Volume 08 Issue 04 Published, November 17, 2004 ISSN 1535-864X

Intel® Technology Journal

Toward The Proactive Enterprise

Enterprise Client Management with Internet Suspend/Resume

Enterprise Client Management with Internet Suspend/Resume

Michael A. Kozuch, Corporate Technology Group, Intel Corporation
Casey J. Helfrich, Corporate Technology Group, Intel Corporation
David O'Hallaron, Carnegie Mellon University
Mahadev Satyanarayanan, Carnegie Mellon University and Intel Corporation

Index words: virtual machine, distributed storage, migration, management

ABSTRACT

Internet Suspend/Resume (ISR) is an exciting new model for managing client machines in the enterprise. ISR provides the administrative benefits of central management without sacrificing the performance benefits of thick-client, personal computing. This capability is made possible through the novel combination of two well-understood technologies: virtual machines and distributed storage management.

With ISR, a user's entire personal computing environment, including the operating system, applications, data files, customizations, and current computing state, is maintained in centralized storage. By leveraging virtual machine technology, this computing environment may be transported through the network and rapidly instantiated on any ISR-enabled client machine. The central management may include automatic backup, virus scanning, and maintenance.

Further, the ISR software stack is naturally partitioned into two parts: the ISR base and the user environment. The ISR base, which comprises the virtual machine monitor and management tools, runs directly on the physical hardware and is centrally managed by the enterprise Information Technology (IT) department. The user environment is the familiar software stack, which comprises the operating system and applications; it may be managed by the end-user, the IT department, or both. This separation enables the user environment to rapidly migrate from physical platform to physical platform to recover from hardware or software failures, for example. This separation also enables the IT department to protect the enterprise network by quarantining badly behaving user environments.

INTRODUCTION

Over the past two decades, the advent of the Personal Computer (PC) has transformed the computing industry. Part of the PC's success in the marketplace is due to its *personal* nature. Individual users have their own hardware resources, manage their own software resources, and are able to customize their computing environment to suit their needs. In an enterprise environment, however, this aspect of personal computing often imposes a maintenance burden on the Information Technology (IT) department that manages the hundreds or thousands of machines that constitute the computing environment of the enterprise.

Internet Suspend/Resume (ISR) is a new technology that improves the manageability of computing environments without sacrificing the personal aspect of modern desktop computing. Moreover, ISR enables centralized-style maintenance of personal computing state while still preserving the performance benefits of thick-client computing.

Personal computing state, in the context of ISR, refers to the user's *entire* computing environment–including the operating system, applications, data files, customizations, and current execution state. The ISR system collects each user's computing environment into a set of files, called a *parcel*, and it maintains these parcels on network servers.

During normal operation, ISR is virtually invisible to the end user. In the context of a corporate campus, for example, as employees prepare to go home in the evening, they *suspend* the operation of their computers by clicking an icon on their desktop. This operation is very similar to closing the lid on a laptop in that the current execution state of the computer is collected. However, an ISR suspend operation also updates the master copy of the parcel, which is stored on a centrally managed ISR server.

Once the user's parcel is updated on the server, the corporate IT department is able to perform common maintenance tasks such as generating a backup copy of the user's environment, or scanning the parcel for virus signatures.

When the employee returns to work the next day, he or she is able to *resume* the execution of his/her computing environment. This operation will instantiate the user's computing environment on that client machine. Execution will resume with precisely the same state that existed at the time of suspend: the correct applications will be open, the user's data files will be open, and the cursor will be in the expected location.

Two properties are essential to the usefulness of ISR as a management tool. First, the IT department administers the personal computing state of every user according to a centralized style. Consequently, the IT department is able to perform traditional centralized maintenance tasks such as backup copy creation on the user parcels while still providing the high-performance end-user experience associated with thick-client computing. Second, the IT department, not the user, administers the low-level ISR software that runs on all client machines.

On an ISR client, the user's environment does not control the hardware directly. Instead, the ISR client program instantiates and supports the user's software. Because the function of this software is limited to supporting the user, or *guest*, software, it is relatively small and simple. Certainly, this program is much less complex that the guest software it supports.

Because the ISR client program is small and simple, it should be less error-prone and easier to maintain than the massive modern operating systems that IT departments currently manage on client machines. Further, the ISR client program does not have to be customized to the guest software that it supports. A single instance of the client program can support many different guest operating systems, for example. Therefore, the ISR client program represents a uniform computing base to the IT department.

Most importantly, the user never modifies the ISR client program. The ISR client architecture effectively divides the client software stack into two parts: the ISR client program and the user environment. The IT department exclusively manages the ISR client program. However, the user environment, which includes the traditional operating system and applications, may be managed either by the user or by the IT department, according to IT policy, just as in non-ISR computing systems.

We expect this division to be the right compromise between user control and IT control in many situations. This organization enables users to modify their software environment while still providing system administrators with a stable, uniform management platform.

ISR DESIGN

When designing the ISR architecture, we developed the following system requirements.

- The user's entire computing environment must be easily managed by the IT department.
- In the event of hardware failure, the user must be able to resume the environment on a new hardware platform that is not necessarily identical to the original.
- The system must enable *checkpointing* of the user's environment, and the user must be able to restore a previously saved checkpoint rapidly.
- Common use scenarios, for example when the user uses the same client platform every day, must perform well and must not impose undue load on the ISR server or network infrastructure.
- The system must support mobile platforms.
- The performance of the system must be comparable to the performance of a traditional (non-ISR) system.

