uncompressed) chunks, and each chunk is mapped to a separate Coda file. There files are organized as a two-level directory tree to allow efficient lookup and access of each chunk. Chunking gives Vulpes control over what parts of the virtual machine state are cached, and when modified virtual machine state is sent to servers.

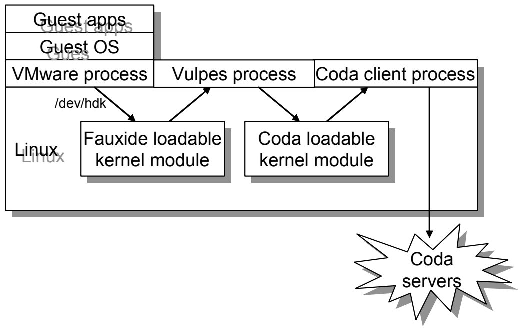


Figure 3: Interactions between the different ISR components

2.2.2 Convergent Encryption

An important feature of ISR is that all files stored on permanent media or transferred over the network are encrypted. This includes files in the local cache of the host machine, and on the remote Coda servers, and on any (optional) Look-aside (LKA) dongles or Content Addressable Storage (CAS) jukeboxes. (More details on LKA and CAS can be found in the Appendix). It is particularly important that these files are encrypted because they can easily contain (in the clear), keyboard buffers, RAM memory images with random segments of memory, guest-OS swap files, `/etc/password` shadow files, etc. When you consider the sensitivity of these files, it becomes obvious that they need to be encrypted.

The solution that ISR uses is a relatively new idea called *convergent encryption*. The idea is to use the actual content of a file to encrypt the file itself. In particular, we compute a SHA-1 hash (20-byte signature) of the original file, and then use that hash as a key for Blowfish encryption of the file. The original SHA-1 hash now becomes a secret key to decrypt the file. We then calculate a new SHA-1 hash of the encrypted file and this, in essence, becomes a public-key that can still uniquely identify the original unencrypted file, but cannot be used to decrypt it. In this way, two identical original files will convergently encrypt to identical encrypted files. This may allow for future enhancements to ISR where a user can query his local network for data that he would otherwise have to fetch over the wide area.

A user keeps a *keyring*, which is a list of SHA-1 tuples *(H1, H2*), where H1 is the SHA-1 of the original file, and H2 is the SHA-1 for the encrypted, compressed file. In effect, H1 is the private key to decrypt the file, while H2 is the public key to identify it. This keyring is kept in memory, and is written to a file that is itself encrypted via the same mechanism during a suspend cycle. The SHA-1 hash for the keyring, which we call the *keyroot,* is the only piece of information that the user needs to keep secret. The keyroot is stored on a special machine in the ISR infrastructure called a *lock server*. At the beginning of a resume operation, the user contacts the lock server and obtains his keyring via an ssh channel.

The primary motivation for using convergent encryption is that it preserves the unique indentifiability of files without exposing their contents. This allows users to take full advantage of performance enhancements such as LKA and CAS.

2.2.3 Compression

Before encryption, ISR compresses the chunk files using standard `gzip` techniques. In general there is a nominal (but noticeable) reduction in file size, but during the compression of the virtualized memory files during a suspend cycle, we see vast improvements. In our experience, a 256MB RAM image typically compresses to around 40MB in the ISR system. It is particularly good that we see such favorable compression on this file, as it is the most crucial file needed to resume the machine at the resume site. This, of course, results in much less data that is required to move through the Internet, which improves resume latency. There are other future technologies, such as ballooning, that may enhance this step even further, but they are currently not included.

2.2.4 Performance

The above technology combined with compression, convergent encryption, and aggressive caching and hording techniques in the distributed file system, provide a nearly transparent experience to the end user. Current benchmarking shows a best-case slowdown of about 5%, and a worst case slowdown of about 30-40% on broadband and LAN networks. There exist certain network conditions where enhancements such as hoarding and look-aside caching will be a necessity for usability of the system.

2.2.5 Operational Description

There are three logical entities in an ISR system: the *client*, *lock server*, and *Coda server*.

Client. The client runs the virtualization software and a local file-system client that does cache management, and services cache misses in the Coda file system. The client also runs ISR software that maps IDE requests from VMware into a two level tree of files in Coda. The client also does aggressive compression and encryption of all data that goes out over a network or hits a real disk, even in the local client cache.

Lock server. The lock server is the logical central point of management for the entire deployment; physically it may be spread out for redundancy and disaster recovery. The lock server is responsible for managing semaphore type locks on all of the parcels it services. The lock server decides which user has access to certain parcels, and instructs the client where to find the appropriate data representing the parcel.

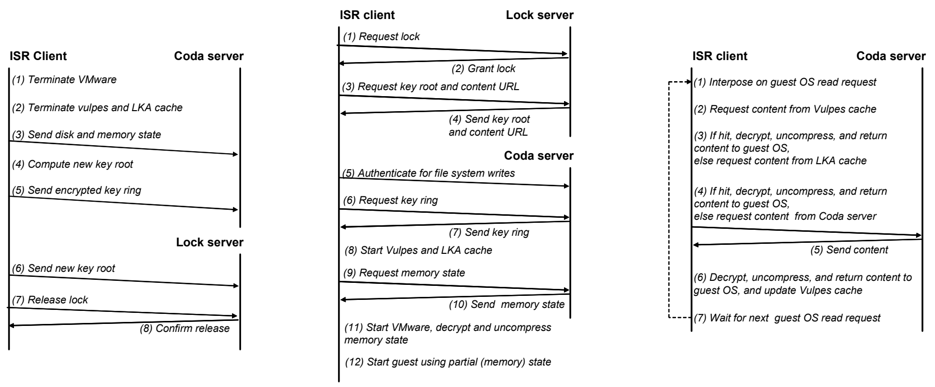
Coda server. After the user authenticates to the lock Server and the memory image for the parcel is bootstrapped into the client machine, the Coda server now acts as a content delivery mechanism over the life of parcel, while it is running on that specific client, that is, until check in. Whenever there is a local cache miss on the client, the Coda server sends the appropriate file, which is now present in the client’s local cache, until eviction.

The *ISR cycle* involves three basic operations: *suspend, resume*, and *run*, which are outlined in Figure 4.

Suspend. The suspend operations involves packaging, compressing and encrypting the parcel’s memory image, syncing all traffic to make sure the host operating system’s kernel IO buffers are emptied, pushing back any pending writes to the Coda server, sending the memory image to the Coda server, and finally contacting the lock server to complete the check-in so the parcel will be available for check-out at other client locations. Anything remaining in the cache is encrypted, and the user also has the option of flushing the cache if they do not plan to return to this location, or it is a public workstation.

Resume. To resume, the client contacts the lock server to get mutually exclusive access to the parcel (the encapsulated virtual machine state) stored on the Coda server, and it also get the keyroot that it will need to decrypt the key ring associated with the parcel. The client than authenticates with the Coda server, requests the keyring, starts the Vulpes process, and does any necessary cache initialization. Next, it requests the memory image of the virtual machine, decrypts and decompresses this image, starts up VMware, and boots the guest VM. At this point the resume is completed, and the guest VM is in the same state it was just prior to the previous suspend operation.

Run. During this step, the guest OS runs its guest applications. Any chunks of the parcel that are not cached on the client are retrieved from the Coda client.


ISR Client
Coda server
(1) Terminate VMware
(2) Terminate vulpes and LKA cache
(3) Send disk and memory state
(4) Compute new key root
(5) Send encrypted key ring
Lock server
(6) Send new key root
(7) Release lock
(8) Confirm release
ISR client
Lock server
(1) Request lock
(2) Grant lock
(3) Request key root and content URL
(4) Send key root and content URL
Coda server
(5) Authenticate for file system writes
(6) Request key ring
(7) Send key ring
(8) Start Vulpes and LKA cache
(9) Request memory state
(10) Send memory state
(11) Start VMware, decrypt and uncompress memory state
(12) Start guest using partial (memory) state
ISR client
Coda server
(1) Interpose on guest OS read request
(2) Request content from Vulpes cache
(3) If hit, decrypt, uncompress, and return content to guest OS, else request content from LKA cache
(4) If hit, decrypt, uncompress, and return content to guest OS, else request content from Coda server
(5) Send content
(6) Decrypt, uncompress, and return content to guest OS, and update Vulpes cache
(7) Wait for next guest OS read request

(a) : Suspend *(b): Resume* *(c): Run*

Figure 4: Operations involved in the ISR process.