Considering the above requirements, we developed a system design that combines virtual machine technology with network data transport. This combination of two old (at least, by computer science standards) technologies forms a new mechanism for managing personal computing state.

Virtual Machine Technology

Virtual machine technology is a well-understood field of computer science dating to the 1960s [4]. Historically, the term, *Virtual Machine (VM)*, referred to abstract software containers that mimic the operation of physical machines. Low-level software, called the *Virtual Machine Monitor (VMM)*, controls the operation of the VM, which includes virtual versions of the devices found in a physical machine (e.g., processors, memory, and disks). Guest software running within a VM container behaves as if it were running on a physical machine, and assuming that the VMM is sufficiently complete, the guest software will be unable to detect that it is interacting with a software container rather than physical hardware.

In the ISR system, VM technology performs several functions. First, the VM abstraction provides a natural interface through which the ISR system can collect the state of the user's environment. Because the VMM manages the entire operation of the VM, the VMM must also maintain the complete state of the VM. At suspend time, the ISR system simply stores a description of the current VM state into the user's parcel. In particular, the ISR system collects this state at the virtual device level (processor state, memory state, disk state, peripheral state, etc.). Because the state of the environment is captured at the virtual hardware interface rather than some layer within the guest environment, the ISR system need not modify the guest software, nor does it need to be aware of which guest software is included in the user's environment.

The second role performed by the use of VM technology is isolating users' environments from differences in hardware platforms. For example, suppose that a user suspends her environment before leaving work at the end of the day and that, during the night, the user's hard drive crashes. Upon realizing this after arriving in the morning, the user could simply retrieve another ISR client platform from the IT department and resume her parcel on that machine. Resuming on anonymous hardware is possible because (1) the user's entire state is maintained on the server, and (2) the VM interface exported by the ISR client software is identical to the old VM interface, even if the underlying hardware platform is not. The VMM hides differences between the crash and recovery machines, and the user's software is unable to detect that the physical machine has changed.

VM technology provides a third benefit in that the VMM can support operations that manage the client hardware in a context outside that of the user's environment. For example, the VMM might manage the physical network interfaces in the client platform, and when guest software initiates traffic through the virtual network interface, the VMM controls the transfer of that traffic to the physical network device. To detect the infection of guest environments by certain computer worms and viruses, the corporate IT department could provide software in the VMM's virtual network component that monitors network activity originating in the user environment. If that activity resembles the traffic patterns corresponding to known worms or viruses, the VMM could filter the problematic network traffic at the client and outside the context of the infected environment. Naturally, the IT department could also alert the user and/or remotely administer the guest environment.

Network Data Transport

While VM technology satisfies our requirements for state encapsulation, hardware isolation, and client administration, it must be paired with network data transport in order to realize the full promise of ISR. Once the user's environment is captured into a parcel, for example, that parcel must be transmitted to a server for safekeeping.

ISR does not impose any specific requirements on the network transport protocol; any mechanism that efficiently transfers data packets from the client to the server and back should suffice. Example mechanisms might include distributed file systems, http-based transport, and session-based mechanisms such as File Transfer Protocol (FTP) and secure shell (ssh). In fact, while our early work [1] relied on a distributed file system, we have recently included support for an http-based system.

When installing an ISR system, the IT department may freely choose the data transport mechanism. However, the ISR installation team should carefully consider several factors before choosing a particular protocol. For example, most ISR files are tens to hundreds of kilobytes in size, and a few are tens of megabytes in size. Will the protocol efficiently transport medium to large files through the network? Are there any peculiarities (e.g., high latencies or asymmetric links) regarding the intended network that may affect the performance of certain protocols? Does the network include firewall, proxy, or Network Address Translation (NAT) machines?

Another factor that the ISR installation team may wish to consider is the physical location of the ISR servers. Because the ISR system maintains the entire state of each user's environment, it provides an attractive approach for enterprise disaster recovery. Site disasters, such as an office building fire, can be devastating to a business when the information maintained on personal computers is lost in the disaster. An ISR system could help an enterprise recover from a disaster, even when many personal computers are lost, provided that the ISR servers survive the disaster–by being replicated and/or physically separated from the clients they serve. In the same way that ISR can help a single user quickly recover from a hardware failure by resuming on an anonymous platform, ISR can help a hundred users quickly recover from a catastrophe by resuming their environments on a hundred anonymous platforms.

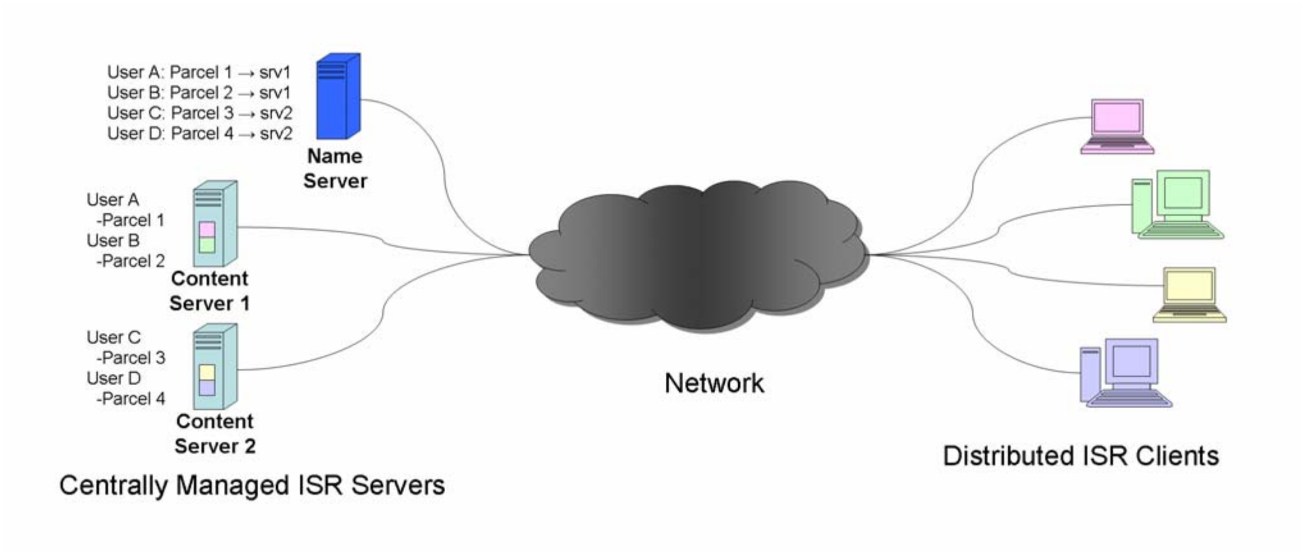


Figure 1: The ISR Network Architecture

SYSTEM ARCHITECTURE

Regardless of the network protocol employed, an ISR system, which is depicted in Figure 1, consists of three logical network entities: clients, nameservers, and content servers. An ISR client is the machine that the user interacts with directly and can be either a desktop or a laptop machine. Content servers manage the data associated with user parcels, and name servers provide a lookup facility through which ISR software can discover which content server is associated with a given user and parcel name. A typical enterprise installation of ISR would include many clients, several content servers, but possibly only one name server.

Client Architecture

Thus far, we have used the terms *resume* and *suspend* loosely to refer to the general operation of the system. In practice, the system includes several more operations and the terms resume and suspend refer specifically to starting and stopping the VM on the client. For example, before a particular environment may be resumed, the client must *check-out* the corresponding parcel from the content server, and after the environment is suspended, the client must *check-in* the parcel. From the user's perspective, the following are the essential ISR operations.

- *checkout* prepares the parcel to be run on the client by obtaining the necessary authentication tokens, decryption keys, software locks, and critical parcel data.
- *resume* resumes the VM and makes the user's environment available.
- *suspend* stops the execution of the VM and saves its current state to the parcel on the client.
- *checkin* saves the current state of the parcel to the content server.
- *discard* deletes the current state of the environment on the client without saving it to the server. This is occasionally useful if the user realizes that something unfortunate occurred during this resume session, such as virus contamination.
- *hoard* caches the entire state of the user's parcel on this client in order to prepare for a planned or possible network disruption.
- *ls* lists the state of all the user's parcels.
- *stat* prints information regarding the state of a particular parcel.

Each parcel may only execute on one client at a time. Hence, the operations checkout and check-in associate a given parcel with a particular client. The resume and suspend operations control the execution of the VM on which the user's environment runs. The discard and hoard operations control the caching of parcel data on a particular client, and the ls and stat operations provide feedback to the user regarding the state of the system.

These operations move the user's parcel through various states as shown in Figure 2. At a given client, the parcel is initially not present. When the user executes the checkout operation, the parcel's critical data are stored on the client and, consequently, the parcel is logically present, but unmodified. Once the user executes the resume operation, the parcel enters the running state and the user may interact with the reconstructed environment. Upon suspend, the environment is halted and the new state of the parcel is stored locally on the client. Note that the modified state is not propagated to the content server until the user issues a checkin command. Consequently, the user may continue using the environment by resuming the current state, or the user may return to the last saved state by issuing a discard command.

The hoard command simply populates the client-side parcel cache. By design, the system is organized in a manner that permits the user to start a hoard operation while in either the unmodified or modified state.

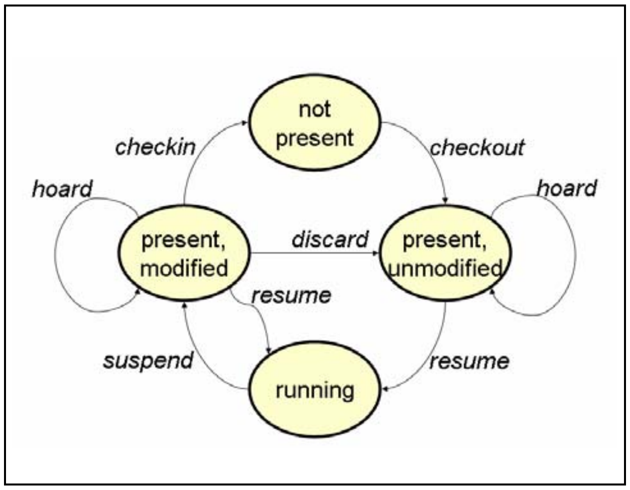
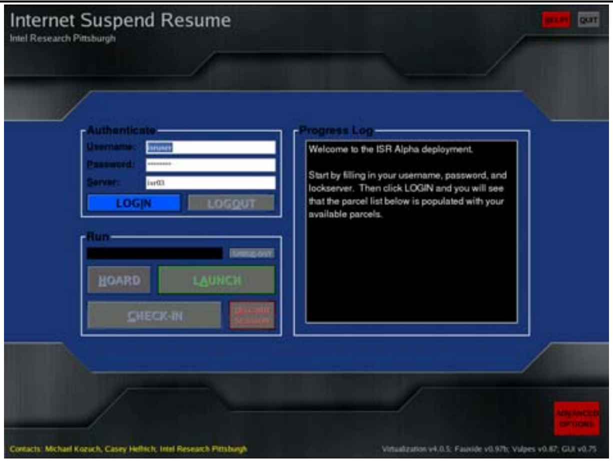


Figure 2: Client-side parcel state transition diagram

In our current implementation, these user commands may either be executed by the user on the command line or through the graphical user interface shown in Figure 3.