Section 3: Pilot Deployment on the CMU Campus

The pilot deployment will consist of 10 to 100 ISR users on the Carnegie Mellon campus over a two year time period. Over its lifetime, the deployment will require a back end server array (description below), approximately 200 laptops and PCs, and the requisite networking to connect the Laptops to the server.

Comment [DFB1]: I'd like to see a section on risk analysis. This section could include subsections on hacking and viruses, legal issues, issues with VMware (e.g., what if the resume site hardware is missing some key peripheral device?) and the like. I'd also like to see some tools and procedures put in place so that the team can collaborate better. Right now we're doing it with email.

Comment [CJH2]: These are topics that Satya, Mark Alexander, Kevin Rusche, et. al. should be tackling. We have a VMware license for each workstation, and have purchased a site license for MS Windows and Office Suite for each system. These particular licensing issues should not be a problem, but we have a legal team verifying this.

3.1 Usage Models

We can characterize the different ways that a person might use ISR as a 2D space where one dimension describes parcels and the other dimension describes hosts. Table 1 shows one representation of this space.

Parcels/Hosts	***WL***	***WLD***	***WLD-HL***	***WLD-HLD***	***WA***	***WA-HA***	***A***
1S-1P		Non-technical users w/o DSL	Non-technical users w/DSL				
1S-2P		Non-technical users w/o DSL	Non-technical users w/DSL				
2S-1P		Technical users w/DSL	Technical users w/DSL				
mS-nP		Technical users w/DSL	Technical users w/DSL				

Table 1: ISR usage models from the point of view of a single user. *Likely CMU users for the most interesting feasible usage models are listed in the table. Guest legend:* ***1S-1P*** *(one guest OS with one personality),* ***1S-2P*** *(one guest OS with two personalities),* ***2S-1P*** *(two guest OSs with one personality for each OS),* ***mS-nP*** *(arbitrary number of guest OSs with arbitrary personalities). Hosts legend:* ***W*** *(workplace),* ***H*** *(home),* ***L*** *(laptop),* ***D*** *(desktop),* ***A*** *(all hosts).*

In this table, the vertical axis shows a range of possibilities for defining and using parcels, with the complexity roughly increasing as we move down the page. Similarly, the horizontal axis shows a range of possibilities for the number, types, and connectivity of the hosts that run the parcels, with the ubiquity of ISR increasing as we move from left to right on the page.

Our model assumes that people will use parcels to run different guest OSs, different personalities of the same OS, or both. Here are the possibilities we think are most likely:

1. ***1S-1P*** *(one guest OS with one personality).* Non-technical users (such as CMU financial analysts) typically use a single guest such as Windows.
2. ***1S-2P*** *(one guest OS with two personalities).* Here, the user maintains a couple of different personalities of the same guest OS, for example, home and work versions of their Windows machine. Again, a likely candidate is a non-technical person who uses Windows exclusively
3. ***2S-1P*** *(two guest OSs with one personality for each OS).* Technical users such as CS faculty and grad students often work with two computers: (1) a Windows laptop for making presentations and reading email during meetings instead of paying attention to the meeting, and (2) a Linux desktop for everything else. Users with two machines have to figure out some kind of scheme for partitioning work and files across these machines. ISR could provide users with access to either of their machines, regardless of which actual machine they happened to be working on at the time. For example, if I had my laptop on a trip, I could use it to access my desktop.
4. ***mS-nP*** *(arbitrary number of guest OSs with arbitrary).* In the most general case, users are running multiple personalities of multiple operating systems. We expect the study to help us understand these scenarios better.

The other axis in our space of usage models is the number, type and connectivity of the hosts that a user has available to run parcels. In the ideal case, every host that a user might ever encounter is an ISR machine. Of course we don't have the resources to implement this in the pilot study. Here is a range of other possibilities for our pilot study:

1. ***WL*** *(workplace laptop).* In this scenario, each user gets a single ISR laptop to use on the CMU campus. This is the easiest solution for the deployment team. Advantages: Users get the benefits of virtual machines (multiple OSs and personalities), as well as the backup, rollback, and discard capabilities of ISR. Disadvantages: users might suspend every once in a while in order to store backups, but they would have little incentive to ever resume. Summary: This scenario might help the development team work out some bugs, but it wouldn't offer much benefit to our users.
2. ***WLD*** *(workplace laptop and desktop).* In this scenario, we give each user an ISR laptop and install ISR on their workplace desktop machine. As we have already mentioned, most technical users find that they need a Windows laptop and a Linux desktop. When they are in the office, people try to work as much as possible on the desktop, because of the bigger screen, keyboard, and overall horsepower. The laptop is typically used for Windows-specific applications such as PowerPoint. Since both work and files are partitioned across the two machines, people need to physically move around the office as they switch from one machine to the other. The ability to run both (virtual machines) on the desktop at the same time would offer some real convenience, since users could take advantage of the nicer environment on the desktop to do all of their work. Similarly, when they take the laptop to a faculty meeting, they will still have access to their desktop

environment. Summary: while this scenario is clearly more useful than ***WL*** and would create more opportunities for suspending and resuming, it is arguable whether it offers potential pilot users enough new capabilities.

3. ***WLD-HL*** *(workplace laptop and desktop, and home laptop).* In this scenario, we give each pilot user an ISR laptop to use at work and on the road, another ISR laptop to use at home, and we install ISR on their workplace desktop machine. (We don't install ISR on their home desktop because it is probably not a CMU asset.). Advantages: This scenario is compelling to users because it allows them to work on the same machines at work or at home. It will be extremely useful to almost everyone in the CMU community, and it will be the most useful data on suspend/resume workloads, network requirements, and LKA caching and content addressable storage. Further, it generalizes to telecommuter applications in the workplace. Disadvantages: Could place too much strain on the CMU network. Requires LKA caching to work without a hitch. An ISR desktop at home would be more compelling than an ISR laptop. Summary: Without a doubt, this is the killer app for ISR. Summary: WLD-HL is the only truly compelling scenario for the prospective users in the Carnegie Mellon pilot study. In order to have a successful pilot deployment, we must offer a robust implementation of this scenario to our users with DSL.
4. ***WLD-HLD*** *(workplace laptop and desktop, and home laptop and desktop). This is a generalization of* **WLD-HL,** clearly useful, but outside the scope of the deployment because of our reluctance to install ISR on non-CMU assets.
5. ***WA*** *(all workplace hosts).* Outside the scope of the pilot deployment.
6. ***WA-HA*** *(all workplace and home hosts).* Outside the scope of the deployment.
7. ***A*** *(all hosts everywhere).* Outside the scope of the deployment.

Putting it all together, for the purposes of the pilot study, the WLD-HL model is by far the most compelling, both to the pilot users and to the deployment team. We should offer a robust version of this model, with robust LKA caching, as soon as possible, to as many users as possible. We can also offer the WLD model to some our users, but in the opinion of at least one of us (O'Hallaron), a robust WLD-HL model is a necessary condition for a successful deployment.

3.2 User Groups

The deployment will consist of a range of users, starting with 10 and scaling to 100. This will take place between summer of 2004 and summer of 2006. One key note: the description of these users contains attributes that are useful for our data collection and experiments, it should not be considered indicative of requirements of a general ISR user population. Later in the deployment, our goals shift and we become more interested in generating more realistic workloads for corporate users. We wish to measure ways to further reduce the costs of ISR so that it can be applied as a cost effective solution to corporate IT management.