Welcome to the ISR Alpha deployment.

Start by filling in your username, password, and lockserver. Then click LOGIN and you will see that the parcel list below is populated with your available parcels.

Contacts: Michael Kozuch, Casey Helfrich; Intel Research Pittsburgh

Virtualization v4.0.5; Fauxide v0.97b; Vulpes v0.87; GUI v0.75

Figure 3: Prototype ISR graphical user interface

These operations are implemented through the orchestration of the three components that constitute the ISR client software depicted in Figure 4. The VMM supports the operation of the user environment, which includes the guest operating system and guest applications. The state files that compose the user's parcel are stored in the parcel cache and managed by the parcel cache manager. The VMM accesses these files on demand while the user's environment is running. The ISR client manager orchestrates the movement of data between the parcel cache manager and the content server during the operations listed above.

For example, during the checkout operation, the client manager first fetches a configuration file describing the location and organization of the user's parcel. With that information, the client manager begins to fetch the associated data over the network and primes the parcel cache manager before invoking the VMM to instantiate the user environment.

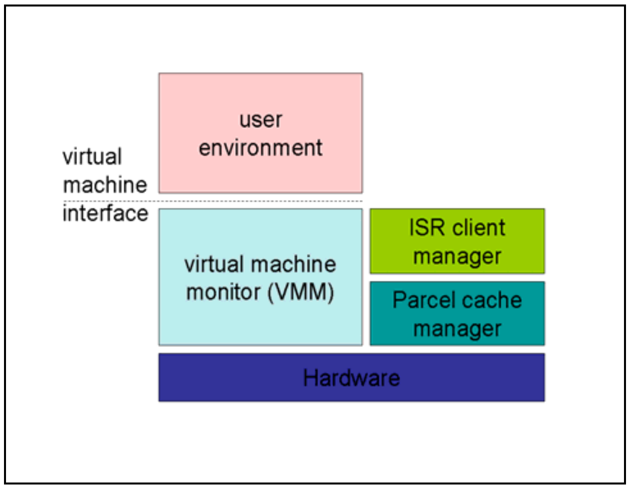


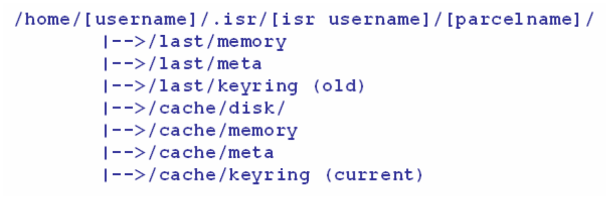
Figure 4: ISR client program

The parcel cache manager is responsible for organizing all parcel data on the client. The ISR design comprehends on-demand fetching of parcel data. That is, the client manager is not required to fetch the entire parcel before invoking the VMM to restore the user's environment. Instead, the client manager need only fetch the critical files needed to resume the VM. As the environment executes, the VMM submits data requests to the parcel cache manager. If the cache manager finds that a requested file does not exist in the cache, it fetches the file from the content server and installs it in the cache before supplying the requested data to the VMM.

In our ISR prototype, the parcel cache manager organizes parcel data into a traditional file hierarchy as depicted in Figure 5. This figure shows an example of the files that will be present on a client machine when a parcel is checked out and in use. The first line in Figure 4 describes the root of the directory used for this parcel. The directory contains two sub-directories: *last* and *cache*. The last directory contains read-only data describing the state of the parcel at the time of checkout. As the user's environment is running, changes to the environment and files fetched to satisfy demand misses are stored in the cache directory.

In particular, as the client uses the parcel, the *cache/disk* directory becomes populated, as parts of the disk are demand fetched. The client can also explicitly populate this directory with the hoard command. Until this directory is fully populated, the user should ensure that the client machine remains connected to the network because the client must be able to satisfy demand misses from the content server. However, once the disk directory is fully populated, the client can operate in disconnected mode and requires no further access to the network until the user wishes to check-in the parcel.

The hoard command enables the user to prepare a client for disconnection. This function is particularly useful for preparing a laptop before removing it from the network. The hoard command and the discard command both operate on the client's cache directory. Hoard causes the cache manager to populate the cache fully. Discard causes the cache manager to delete the cache without checking it in to the content server, thereby irrevocably destroying the data.



```
/home/[username]/.isr/[isr username]/[parcelname]/
        |-->/last/memory
        |-->/last/meta
        |-->/last/keyring (old)
        |-->/cache/disk/
        |-->/cache/memory
        |-->/cache/meta
        |-->/cache/keyring (current)
```

Figure 5: Client parcel cache example

Name Server Architecture

Each session on a client typically begins with a checkout operation, which, in turn, begins by contacting the ISR name server to determine the location of the user's parcels. The role of a name server is similar to that of DNS servers on the Internet. The name server is the only machine that a user needs to identify to the ISR client software, as the name server contains information that describes the location of all other content.

Each username within a name server must be unique because a <*name server, username*>-tuple identifies a single user in the ISR system. Further, the system must ensure that each parcel name is used only once for each user as a <*name server, username, parcel name*>-tuple uniquely identifies a parcel.

When the user initiates a checkout operation, the ISR client software executes an authentication sequence with the name server and fetches the metadata associated with that parcel. The metadata identifies the content server(s) responsible for maintaining the parcel. The client software uses the retrieved metadata to contact the appropriate content server and begin fetching data.

In some implementations, such as ours, the parcel metadata may also include encryption keys required for decrypting the content on the content server. Our current implementation of the name server is a hardened Linux[*] computer running sshd, the secure shell (ssh) daemon. The client program contacts the machine via ssh and downloads a single file called *parcel.cfg*. Parcel.cfg is a small protected text file that contains all of the necessary configuration data describing a parcel, including the protocol used to access the content, the path/URL of that content, and the master decryption key, called the *keyroot*. The parcel.cfg file can also be extended to contain any protocol-specific information.

Content Server Architecture

As mentioned previously, the ISR system imposes very few requirements on the operation of the content server. This component can be implemented as a distributed file system server, database server, or other distributed storage system such as a distributed hash table-based service.

Essentially, this component need only be able to deliver data files as ISR clients request them. However, many implementations will naturally organize their content as described in this section. Understanding the structure of a parcel is essential to understanding the structure of the content server. An ISR parcel contains three logical types of data: memory, disk, and metadata.

The memory data corresponds to the state of the physical memory in the VM. For example, if the user's VM includes 512 MB of RAM, this file describes the 512 MB state of physical memory. In practice, however, we have observed the size of the memory file to be typically half the size of the virtual machine RAM once it is compressed, encrypted, and written to disk. As a special case, when the VM has been powered off, this file can be eliminated as the contents of the virtual RAM can be reset during the next resume operation.

The disk data represents the contents of the disk(s) in the VMs. Disk data can potentially be very large–several gigabytes to several tens of gigabytes, and because of this, much of the research in ISR has been focused on the

[*] Other brands and names are the property of their respective owners.

management of disk data. To enable quick access to portions of the disk and to support on-demand fetch, the disk state is divided into small files that are compressed individually. Each of these files represents a contiguous range of sectors on the virtual disk called a *chunk*. Each of these files is individually compressed and encrypted whenever they are not in use by the virtualization layer on the client. Carefully encrypting these files is particularly important because they may contain information of unknown sensitivity including protected guest files, the guest swap file, and the virtual disk metadata. The memory image must also be carefully encrypted for similar reasons.

The metadata associated with a parcel is a collection of very small configuration files used by the virtualization layer, various logs, and one file that contains the decryption keys for the disk data, called the *keyring*. For each of the disk chunks, the keyring contains two 20-byte entries. The first entry is the disk chunk's decryption *key* and the second entry is the disk chunk's lookup *tag*.

Keyring

The encryption technique that ISR employs for encoding disk chunks is convergent encryption [3]. Convergent encryption provides the useful property that two users who have the same file can both encrypt the file to the same cipher text without having to exchange any encryption keys.

This result is achieved by deriving the encryption key from the data, itself. If both users employ a cryptographic hash of the data as the encryption key, each will encrypt the original plaintext to the same cipher text. Each disk chunk's key and cipher text are derived in this manner:

key = hash(chunk)

cipher text = $\text{encrypt}_{\text{key}}$(chunk)

The motivation for using this encryption technique is the observation that many users' parcels may exhibit significant data similarity. For example, the parcels of users whose environments include the same operating systems and/or applications may have very similar contents. If these users are able to satisfy disk chunk requests amongst themselves, they will possibly (a) reduce the load on the server, (b) reduce the load on the network, and (c) reduce the average latency of chunk requests experienced by the users.

To enable inter-user data exchange, the users must not only encrypt their data in a common manner, they must also maintain a means for addressing those blocks. Hence, the keyring file also includes a tag for each chunk such that the tag is a hash of the cipher text:

tag = hash(cipher text)

The tag now uniquely identifies a disk chunk. An ISR client can request disk chunk files using this tag and decrypt those files using the associated key.

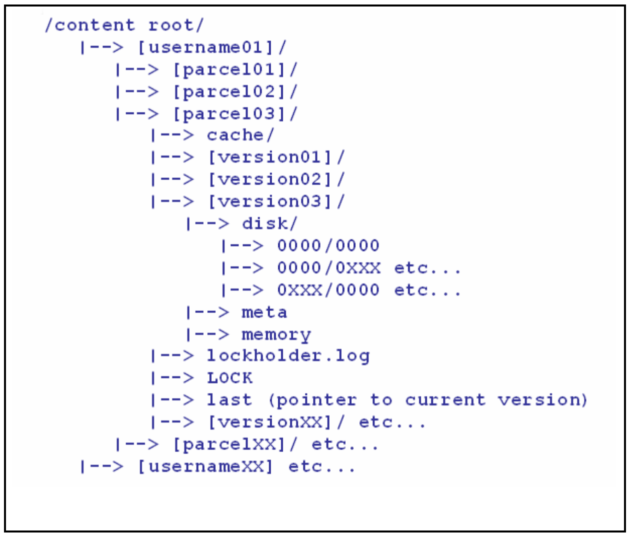
As an example, suppose that user A is working on a client machine that does not contain a fully populated cache for user A's parcel. User B uses the same operating system and applications as user A, and user B's client machine is in an adjacent office. Before resuming the VM, A's client has fetched the keyring associated with A's parcel. At runtime, if A's cache manager determines that a particular chunk file is needed, it can lookup the tag associated with that chunk in the keyring and request the file by tag value from B's client (by using, for example, the techniques reported in [2]). If B's client returns the file, A's client can decrypt it using the associated key.