Users will have the following necessary characteristics: They should already be a moderate to heavy computer user (2-4 hours/day). They need to be willing to provide a 10 minute feedback once a week. They need to have a network connection where they normally work (not wireless, at this time, unless Linux wireless support can be shown to be acceptable). The desired characteristics of our users are the following. They should always be trying to extend their computing model and try new modes of operation. They should be interested in inventing new work methods when new tools are made available. Ideally, they should have knowledge of distributed file systems.

The following are the proposed user groups and dates that they will be added to the deployment.

7 ISR and Coda developers (summer '04) [7]. We believe that system developers must use their own systems. Thus the following people (at a minimum) will be using ISR on a daily basis starting in summer 2004: Satya (Intel and CMU), Dave O'Hallaron (CMU), Casey Helfrich (Intel), Mike Kozuch (Intel), Jan Harkes (CMU Coda developer), Babu Pillai (Intel), and Haifeng Yu (Intel). Satya and Dave will have ISR installed on their CMU desktop systems so that they can suspend and resume between CMU and Intel. Casey will have ISR installed on his home DSL system, and Dave Bantz will have ISR on his home cable system and his office in Yorktown Heights.

10 CMU graduate students (summer '04) [17]. During this phase, we will identify any major bugs in the ISR implementation. These users should be considered part of the development team. They are sophisticated computer users who are tolerant of system failures. Desired outcome is to have a stable ISR implementation/infrastructure in place on the Carnegie Mellon campus. We also will refine out instrumentation and subjective data collection methodology. We will also evaluate how feasible it is to allow our users to include their personal desktops as ISR workstations. We will assume strong connectivity at most ISR hosts (WLD), although a subset of the group will begin pushing the (WLD-HL) model using their home DSL connections.

Comment [DFB3]: As testers as well as scenario developers. This is a shakedown period during which the ISR service should be quite stable. Otherwise we will get a bad reputation and be unable to find additional users. There must be a compelling scenario of use at this stage, and I'll be surprised if it is work-work.

Comment [CJH4]: I agree, but we should perhaps move this requirement to the Fall '04 section below. I don't want to set goals that we cannot reach. It should be very stable by the END of this phase.

20 more CMU graduate students (fall '04) [37]. At this point our data collection techniques are generating real usable data. These students should be of the same caliber of the initial ten, but the selection criteria may be relaxed. During this phase we will begin to analyze the work-home model with a few select users. We will study multiple parcels per user, and how people use the system to enhance their computing experience, both in the WLD and WLD-HL model. This is also our first opportunity to study how the ISR server infrastructure will scale as we add significant numbers of users. Desired outcomes are to collect the first data on usage patterns, resource consumption for the work-work and work-home models.

5 elite CMU IT staff (spring '05) [42]. These users will generate different workloads than the grad students to model a general corporate customer. As these IT staffers become more knowledgeable, they may start playing a management role in the ISR deployment. Desired data are to collect more data on the WLD and WLD-HL from this different, but still highly technical user group. They will also likely use the multiple-parcels-per-user function of ISR.

10 Internet Services course graduate students (spring '05) [52]. These users are part of a project course taught by Dave O'Hallaron and will generate internet service models and plans using core ISR functionality. They will allow us to study flash-traffic caused by ISR users having similar suspend-resume schedules. Desired outcomes are getting some improvements or extensions to the system and more ideas about creative uses for the system.

10 CMU financial analysts (fall '05) [62]. This is our first group of true non-technical users. Desired outcome here is to learn how non-technical people use the system, and what kinds of additional requirements such users have. These users should have a greater demand for privacy and security than our previous phases.

40 additional users (fall '05) [100]. This gives us approximately 62 active users by February 2005. Based on measured attributes of the deployment, we will then scale to 100 active users over summer '05 and the rest of the year. Desired outcomes here are to better understand the resource and management requirements of a larger scale deployment.

3.3 Finding Users

Dave O'Hallaron and Satya will recruit graduate student users through a series of seminars held on the CMU campus at the beginning of each term (including summer). Mark Poepping (and possibly Bob Cosgrove in SCS) will recruit the elite IT staff members in spring 2005. Dave O'Hallaron will recruit students for his grad class using posters and personal contacts. Mark Poepping will coordinate the recruitment of the 10 CMU Financial Analysts in fall '05.

3.4 Relationship with Users

The relationship with each ISR study user is based on the premise that the study is a two-way street, where both the user and the deployment team get something of value. Given that premise, here is the agreement we will make with each user:

Comment [DFB5]: Is this a 24x7 operation? Do users get a portal? What are our quality-of-service guarantees? What are the procedures for break/fix? Do we need a trouble-ticket management system (even a manual one)? When do we take the system down for maintenance and upgrades?

Comment [CJH6]: I would love suggestions on these topics. This should be a discussion point for our meeting on Monday.

Each user in the ISR study will receive an IBM laptop computer. This is not a gift, but rather a loan with generous terms. In return for the use of the laptop, we ask the users to do the following:

1. Abide by Carnegie Mellon computing guidelines.
2. Use the ISR system on a daily basis, agreeing not to uninstall or bypass the ISR system.
3. Provide personal feedback periodically.
4. Report any problems as they crop up.

Users can keep their laptops as long as they satisfy these four conditions. If they stop satisfying any of these four conditions, then they will be asked to return the laptops. Before participating in the study, each user will sign a consent form that outlines this agreement.

In return, here is what we will provide for our users.

Comment [DFB7]: We should address the question of how a user migrates from their current personal computing environment to ISR, and how they migrate back. Giving them their files back leaves them with an ugly job, one that many will not be able to do.

Comment [CJH8]: A cleaner migration path does not currently exist, we will need to discuss this on Monday.

1. Given a system failure, we will provide recovery of a user's parcel from the last successful suspend point. The ISR backup/rollback system will be limited to use by system administrators for now, but as this becomes more robust, it may migrate to being a user feature.
2. If a user's parcel is damaged either by user behavior, or system instability, they should be able to place a call to an ISR administrator (during normal working hours), who will rollback their machine to the last suspend point, and manually release the lock on the parcel, so the user can then re-resume and continue working. As this feature becomes more robust and user friendly, it will be built into the user interface as a standard option during resume.
3. Initially, the primary points of contact will be Casey Helfrich and David O'Hallaron. We will have secondary support on Coda issues from Jan Harkes of CMU, and support on Vulpes and Fauxide issues from Michael Kozuch, of Intel Corporation. This organization should work through phases one and two, until December 2004, at this point the deployment will have grown to a point where Casey and David's bandwidth may be saturated, and other solutions, such as a CMU help desk, will need to be explored.
4. If for any reason, a user leaves the study, we promise to return any personal files stored in their parcels in a form that they can use. For example, we might save all of their personal files in a protected AFS directory.

3.5 Data Collection

A major goal of the study is to get data that will help us improve the ISR design and to provide evidence of improved TCO. We will also use this data to target areas of ISR that can be tuned to better match the usage patterns of real users. The following data will help us in these goals. Following each item is a description of how we will gather the data, and if it is instrumentation, a note if code still needs to be written to collect this data.

Comment [DFB9]: The category of management of the ISR infrastructure does not seem to be represented. We will need to know what tasks administrators perform, how they are performed, how long they take and what tools they need (especially tools they don't have).

Comment [CJH10]: Once the infrastructure is up and running, the CMU operators should not have to do anything, unless there is a hardware failure. Dave O'Hallaron and I will be able to manage these machines remotely.