Note that while clients do advertise the chunks being sought using the tag associated with that chunk, the tag does not reveal any information regarding the contents of the file and cannot be used to decrypt the file. Only the keys can be used to decrypt the chunk files, and these are never exchanged between clients. Instead they are stored in encrypted form in the keyring on the content server and only decrypted on the client. The encryption key for the keyring is the only secret a user will need to unlock access to his system. This secret is part of the parcel.cfg file that the client obtains from the name server during a checkout operation.

While initially intended as a mechanism for maintaining the association between chunk files and their tags and keys, the keyring has proven to be a surprisingly useful tool. For example, the keyring file allows for quick calculation of disk changes by comparing a newer keyring with an older one. Because each keyring includes a single entry for each disk chunk, by comparing the two keyring files entry-by-entry ISR software can determine which of the disk chunks has been modified. This technique is used in our implementation of the checkin operation to determine which disk chunks need to be uploaded from the client to the content server.

Content Server Structure

The structure of the content server will vary slightly, depending on which protocol it services. Figure 6 depicts an organization that should be useful for http- or distributed-file system-based content servers.



```
/content root/
   |--> [username01]/
      |--> [parcel01]/
      |--> [parcel02]/
      |--> [parcel03]/
         |--> cache/
         |--> [version01]/
         |--> [version02]/
         |--> [version03]/
            |--> disk/
               |--> 0000/0000
               |--> 0000/0XXX etc...
               |--> 0XXX/0000 etc...
            |--> meta
            |--> memory
         |--> lockholder.log
         |--> LOCK
         |--> last (pointer to current version)
         |--> [versionXX]/ etc...
      |--> [parcelXX]/ etc...
   |--> [usernameXX] etc...
```

Figure 6: Content server data organization

Figure 6 depicts a content server that provides service for several users and permits multiple parcels per user. Each parcel can have many versions; each version represents the state of the machine at a checkin time during the life of the parcel. Each version contains enough information to recover the state of the parcel at that time. The version pointed to by "last" contains a fully populated parcel, while the previous versions only contain the delta information describing what virtual machine state changed between it and the subsequent version.

Parcel Versioning and Rollback

Rollback is an operation that enables a user to revert the state of his/her environment to the state associated with a previous checkin point. For example, if a user realizes that she introduced a computer virus into her environment on Wednesday, the user could employ the rollback operation to restore the state of the parcel to the state corresponding to some version prior to Wednesday's.

To support this feature, the ISR content server maintains a version of the user's parcel for every possible rollback point. Naturally, naively maintaining many copies of the parcel would require an excessive allocation of storage space. Instead, we can capitalize on the fact that a parcel typically does not change much from one version to the next. Consequently, each version directory only contains the data that changed between that version and the next. We expect the delta-encoding format to provide a space savings of between one and two orders of magnitude.

The number of versions maintained per parcel and the timing (weekly, monthly, etc.) of those versions are policy decisions set by the content server administrators. Fortunately, the delta-encoding format of the versions enables the system administrator to collapse several versions when it becomes necessary to reclaim space. Consequently, the policy may be dynamic; when the available disk space on the content server falls below some threshold, a reclamation process may iterate through all the parcels collapsing versions until sufficient free disk space becomes available.

Data Transport Protocol

As mentioned previously, we have experimented with two different content server protocols. The first protocol requires that the ISR content server data appears to reside in the local file system of the host. The data could either be truly local (on an attached portable hard-drive, for example), or they could appear to be local (in a mounted distributed file system, for example). In either case, the ISR client software accesses the data through simple file operations such as open, close, read, and write.

In the second protocol, the disk data are logically remote, and the ISR client software must fetch the data explicitly from a remote server via http or ssh. When a section of disk is requested and it is not yet cached on the client, the chunk files are explicitly requested from the content server and cached on the client to service future access requests.

The system currently supports both approaches. During a checkout operation, the client fetches the parcel.cfg file that describes which transport protocol is used for this parcel. Further, we have defined an abstraction layer in the client software so that the client can switch between the various transport protocols by calling into dynamically loaded libraries. This mechanism also supports the development of new transport mechanisms.

CHALLENGES AND SOLUTIONS

In developing the ISR architecture and initial implementation, we encountered a number of challenges. In this section, we list several of these challenges and our current solutions.

Large Environment State

By far, the greatest challenge that ISR presents is the enormous volume of data associated with a parcel. In particular, the virtual RAM is typically on the order of a hundred megabytes and the virtual disk drive is on the order of gigabytes. A naïve approach for data movement, transferring the complete image between client and server during every checkout and checkin operation, would be impractical even over fast corporate networks. Transferring ten gigabytes over a 100 Mbps network requires at least 800 seconds (15 minutes).

We have already described several of the techniques employed to reduce the impact of the state size. First and foremost, we have organized the parcel so that the disk data may be fetched on demand. This reduces the volume of data that must be fetched when performing a checkout operation on a client with an empty cache to the size of the memory image rather than the disk image. Further, we employ standard compression to reduce the footprint of the memory image. If the compressed memory image is 100 MB, fetching the image over a 100 Mbps network will require at least eight seconds. In practice, the startup sequence including authentication, download, decryption, and decompression requires approximately 30 seconds.

Each of the disk chunks may be fetched separately, and consequently, each is compressed and encrypted separately. In our implementation, we have chosen a chunk size of 128 KB. The compression ratio can range from 0% to 100%. Portions of the disk that are empty, because they have not been used by the guest operating system, compress to 152 bytes while portions that contain nearly random data are uncompressible.

The chunk size is a parameter chosen by the system administrator for each parcel. The chunk size is essentially the cache line size for the parcel cache on the client. Choosing a larger chunk size will typically reduce the number of misses observed by the cache but will also increase the network bandwidth consumed by the client. Our experiments indicate that chunk sizes in the range of 64 KB to 256 KB are reasonable.

To compensate for potentially poor network performance, we rely heavily on client-side caching. Every chunk fetched by the client software is placed in the client parcel cache. Further, because this cache is disk-resident and disk space is relatively inexpensive, we assume that evictions will be relatively rare events. We also define the hoard operation to provide good performance in cases where the use of a particular client is known *a priori*. This operation can also be issued remotely. If a user knows that she is going to a remote site with low network bandwidth, a hoard operation can be issued remotely to ensure that data are prefetched into the remote client cache while the user is in transit.

For situations in which the network conditions are poor, and the location could not have been predicted, we have developed an optional performance enhancement to ISR called Lookaside caching (LKA) [5]. This technique relies on the user carrying a portable storage device such as a flash memory device to maintain compressed and encrypted copies of parcel data. When the user attempts to access his environment from an ISR client, the ISR client software can fetch necessary data from the device rather than from the content server. However, we require the client to verify the validity of the data on the portable device with the content server before using it. Again, we can rely on the keyring to provide this validation. Because the keyring contains the hash for every disk block, the client can simply hash data blocks on the portable device and compare the hashes to the keyring tag to determine if the portable device contains a correct copy of the data. In this way, the system never relies on the portable device; it can be lost, forgotten, or contain old data. The content server is always considered to contain the authoritative version of the user's parcel. Further, because the data on the portable device are encrypted, other users cannot make use of the portable device if it is lost or stolen.

Finally, we rely on delta encoding during both checkin and rollback operations as well as for reducing the amount of space occupied on disk by our parcel versioning system. For example, to represent a virtual machine with 256 MB of RAM and a 10 GB hard drive with Microsoft Windows[*] XP* and Microsoft Office* XP* installed occupies approximately 2.4 GB of server side storage after compression. For each rollback point that a user chooses to keep, delta encoding often reduces the server storage required to approximately 100 MB per version.

Heterogeneous Clients

Another potential impediment to widespread ISR adoption is the diversity of client machines found in the typical enterprise. Fortunately, the virtualization layer in the client software stack handles heterogeneous clients relatively easily.

In fact, virtualization is able to convert this challenge into a feature. If an enterprise adopts the ISR architecture, migration from one platform to the next becomes much easier. Currently, to migrate a user from one platform to a newer platform with a faster processor, either the IT department or the user must construct a new environment on the new machine and carefully consider which files and programs to install on the new machine before destroying the old environment. With ISR, simply suspend execution on the old machine, checkin the parcel, replace the hardware, checkout the parcel, and resume.

The one requirement that ISR imposes on the virtualization layer is that all client software provide the same VM interface. The VMMs on different clients need not be identical, but they must export the same VM interface. If the VMMs do not export the same interface, guest software may become confused when it resumes on a virtual machine that is different from the suspend site VM.

On the surface, differences in attached peripherals on various clients would appear to be a problem, but with the advent of solid support for plug-n-play devices in modern operating systems, this issue is largely resolved. Modern operating systems are able to handle the addition or

[*] Other brands and names are the property of their respective owners.

removal of plug-n-play devices (a USB printer, for example) cleanly and transparently. For example, suppose that the suspend site client is connected to a USB printer, but that no printer is connected to the client at resume time, the guest operating system will simply detect that the device has been unplugged and respond accordingly.

Similarly, modern operating systems are typically able to respond well to changing network conditions. When a modern laptop is disconnected from one network and reconnected to another network, the operating system is typically able to reconfigure the network stack automatically to compensate. In the same way, if a guest operating system within a VM detects that the network conditions changed between suspend and resume, the operating system will reconfigure networking to compensate.

Naturally, persistent network connections do prove to be a problem in the ISR system across suspend-resume cycles, but no more so than in general laptop usage. Persistent network connections are typically disrupted during laptop suspend-resume cycles due to time-out, address migration issues, or both. A suspend-resume cycle in the ISR system will cause similar results.

OVER-THE-WIRE MOBILITY

While this paper primarily discusses ISR in the context of enterprise management, the same infrastructure also supports over-the-wire mobility [1]. In particular, the same technology that enables a user to resume his or her environment on an anonymous hardware platform after a hardware failure enables the user to resume on a different hardware platform in the absence of failures. The user simply suspends on one client and resumes on another.