- System acceptance
  - How many users have agreed to use the system? (instrumentation)
  - How many are actually using it? (instrumentation)
  - Change from last month? (instrumentation)
  - Can we characterize the users? (manual)
  - Are users interacting with the host? (manual)
- Subjective survey results

  - How useful is the system? Needs met? (manual)
  - User satisfaction: functional and performance (manual)
  - Could user do what they wanted/expected? (manual)
  - If a user leaves the deployment, why did they choose to do so?
- Use characterization
  - How often are parcels resumed? (instrumentation)
  - How long are resumed parcels active? (instrumentation)
  - How often are conflicts detected? (instrumentation, to be written [tbw])
  - How many parcels per user? (instrumentation)
  - Was guest suspended or shut-down? Rebooted? (instrumentation, tbw)
  - Seriousness of use: disk activity, CPU cycles, network packets (instrumentation)
  - How often are parcels checkpointed? Rolled back? (instrumentation)
- Robustness
  - How many support calls were fielded? Type? (manual)
  - Is the VMM crashing? The VMs? (manual)
  - Is the ISR software crashing? Coda? (manual)
  - Root cause: network, graphics, user error, etc. (manual)
- Environments
  - What bandwidth is available at resume site? (instrumentation, tbw)
  - Resume machine characteristics? (instrumentation, tbw)
  - Resume locations (IP): fixed and mobile? (instrumentation)
- Disk data patterns
  - How many disk blocks are read / written? (instrumentation)
  - How many chunks are read / written? Dirty? (instrumentation)
  - How many write-before-reads? (instrumentation)
- Memory Image
  - What is the size of the memory image? (instrumentation, tbw)
  - Change over time? (instrumentation, tbw)
  - Delta characteristics? (instrumentation, tbw)
- Features
  - Use of LKA? Hit rate? (instrumentation)
  - Use of hoarding? Effectiveness? (instrumentation, tbw)
  - Venus cache hit rate? (instrumentation, tbw)
  - What are typical demand-miss penalties? (???)
  - Did the client become (write) disconnected? (instrumentation, tbw)
  - Server issues: load, uptime, growth, etc. (instrumentation, tbw)
- Futures
  - LKA/CASPER over the network? Hit rate? (instrumentation, tbw)
  - Number of providers? (instrumentation, tbw)

Comment [DFB11]: I'd really like to see some comparison of suspend/resume and shutdown/restart as usage models.

Comment [CJH12]: We should discuss this on Monday, I do not believe the shutdown/restart model is very compelling (meaning, I don't think it gives much of a benefit given our current implementation), but we will see.

We will also be collecting the following data using existing instrumentation: CPU utilization of servers, network utilization from server perspective, total number of parcels, total number of suspend/resume/rollback events, total dirty state generated between each resume/suspend boundary, total number of users, time that parcels spend checked out vs. checked in.

The empirical data resides on the lock server and should be harvested once a day. For convenience, a program should be written to do this in an automated fashion, resulting in a directory with all of this data for the week. Ideally this raw data will be formatted or (collected directly into) a back-end database such as SQL or DB2.

3.6 Who Gets Access to the Data?

This raw data should be available to all members of the ISR team, from Intel, IBM, and CMU. This, of course, hinges on Human subject testing agreements with CMU, and the ability to make certain data traces anonymous. This should not be a problem, because we only see disk access data traces, not the data itself, but we need to be sure.

Dave O'Hallaron has applied for clearance from the Carnegie Mellon Institutional Review Board (IRB) and expects approval by the end of April.

3.7 Success Criteria

The deployment can be deemed a success if at least four of the below criteria are met:

1. Broad demand for ISR from the CMU constituents, students and staff/professionals) by the end of the first two years.

2. Continued usage of ISR after two years,

3. One core idea from ISR driven from lab test to "real product" – technology transfer.

4. Services (enterprise) model definition by year 1.5 (IBM interest to trigger an internal development plan).

5. Quantifying the TCO.

6. Whitepaper/publishing – at least one joint publication (significant).

7. Having a qualified ISR test bed (Best Known Method).

8. ISR usage spreads beyond the Carnegie Mellon Deployment in a meaningful way, perhaps to the Intel Campus, IBM Watson, or others.

3.8 Server Hardware

One of the goals of the deployment is to determine the number of users a given ISR server can service at a certain QOS. The current design calls for one server blade per 6 users (this will, of course, scale up dramatically). The server-side parts list for this project is the following:

1 – 42U Rack – IBM Netbay Standard Rack
1 – APC 5000RMB Power Source
1 – 7U Blade Center Rack
4 – Redundant 1800W power supplies
1 – FAStT600 RAID 5 Array and controller 14 x 146GB Hard Drives → 1.5TB storage
1 – Fiber-Channel Interface
9 – 2.8GHz Xeon Processor based blades with 2.5GB ECC RAM and 40GB on-board HDD

So what we have here is one IBM Blade chassis that is currently half-populated with blades, but is set up for easy future expandability. All that is necessary to expand the capabilities of the rack is to purchase and insert more blades. The power sources and interfaces are already set up so it should be plug and play, though there may be a learning curve, since this is cutting edge hardware. If additional storage is needed, another FAStT600 RAID controller will have to be purchased, as the current one is completely populated with hard drives.

3.9 Client Hardware

An ISR client machine can currently be a laptop or PC that has at least DSL speed network connectivity, a Linux OS, Fauxide Kernel Module, Vulpes and VMware installed. Ideally we want 100 Mbps connectivity, but it is not strictly necessary. All parcels will be run in 1400x1050 graphic resolution, to assure consistency across suspend/resume boundaries, due to a VMware limitation. This should not be an issue in the future, but it is something to be aware of today.

We have currently secured 30 IBM T41p laptops for Phase one deployment. The clients are configured as follows:

1.7GHz Pentium M Processor based laptops
1GB RAM
60GB 5400RPM HDD
1400x1050 Resolut[illegible]creen
1Gbps integrated Ethernet connection
Integrated 802.11a/b/g IBM Wireless connection
RedHat 9.0 + patches (2.4.20-6 kernel)
Fauxide 1.0+
Vulpes 1.0+
VMware Workstation 4.5+

Comment [DFB13]: We have to have some assurances that we will not be sued by software vendors for violating their license agreements. In particular, although we believe that Windows activation will not be a problem in fact, we don't want Microsoft sending us a "cease-and-desist" order because their OS is migrating between platforms. Whose legal team should be involved?

Comment [CJH14]: Earl Nied of Intel's Legal team will be involved with this process.

3.10 Parcel Setup

Client parcels will initially be setup with 256MB of Virtual RAM (we may scale this to 512MB, depending on performance data). They will also be setup with a 10GB Virtual HDD. Each parcel's "Gold Image", at the time of this writing, takes approximately 4GB of real server-side storage space. As the user fills there hard drive, in the absolute worst case, parcels configured in the above manner will require 11GB of real server-side storage. In general we expect around 6-8GB of server-side storage per parcel per user for average usage.

Additional server-side storage is necessary if we decided to provide rollback functionality for the parcel. The growth of necessary storage space in this case is also bounded. In the absolute worst case (very, very rare) it will grow to n*Total Size of worst case parcel, where n is the number of rollback points you wish to save.

In reality, it will take much less storage space to provide this service, as the worst case assumes that the user is using every byte of their virtual hard drive, and the user completely changes the contents of there ENTIRE hard drive between each rollback point. During phase one of the deployment we will gather much more realistic information for how much storage is actually needed for a given ISR user.

Assuming a QOS of 2 parcels per user, each with 4 rollback points, in the worst case we can provide service for 15 users with our current hardware. Of course, this is again the absolute worst case and will NOT happen.

In the probable, average case, we have the following:

2 parcels with 4 rollback points = 16GB/user → 95 users supported

So with no optimizations to server-side storage mechanisms we are already approaching our goal of 100 users supported by the server-side architecture in the average case.

3.11 Logistics of machine placement

A meeting is scheduled with Carnegie Mellon representatives for Tuesday, April 27th to discuss placement of public ISR workstations.

3.12 Disaster Recovery

Comment [DFB15]: A vital section. Nothing will kill the acceptance of the pilot like a lost parcel. We must have a plan for backing up parcels to an alternate store (CMU?) What happens on an extended outage of the lock server or the Coda server?

Comment [CJH16]: This is a difficult section, especially given lack of funds to implement anything, and I need suggestions from other members of the team.

The current plan is to depend of the Replicated RAID 5 array to provide consistent access to data during the first two phases (until September 1st, 2004) after this point a tape backup, or offsite backup of parcel data and lock server state needs to be set up. Currently there are no funds for this aspect of the project.

The hardware itself is in a complex with 24x7 operators. It has a full scale diesel generator power backup, and climate control. If the room infrastructure goes down, then most likely the entire campus is having an outage, so ISR would be unavailable anyway.

A good first approach to protect against data corruption, which is economically feasible, is to add in one more 1U or 2U server with somewhat large disks to the rack, and have a cron job that grabs parcel locks for each user and copies them onto its disks once a week.

3.13 Hardware Upg and Software Management

Comment [DFB17]: A vital section, which must also include software management. What tools will be used? What are the anticipated key management tasks? Who's going to do the management? How do we patch Linux and VMware on those laptops in user hands? Will it be a depot service model? Lots of important questions here…

Comment [CJH18]: This needs to be discussed on Monday.

To be Determined

Section 4: Appendix A - Explanation of ISR Terms

Parcel

A parcel is a set of files that ISR creates that completely encapsulates and defines your computing environment.

Fauxide

Fauxide (false-IDE) is the kernel level device-driver that presents the system with a fake IDE drive: /dev/hdk. It accepts IDE commands and appears to be a real IDE drive, but it is able to pass these commands up into user space, where Vulpes is waiting.

Vulpes

Vulpes is the user level process that redirects all files requests that Fauxide captures to Coda or some other distribution mechanism. Vulpes also contains the compression/encryption layer in ISR.

Suspend

A user presses a button to "suspend" their machine. This flushes all dirty state back to the infrastructure servers, where it can be optionally backed up, and is made available at any other resume sites. The copy of that state could also optionally be kept on the physical machine in encrypted files, if the user is planning on returning to that machine. This would vastly improve resume latency.

Resume

The act of a user identifying themselves on an ISR enabled computer. The machine then "resumes" and becomes their computer until they suspend.

Suspend Site

A Suspend site is the computer where a user suspends their machine and leaves.

Resume Site

The resume site is one of many, many computers that a user could walk up to and start using. When a user resumes, that computer becomes their computer until they suspend.

Resume Latency

Resume latency is the amount of time from when a user is identified, to when they can start using their machine at a resume site.

Convergent Encryption

Encryption where the key used to encrypt the file is based on the contents of that file. The key is usually some short signature that uniquely identifies that file, and that file alone.

Section 5: Appendix B - Explanation of Coda Terms

RVM: 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. 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. 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.

SCM: System Control Machine

****NOTE: a common misconception about the SCM is that it must always be available for Coda to work. This is not true. The SCM is used for normal administrative tasks such as user and volume creation (and sometimes shares Coda volumes of its own), but it is not necessary for general Coda file-system traffic.

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.

Vice: The File Sharing Program

Vice is the server-side process for Coda. It handles replication, distribution, backup, and consistency issues for the files that it shares.

Venus: The Client-Side File Access Program

Venus is the client side process for Coda. It handles cache management, connection monitoring, and reintegration after disconnection, as well as general file access requests to the server.

Coda Cell or Realm

A Coda cell (or realm) 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 file system, 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 file system Coda servers store files identified by a number typically all under a directory /vicepa. The Meta data (owners, access control lists, version vectors) and directory contents is stored in an RVM data file which would often be a raw disk partition.

Client Data

Client data 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 privileges 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.

Data Location

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

Section 6: Appendix C – ISR Implementation Details

6.1 ISR: File Locations

In ISR, there are three individual computers that we need to be concerned with. These three computers each play a role in ISR and have certain files located on them. These three machines are: LOCKSRV, CODASRV and CLIENT.

LOCKSRV

```
${HOME}/.isr/[parcel]/nonce
${HOME}/.isr/[parcel]/lockholder
${HOME}/.isr/[parcel]/parcel.cfg
${HOME}/.isr/[parcel]/keys/keyroot
```

LOCKSRV is the first machine contacted by the client through a secure ssh channel. When a client machine tries to checkout a parcel, it first creates a semaphore (if it doesn't already exist). If the semaphore exists, then the parcel is already checked out and is unavailable to this client machine.

The semaphore is formed from a timestamp, version number, and a random number read from `/dev/urandom`. This is copied onto the client machine as well. The client then copies the `/keys/keyroot` file, which is the master key to unlock their keyring, which they will read from their parcel directory on CODASRV.

Finally the client will copy the parcel.cfg file, which contains the path to their parcel in Coda, and a version number. The path is of the form:

```
SRC:=CODA://[path]
```

if the parcel is kept in Coda, but it could, for example, be in the form of:

```
SRC:=HTTP://[path] or
SRC:=NFS://[path] or
SRC:=CAS://[path] or
SRC:=HFS://[path]
```

If the parcel is kept on a web server, nfs server, etc...

CODASRV

```
/coda/[realm]/${USER}/isr/[parcel]/hdk/
/coda/[realm]/${USER}/isr/[parcel]/keys/keyring.enc
/coda/[realm]/${USER}/isr/[parcel]/cfg.tgz.enc
```

Now the client machine has enough info to start accessing the parcel itself, which is kept on CODASRV, but mounted on the client, at the above path. The client uses the path provided by the `parcel.cfg` file to access the `/keys/keyring.enc` file in Coda. This file is downloaded to the client directory and decrypted using the keyroot file.

Now the `cfg.tgz.enc` file is downloaded, decrypted and decompressed to become the client's `cfg/` directory. This directory contains various configuration files used by VMware, as well as the `[parcel].vmss` file, which contains the memory image of the virtual machine.

The Virtual machine is launched, and the `/dev/hdk/` directory presented by fauxide is accessed through Coda, and optionally cached in the `isr/[parcel]/hdk/` directory on the client's user directory.

CLIENT

```
${HOME}/.isr/[parcel]/cfg/
${HOME}/.isr/[parcel]/hdk/
${HOME}/.isr/[parcel]/keys/keyring
${HOME}/.isr/[parcel]/keys/keyroot
${HOME}/.isr/[parcel]/nonce
${HOME}/.isr/[parcel]/lockholder
${HOME}/.isr/[parcel]/parcel.cfg
```

Above are the files and directories that will be present on a client machine, when a parcel is checked out and in use.

6.2 ISR: BIN Scripts

In ISR, there are two computers that contain executable scripts that are run by the GUI. These three machines are: CLIENT and LOCKSRV.

LOCKSRV

All scripts are located in: `/usr/local/isr/bin/`.

```
# isr_list_parcels [user]
```

This script simply lists the parcels that exist for a certain user on this LOCKSRV.

```
# isr_cat_parcel_cfg [user] [parcel]
```

This script displays the contents of the parcel.cfg files, which can then be captured by the client.

```
# isr_cat_keyroot [user] [parcel]
```

This script displays the contents of the keyroot file as ASCII, which is then captured by the client.

```
# isr_acquire_lock [user] [parcel]
```

This script creates the semaphore that locks the parcel so it can only be updated from one client at a time.

```
# isr_release_lock [user] [parcel]
```

This script removes the semaphore from the system, making the parcel available to a new resume site.

CLIENT

All scripts are located in: `/usr/local/isr/bin/`.