This capability is potentially very interesting in a number of situations. For example, some environments require a fluid office allocation; when employees arrive for work, they are given an office assignment out of a pool of available offices. ISR supports a clean mechanism for customizing the computers in those offices. The user simply sits down at the client machine and resumes his or her environment.

ISR could also be used to support employees who work at home some of the time and at work some of the time. Rather than carrying a laptop to and from work, the employee can leverage the network to transport her work environment from work to home and vice versa. This scenario is a particularly good application of prefetching as the employee may follow certain patterns such as arriving at work every day at 8:00 and leaving at 5:00. Such predictability greatly improves the effectiveness of the client parcel caches.

CURRENT PROJECT STATUS

We have built a prototype implementation of ISR and have been using the prototype internally for experimentation for some time. The prototype has become sufficiently robust that we plan to conduct a test deployment of the system with external volunteer users.

Through this test deployment, we hope to evaluate the following questions quantitatively:

- *How do users use the system?* How many times does the average user checkin, suspend, and discard? Are users doing something unforeseen and/or interesting with the system?
- *What are the hardware requirements for the system?* What is the load on the server? What is the load on the network? How much disk space is consumed per checkin?
- *How satisfied are users with the system?* Does the system fulfill an interesting need? Will users continue to use it? Will users recommend it to their friends?
- *Does ISR improve enterprise management?* Does system administration time increase or decrease in the context of ISR?

To answer these questions, we have instrumented both the server-side and client-side ISR code. The instrumentation will gather various statistics such as average number of bytes sent per checkin and frequency of checkin. The users will be fully aware that this information is being gathered, but we will try to protect their privacy by making the collected data anonymous and not observing activity *within* the guest environment.

We hope to start with approximately ten users and to increase that number to approximately 100 at the peak of the trial. Additionally, we hope to attract tolerant users in the early stages as we work out any bugs that remain at the beginning of the deployment. In the later stages, we plan to open the deployment to more novice computer users so that we can better evaluate a typical user's impression of the system.

CONCLUSION

ISR leverages virtual machine technology and network accessible storage to improve the management of enterprise clients by (1) cleanly separating the client software stack into a portion that is managed by IT and a portion that can be managed by the user, (2) providing centralized management of the entire user software environment, and (3) providing a mechanism for simple, rapid environment migration. Through this combination, ISR is able to simultaneously deliver the administrative benefits of centralized management and the rich user experience of thick-client computing.

REFERENCES

[1] Kozuch, M., Satyanarayanan, M., Bressoud, T., Helfrich, C., Sinnamohideen, S., "Seamless Mobile Computing on Fixed Infrastructure," *IEEE Computer*, July 2004, pp. 65-72.

[2] Bressoud, T., Kozuch, M., Helfrich, C., and Satyanarayanan, M., "OpenCAS: A Flexible Architecture for Building and Accessing Content Addressable Storage," *2004 International Workshop on Scalable File Systems and Storage Technologies*, September 15, 2004.

[3] Douceur, J., Adya, A., Bolosky, W., Simon, D., and Theimer, M., "Reclaiming space from duplicate files in a serverless distributed file system," *Proceedings of the International Conference on Distributed Computing Systems (ICDCS 2002)*, Vienna, Austria, July 2002.

[4] Goldberg, R., "Survey of Virtual Machine Research," *IEEE Computer*, June 1974, pages 34-45.

[5] Tolia, N., Harkes, J., Kozuch, M., and Satyanarayanan, M., "Integrating Portable and Distributed Storage," *Proceedings of the 3rd USENIX Conference on File and Storage Technologies*, 2004.

AUTHORS' BIOGRAPHIES

Michael Kozuch is a senior researcher for Intel Corporation. Mike received a B.S. degree from Penn State University in 1992 and a Ph.D. degree from Princeton University in 1997, both in electrical engineering. Mike has worked for Intel research labs since 1997, four years in Oregon and three years in Pittsburgh, Pennsylvania. His research focuses on novel uses of virtual machine technology. His e-mail is makozuch at ichips.intel.com.

Casey Helfrich is a research engineer at the Intel Research Lab in Pittsburgh. He received a Bachelor's degree in Physics from Carnegie Mellon University in 2001 and an additional B.S. degree in Computer Science from Carnegie Mellon University in 2002. He joined the Pittsburgh lab at its inception and helped design and build the IT infrastructure for Intel Research. His e-mail is casey.j.helfrich at intel.com.

David O'Hallaron has been a faculty member of the Carnegie Mellon University School of Computer Science since 1989. His interests include high-performance distributed systems, Internet services, distributed search, and scientific computing. David has co-authored three books including *Computer Systems: A Programmer's Perspective* and won the 2003 Gordon Bell award for special achievement. His e-mail is droh at cs.cmu.edu.

Mahadev Satyanarayanan is the Carnegie Group Professor of Computer Science at Carnegie Mellon University. His research interests include mobile computing, pervasive computing, and distributed systems (especially distributed file systems). From 2001 to 2004 he was the founding director of Intel Research Pittsburgh, where the Internet Suspend/Resume project was initiated. He is a Fellow of the ACM and the IEEE, and the founding Editor-in-Chief of IEEE Pervasive Computing. His e-mail is satya at cs.cmu.edu.

THIS PAGE INTENTIONALLY LEFT BLANK

For further information visit:
developer.intel.com/technology/itj/index.htm