They all take optional arguments:
`[--parcel {arg} --locksrv {arg} --user {arg} --version --help].`

```
# isr_get_parcel_names
```

Passes your username to LOCKSRV and triggers its `isr_list_parcels` script.

```
# isr_parcel_co
```

Takes your username and parcelname and send them to LOCKSRV, and triggers its `isr_acquire_lock` script, which populates the client's semaphore; then it gets the `parcel.cfg`, and `keys/keyroot` files. Now that the client has the `parcel.cfg` and `keys/keyroot` info, it can decrypt and decompress the `keyring.enc` and `cfg.tgz.enc` files from your parcel source, which it parses from `parcel.cfg`.

```
# isr_parcel_hoard
```

Triggers the client to copy the entire `/hdk` directory from the parcel source to
`${HOME}/.isr/[parcel]/hdk`

```
# isr_parcel_verify
```

Compares the cached versions of the files in the `${HOME}/.isr/[parcel]/hdk` directory to the signatures in the `/keys/keyring` file obtained from LOCKSRV and evicts any old versions.

```
# isr_parcel_run
```

This script makes sure the client has issued an `isr_parcel_checkout` and
`isr_parcel_check_version` and then confirms that the fauxide kernel module is loaded and that vulpes is running. It then launches the machine from the parcel referred to in the clients
`/${HOME}/.isr/[parcel]/cfg/[parcel].vmx` file.

```
# isr_parcel_checkpoint
```

This script causes the client to sync its entire `${HOME}/.isr/[parcel]/hdk` directory to CODASRV, as well as syncing its current `keyring` file, `keyroot` file, and `parcel.cfg` file. It also makes a compressed and encrypted version of the `/cfg` directory and pushes that update to CODASRV. It may also optionally trigger a backup image to be formed.

```
# isr_parcel_ci
```

Issues an `isr_parcel_checkpoint` and then relinquishes its semaphore file, so that another client machine my grab it. This is a complete suspend cycle.

Optional CLIENT commands

```
# isr_connect_lka [path]
```

Assumes an LKA database is mounted at `[path]` and issues an `lka +/[path]` on the client, to add it as a possible LKA source for Coda.

If Coda is not being used as the distribution mechanism, a vulpes level command may be issues to tell vulpes to look-aside at `[path]` as a possible source of data.

Section 7: Appendix D – Building a Coda Server

7.1 Step 1: Server Disk Partitioning

We start with an example of a real partitioning of a server currently in use at Intel Research Pittsburgh, explanations to follow:

Table 1: Example of Partitions for a Coda Server

Partition	Storage Purpose	Mounted	Typical Size	Fscked
sda2	Root and User File System	/	6GB	Yes
sda5	Var File system	/var	100MB	Yes
sda3	Vice File System	/vice	20GB	Yes
sda6	RVM Data and Log	/rvm	2GB	no
sdb1	Coda FS	/vicepa	50GB	Yes

The Coda servers we have set up in the lab have two physical IDE disks, 72GB each. The first 72GB disk is designated at hda and contains the server Operating system (RedHat 9.0 Linux) and control software for coda (vice). The second 72GB disk contains the raw Coda file share, as well as the metadata/control systems for Coda.

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.

During the installation, you will have the opportunity to create additional user accounts; Coda will require a user account that is not root (for security reasons) to have administrative control over Coda.

7.2 Step 2: Install and Patch/Upgrade your Operating System

We assume you will be installing RedHat 9.0 (the latest release when this document was written). If a newer version is available then by all means use this version.

After installing your OS and gaining Internet access, be sure to register with RedHat, by typing `rhn_register` at a command prompt. After registering, type `up2date` at the command prompt to download all of the latest packages available for your system.

One very important thing to note is that there are known problems with the EXT3 journaling in kernel versions below 2.4.9 when combined with Coda on the client side, so you must be sure to upgrade your kernel to a version later than 2.4.20 to be sure this won't be an issue. Unfortunately RedHat is conservative when choosing which kernels to include with their packages, so you will need to get an updated kernel from an outside source, such as www.kernel.org otherwise you will see problems where even simple client commands such as cp will crash and you will need to reboot the client computer.

IMPORTANT!

Coda recently underwent a massive revision, when Coda version 6.0 was released. This necessitated a new kernel module, which is currently not included with standard kernel releases. This may change in the near future, but for now we must compile our own new module. The latest version can always be found at:

http://www.coda.cs.cmu.edu/mirrors.html

The necessary file can be found in the `/pub/coda/linux/src/` directory of the appropriate mirror ftp site. In particular, you are looking for the `linux-coda-6.0.0.tgz` file. This is a tar'd gzip'd source tree for the new Coda kernel module.

Now unpack and compile the kernel module using the following commands: (assumes "#" is your command prompt)

```
# tar xzvf linux-coda-6.0.0.tgz
# cd linux-coda
# ./Configure
# make
```

Make will ask you questions about the current running kernel on your system. Answer appropriately and make will succeed. When this is finished, you will have a shiny new `coda.o` file in the `linux-coda/linux2.4/` directory (assuming you are running a 2.4.x kernel). You now want to use that file to replace the existing coda.o file on your system.

If you look in your `/lib/modules/[kernel-release]/fs/` directory you will see your old `coda.o` module. Note: Unfortunately depending on how you built your system, the module can be located in several (similar) places. You can find the old version of `coda.o` by issuing the following command:

```
# locate coda.o
```

This will display the full path of the location of the old coda kernel module. For the remainder of this document, I will assume it was located in the `/lib/modules/[kernel]/fs/` directory, but you should use the directory appropriate for your system. In order to remove (or rename for backup) this file, you must make sure it is not loaded. To unload, backup the old module, move the new module into place and start that module, run the following commands from the `linux-coda/linux2.4` directory:

```
# rmmod coda.o
# mv /lib/modules/[kernel]/fs/coda.o /lib/modules/[kernel]/fs/coda_OLD.o
# cp coda.o /lib/modules/[kernel]/fs/
# insmod /lib/modules/[kernel]/fs/coda.o
```

Now your kernel is capable of running the latest version of coda.

7.3 Step 3: Download/Unpack/Compile Coda Source

The latest version of the Coda source coda can always be found at:

http://www.coda.cs.cmu.edu/mirrors.html

In order to run the Coda server, you should follow the links from the above page to an appropriate FTP mirror. The required files at the time I wrote this document are as follows, but of course always get the latest versions:

```
lwp-1.10.tar.gz
rpc2-1.19.tar.gz
rvm-1.8.tar.gz
coda-6.0.5.tar.gz
```

All of these files can be found in the `/pub/coda/linux/src/` directory of the appropriate mirror ftp site. One important thing to be aware of is that FTP sites usually show files in lexographic (alphabetical) order, so the latest versions of the RPMs are not always listed at the bottom, since lexographically 5.3.2 comes after 5.3.19, etc... So be sure that you are truly getting the latest versions of the RPMs or there will be incompatibility issues. Once you find the files, save them to the directory of your choice on your computer

Now unpack the tarballs using the following commands: (assumes "#" is your command prompt)

```
# tar zxvf lwp-1.10.tar.gz
# tar zxvf rpc2-1.19.tar.gz
# tar zxvf rvm-1.8.tar.gz
# tar zxvf coda-6.0.5.tar.gz
```

Now it is time to compile and install the various packages. LWP should be installed first, followed by RVM and RPC2 and finally Coda. All of the commands below can be run by a user account, except for make install and make server-install which must be run by root:

```
# cd lwp-1.10
# ./Configure
# make
# make install
# cd ../rpc2-1.19
# ./Configure
# make
# make install
# cd ../rvm-1.8
# ./Configure
# make
# make install
# cd ../coda-6.0.2
# ./Configure
# make
# make server-install
```

At this point a Coda Server is installed on your system, but is not configured or active.

7.4 Step 4: Configure your Coda Server

Now that you have installed Coda, it is time to actually configure it and get it running. Of course this is the hardest part :-) but if you have followed all of the steps above, you should not hit any brick walls. Most importantly I will now assume that you do have an up to date OS, and have mounted the `/vice`, `/vicepa` and `/rvm` directories. The empty files `LOG` and `DATA` should exist in the `/rvm` directory. One last thing to do before we start is to prepare three secret tokens up to 8 characters long (e.g. elephant) that will be used to allow server-server authentication, simply keep these in your head for the time being.

Configuring Coda is either done manually (not recommended) or using the script vice-setup. All of the setup scripts can be found in the `/usr/sbin/` directory of your server, and reading through them can be very informative as long as you have a decent level of Linux experience.

At the command prompt type vice-setup and we are off and running.

`Vice-setup` does many things behind the scenes including setting up the `/vice` directory if it does not exist and populating it with various required subdirectories and files. The script will ask you a series of questions, most of the answers are obvious, and for the most part the default answers that it provides are the only intelligent possible responses. When it asks you if this is the SCM, the answer is yes, since this is the first server, it must be the SCM. As you add servers to your setup, this will not be the case, obviously. When it asks you to supply a server number, I recommend "1" since this is the first server, not rocket science. When it asks you for the uid and name of a user account to be the Coda administrator, a good generic response is "6000" and "codaroot". Here are some of the non-obvious parts of the script:

```
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?
```

This will be the file that we touched earlier to create, so the answer is `/rvm/LOG`

```
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. For the Coda servers in the Intel Research Pittsburgh lab, we are sharing a larger volume, so have a 25M log file.

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

Again, the obvious (and correct) answer is `/rvm/DATA`

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

This specifies 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 fed to `codasrv` when it is started. Note: in newer versions of Coda larger data partitions are supported but this change was not reflected in the `vice-setup` script. In the Intel Research Pittsburgh setup, we have a 500M DATA file. 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;
500M is good for up to 9GB;

```
Proceed, and wipe out old data? [y/N]
```

WARNING if you have entered erroneous entries for RVM log and RVM data partitions, you could damage your system. Be sure you entered correct information!

Once you successfully complete vice-setup you can start your Coda file-server. The end of the `vice-setup` script will give you the instructions on what commands to enter to get your system up and running. We encourage you to finish reading the rest of this document before you issue those commands.

Below is a full transcript of a vice-setup for a Coda server in the Intel Research Pittsburgh lab:

```
[root@isr03 /]# vice-setup Welcome to the Coda Server Setup script!
You already have a file /usr/local/etc/coda/server.conf!
Continuing will remove that file.
Do you want to continue? [yes/no] yes
Setting up config files for a coda server.
Do you want the file /usr/local/etc/coda/server.conf created? [yes]
What is the root directory for your coda server(s)? [/vice]
Setting up /vice.
Directories under /vice are set up.

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

Setting up tokens for authentication.
The following token must be identical on all servers.
Enter a random token for update authentication : elephant
The following token must be identical on all servers.
Enter a random token for auth2 authentication : elephant
The following token must be identical on all servers.
Enter a random token for volutil authentication : elephant
tokens done!

Setting up the file list for update client
Filelist for update ready.
/etc/services already has new services registered! Good.
/etc/services ready for Coda
Now installing files specific to the SCM...

Setting up servers file.
Enter an id for the SCM server. (hostname isr03)
The serverid is a unique number between 0 and 255.
You should avoid 0, 127, and 255.
serverid: 100
done!
Initializing the VSGDB to contain the SCM as E0000100
/vice/db/VSGDB set up

Setting up ROOTVOLUME file
Enter the name of the rootvolume (< 32 chars) : coda.root


Setting up users and groups for Coda
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: 6000
Enter the username of this user: codaroot
```

```
An initial administrative user codaroot (id 6000)
with Coda password "changeme" now exists.

A server needs a small log file or disk partition, preferably on a
disk by itself. It also needs a metadata file or partition of approx
4% of your filespace.

Raw partitions have advantages because we can write to the disk
faster, but we have to load a copy of the complete RVM data
partition into memory. With files we can use a private mmap, which
reduces memory pressure and speeds up server startup by several
orders of magnitude.

Servers with a smaller dataset but heavy write activity will
probably benefit from partitions. Mostly read-only servers with a
large dataset will definitely benefit from an RVM data file. Nobody
has really measured where the breakeven point is, so I cannot
really give any hard numbers.

-------------------------------------------------------
WARNING: you are going to play with your partitions now.
verify all answers you give.
-------------------------------------------------------

WARNING: these choices are not easy to change once you are up and
running.

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

What is your log partition? /rvm/LOG

The log size must be smaller than you log partition. We
recommend not more than 30M log size, and 2M is a good choice.
What is your log size? (enter as e.g. '2M') 30M

What is your data partition (or file)? /rvm/DATA

The data size must be approx 4% of you server file space. We
have templates for servers of approx: 500M, 1G, 2.2G, 3.3G, 8G
(you can store less, but not more on such servers).
The corresponding data sizes are 22M, 44M, 90M, 130M, 315M.
Pick one of the defaults, otherwise I will bail out

Remember that RVM data will have to be mmapped or loaded
into memory, so if anything fails with an error like
RVM_EINTERNAL you might have to add more swap space.

What is the size of you data partition (or file)
[22M, 44M, 90M, 130M, 200M, 315M]: 500M

!!!!!!!!!!!!!!
Your size is an experimental size. Be warned!
You may want to run with private mapping for RVM.


--------------------------------------------------------
WARNING: DATA and LOG partitions are about to be wiped.
--------------------------------------------------------

--- log area: /rvm/LOG, size 30M.
--- data area: /rvm/DATA, size 500M.

Proceed, and wipe out old data? [y/n] y


LOG file has been initialized!


Rdsinit will initialize data and log.
This takes a while.
rvm_initialize succeeded.
```

```
Going to initialize data file to zero, could take awhile.
done.
rds_zap_heap completed successfully.
rvm_terminate succeeded.

RVM setup is done!


Directories on the server will be used to store container files
that hold the actual data of files stored in Coda. Directory
contents as well as metadata will be stored in the RVM segment
that we already configured earlier.

You should only have one container file hierarchy for each disk
partition, otherwise the server will generate incorrect
estimates about the actual amount of exportable disk space.

Where shall we store your file data [/vicepa]?
Shall I set up a vicetab entry for /vicepa (y/n) y
Select the maximum number of files for the server.
[256K, 1M, 2M, 16M]: 16M

Server directory /vicepa is set up!

Congratulations: your configuration is ready...and now
to get going do the following:
 - start the auth2 server as: auth2
 - start rpc2portmap as: rpc2portmap
 - start updatesrv as: updatesrv
 - start updateclnt as: updateclnt -h isr03.research.intel-research.net
 - start the fileserver: startserver &
 - wait until the server is up: tail -f /vice/srv/SrvLog
 - create your root volume: createvol_rep coda.root E0000100 /vicepa
 - setup a client: venus-setup isr03.research.intel-research.net 20000
 - start venus: venus
 - enjoy Coda.
 - for more information see http://www.coda.cs.cmu.edu.
```

Note: During the vice-setup script, you might get an error message saying "`Cannot find librpc2.so.3`", or something similar. To fix this simply run a locate [filename] on the missing files. Then add the directory where they are located to your `/etc/ld.so.conf` file and then run the command `ldconfig` to refresh the list with your new data. Probable locations for these "missing" files are `/usr/local/coda/lib` or `/usr/local/lib`.

7.5 Step 5: Creating your Shared Volumes

Now the server is configured, but now we have to actually create the volumes to share over the network. We must first create the root volume at a starting point. This is one of the commands that are displayed at the end of the vice-setup script. So run those commands in order:

```
# auth2
# rpc2portmap
# updatesrv
# updateclnt -h [SCM name]
# startserver &
```

Wait until the server is up. Check server status by running:

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

Now create your root volume by running:

```
createvol_rep coda.root E0000100 /vicepa
```

These commands are all that are necessary at this time. Now you want to create your user level accounts so you can start providing service to your clients.

7.6 Step 6: Creating your User Accounts

It's time to get familiar with pdbtool. It is a very powerful coda user administration tool. For more information on available commands type pdbtool help at the command line. Here I will go over the required commands to get started.

The following commands should be run as root on your **SCM server machine**:

```
# pdbtool
pdbtool> ng users [group number]
pdbtool> nui [username] [uid]
pdbtool> q
# createvol_rep users:[username] E0000100 /vicepa
```

This is the most minimal setup necessary. You have started pdbtool, and created a new username and an associated used id the uid can be any unused uid on your system. I recommend making your coda users have a uid between 6000 and 7000, since this will guarantee they will be unique and easy to identify. The group number for the "users" group can be something like 5000. The last command actually creates the volume that will hold the new user's filesystem data.

Now we need to set the initial password with the authentication server. You should now go to a **client machine** and log in to Coda as codaroot by using the following command:

```
# clog codaroot@[name of realm]
password: [codaroot password]
```

Now you have to create authentication records for the new user you created in pdbtool above:

```
# au -h [name of SCM] nu
Your Vice Name: codaroot
Your Vice Password: [codaroot password]
New User Name: [newuser]
New User Password: [a generic password]
```

Now you must create a home volume for the new user and add a directory mount point for that volume:

```
# mkdir /coda/[realmname]/users/ (for creation of the first user account)
# cfs mkm /coda/[realmname]/users/[newuser] users:[newuser]
```

Now you need to associate ownership rights of the directory with the new user:

```
# cfs sa /coda/[realmname]/users/[newuser] [newuser] all
```

All of this happens before the client even gets involved (that why you are the administrator). Now we are ready for the real client to connect and change his password to something private. That is done on the new client machine with the following command (after the client is installed and setup and connected):

```
# cpasswd -h [name of SCM] [newuser]
```

The system will then ask for the old password, and then let the user define the new one.

7.7 Step 7: Wrapping it all Up

The server is configured, the user accounts are created, and the server is up and running. Depending on the number of users and the type of work that will be stored on the server(s), you might want to consider a backup system. The great thing is any client computer can act as a backup for the entire system. This is discussed much more thoroughly in our Coda document about the Amanda tape-backup system for Coda available in our Coda document library.

Section 8: Appendix E – Building an ISR Client

8.1 Step 1: Update your Coda Kernel Module

Coda recently underwent a massive revision, when Coda version 6.0 was released. This necessitated a new kernel module, which is currently not included with standard kernel releases. This may change in the near future, but for now we must compile our own new module. The latest version can always be found at:

http://www.coda.cs.cmu.edu/mirrors.html

The necessary file can be found in the `/pub/coda/linux/src/` directory of the appropriate mirror ftp site. In particular, you are looking for the `linux-coda-6.0.0.tgz` file. This is a tar'd gzip'd source tree for the new Coda kernel module.

Now unpack and compile the kernel module using the following commands: (assumes "#" is your command prompt)

```
# tar xzvf linux-coda-6.0.0.tgz
# cd linux-coda
# ./Configure
# make coda.o
# su
# make install
```

Note: during the compilation of `coda.o` you may come across a bug in the file `modversions.h` where the `./Configure` command adds four inappropriate lines to this file, that should be manually removed before moving to the next step. To remedy this problem, open `modversions.h` in a text editor and remove the four identical lines of the form:

[To be determined]

Make will ask you questions about the current running kernel on your system. Answer appropriately and make will succeed. When this is finished, you will have a shiny new `coda.o` file in the `linux-coda/linux2.4/` directory (assuming you are running a 2.4.x kernel). You now want to use that file to replace the existing `coda.o` file on your system.

If you look in your `/lib/modules/[kernel-release]/fs/` directory you will see your old `coda.o` module. Note: Unfortunately depending on how you built your system, the module can be located in several (similar) places. You can find the old version of `coda.o` by issuing the following command:

```
# locate coda.o
```

This will display the full path of the location of the old coda kernel module. For the remainder of this document, I will assume it was located in the `/lib/modules/[kernel]/fs/` directory, but you should use the directory appropriate for your system. In order to remove (or rename for backup) this file, you must make sure it is not loaded. To unload, backup the old module, move the new module into place and start that module, run the following commands from the `linux-coda/linux2.4` directory:

```
# rmmod coda.o
# mv /lib/modules/[kernel]/fs/coda.o
```

```
/lib/modules/[kernel]/fs/coda_OLD.o
# cp coda.o /lib/modules/[kernel]/fs/
# insmod /lib/modules/[kernel]/fs/coda.o
```

Now your kernel is capable of running the latest version of coda.

Note: some Linux installations place the `rmmod` and `insmod` binaries on your system in a directory not generically included in your bash $PATH. You can find these files by issuing a `locate rmmod` command.

8.2 Step 2: Download/Unpack/Compile Coda Client

Assuming that you have a Coda Server setup, it is time to setup your Coda client. The latest version can always be found at:

http://www.coda.cs.cmu.edu/mirrors.html

In order to run the Coda client, you should follow the links from the above page to an appropriate FTP mirror. The required files at the time I wrote this document are as follows, but of course always get the latest versions:

```
lwp-1.10.tar.gz
rpc2-1.19.tar.gz
rvm-1.8.tar.gz
coda-6.0.2.tar.gz
```

All of these files can be found in the `/pub/coda/linux/src/` directory of the appropriate mirror ftp site. One important thing to be aware of is that FTP sites usually show files in lexographic (alphabetical) order, so the latest versions of the RPMs are not always listed at the bottom, since lexographically 5.3.2 comes after 5.3.19, etc... So be sure that you are truly getting the latest versions of the RPMs or there will be incompatibility issues. Once you find the files, save them to the directory of your choice on your computer

Now unpack the tarballs using the following commands: (assumes "#" is your command prompt)

```
# tar zxvf lwp-1.10.tar.gz
# tar zxvf rpc2-1.19.tar.gz
# tar zxvf rvm-1.8.tar.gz
# tar zxvf coda-6.0.2.tar.gz
```

Now it is time to compile and install the various packages. LWP should be installed first, followed by RVM and RPC2 and finally Coda. All of the commands below can be run by a user account, except for make install which must be run by root:

```
# cd lwp-1.10
# ./configure
# make
# make install
# cd ../rpc2-1.19
# ./configure
# make
# make install
# cd ../rvm-1.8
# ./configure
# make
# make install
# cd ../coda-6.0.2
# ./configure
# make
# make client-install
```

At this point a Coda Client is installed on your system, but is not configured or active.

8.3 Step 3: Configure Client and Contact your Coda Server

Now that you have installed Coda, it is time to actually configure it and get it running. Configuring your Coda client is either done manually (not recommended) or using the script `venus-setup`. All of the setup scripts can be found in the `/usr/sbin/` directory of your server, and reading through them can be very informative as long as you have a decent level of Linux experience.

At this point you must know the name of your SCM Server. The following command will configure Coda for your machine:

```
# venus-setup [full_name_of_scm] [cache_size_in_kB]
EXAMPLE: # venus-setup testserver.coda.cs.cmu.edu 80000
```

At this point Coda should pause for a few seconds and then give a confirmation that your system is now configured for Coda. Now you must confirm that your local realm has been configured. Contact your system administrator and find out the name of the realm, and the server names that make up that realm. Then add that info to the `/etc/coda/realms` file on your client computer. You add the info by appending a line of the following form:

```
[realmname]        [space-separated-list-of-servers]
```

After the realms file is properly populated, one last step will get you up and running:

```
# venus &
```

This actually starts your client process. Some info will be displayed on the screen, ending with "`/coda is now mounted`". At this point the filesystem that is shared by your Coda Servers will now show up in the `/coda/[realmname]` directory of your computer. If you wish to view the network activity of your coda client, open up an xterm window and type the command codacon. Codacon will show you all transactions taking place between your Coda Client and Server.

If, when you start venus, you get an error message saying Cannot find `[filename].so.4`, or Cannot find `[filename].h`, simply run a locate [filename] on the missing files. Then add the directory where they are located to your `/etc/ld.so.conf` file and then run the command `ldconfig` to refresh the list with your new data. Probable locations for these "missing" files are `/usr/local/coda/lib` or `/usr/local/lib`.

8.4 Step 4: Authenticate for Write Access

Now you have Coda up and running, but your still can't write files. For this your need to log into your Coda account on the SCM. If you don't have an account, contact your system administrator. You can gain write access to your assigned directories with the following command (note: the ISR scripts automatically complete this step, when you are resuming a parcel):

```
# clog [username]@[realmname]
EXAMPLE: clog isruser@isr
```