Freeing your Computer from the Hardware

Constantine P. Sapuntzakis Ramesh Chandra James C. Norris
Monica S. Lam Mendel Rosenblum

Computer Science Department
Stanford University
{csapuntz, rameshch, jcn, lam, mendel}@stanford.edu

ABSTRACT

In this paper we propose an alternative software model for modern computing environments. The new model features the decoupling of software execution from the computer hardware by encapsulating all software execution in an abstraction we call *capsules*. We treat capsules as first class objects that can be named, controlled, migrated, and replicated.

We argue that by putting software execution into capsules and treating them as first class objects we can address many of the important problems facing modern computing environments. We show how this structure can help deal with difficulties in system administration, security, and mobility.

Furthermore, we describe a vision for a future capsule-based computing environment of a *compute utility* that allows an environment of distributed machines to be used and managed with ease, flexibility, and efficiency. Our prototype of the utility, call the Collective, uses capsules to provide mobility of user environments, fault-tolerance, and easier system administration.

We describe the current implementation of the Collective that implements capsules as virtual machines on VMware's x86-based virtual machine technology. The system is able to demonstrate the benefits of capsules while leveraging the huge base of applications written for x86-based personal computers.

1. INTRODUCTION

Software on modern computer systems has evolved to form a hard coupling with the computer hardware it runs on. Operating systems and application programs must be "installed" on a particular computer before they can be run. Furthermore, executing software builds state and connections that form deep couplings with the operating system and hardware, effectively tying the computation to the machine on which it was started.

In this paper we propose an alternative structure in which both running and non-running software is encapsulated, decoupled from the hardware, and treated as a first-class object that can be named and manipulated by the system. We call this encapsulated software environment a *capsule*. Capsules can contain executing software as well as all the state needed to support the execution. This can include multiple processes, files, and other software including libraries or even a complete operating system.

Capsules run on a *platform* but are not permanently coupled to the platform. As first class objects, capsules *docked* on a particular platform can be *undocked* from the platform and encoded as a string of bytes that can be transferred over a network or stored in a permanent store. Besides migration, capsules can be replicated with a *cloning* operation.

The benefits of encapsulating computation have been recognized before by research projects such as Condor [12], and Sprite [3], and companies such as Ensim and Ejacent. However, we are the first to explore implementing capsules as virtual machines running on a virtual machine monitor. By having the capsules match the interface of an industry standard computer such as x86 compatible PCs [7, 19], we are able to construct capsules capable of running and manipulating most existing software. Furthermore, virtual machine monitors allow capsules to be deployed in existing computing environments, leveraging the large base of existing software.

We show that this capsule-based computing can be used to solve a number of key problems in modern computing environments. We show how some of the harder problems in the areas of system administration, security, and mobility become easier with capsules. Furthermore, we argue that this encapsulation and promotion of software to first-class status leads to a new "utility" model of computation where computing services will be viewed as available and easy to use as other modern utilities, like power, water, or telephone service. We show how this new model generates a superior computing environment.

This paper is organized as follows. For the rest of this section we describe the benefit of capsules in today's computing environments (Section 1.1) as well as a future compute utility model (Section 1.2) and how virtual machine monitors can be used to implement capsules (Section 1.3). In Section 2 we describe the beneficial properties of capsules while in Section 3 we detail the operations supported on capsules. Section 4 describes our use of a virtual machine monitor to implement a platform for capsules. Section 5 describes the system architecture of the Collective, our prototype compute utility constructed using commodity x86-based software running on virtual machine monitors. Section 6 and Section 7 contain a discussion of the related work and concluding remarks.

1.1 Using Capsules in Today's Environment

Capsules can be used to manage the large number of computers around us. They address several important issues we face today, as discussed below.

System administration. The difficulty of administering and maintaining our systems is a major source of lost productivity in the workplace and a barrier to entry of

computing in the home. System administration requires detailed knowledge of the hardware and operating systems, and is a frustrating task even for computer experts. Many users must manage their own systems; not being experts in the field, they end up spending a lot of time learning the intricacies of the system, performing administrative functions, and often recovering from costly mistakes.

Making operating systems first-class capsules gives us new ways of addressing this problem. Operating system capsules allow system administration tasks to be performed once, encapsulated and replicated on multiple platforms. We recover from operator mistakes by checkpointing the capsules before making any modifications.

Security. Numerous computer attacks in recent years have resulted in significant data and time loss. Today's security problems are compounded by home machines administered by end users who are not well versed in security issues. Only 2% of the machines in use today are up-to-date with regard to security patches. By using capsules, we need only apply patches to one version of the operating system that is then run on many machines.

Mobility. Despite the pervasiveness of high-bandwidth Internet connections and the low cost of computers, we currently achieve mobility by carrying our laptops everywhere. We end up juggling a variety of different computing environments, e.g. a desktop machine, a laptop, and possibly also compute servers, because none of the above fulfills our computing needs on its own. With compute capsules, we can simply transfer the capsule between computers, so that we can use the same computing environment anywhere we go.

1.2 A New Compute Utility Model

Capsules can also be used in creating new computing environments. We envision that, in the future, computing services will be just as available and easy to use as any of the modern utilities, like power, water, or telephone service. Instead of requiring users to procure, administer and upgrade their equipment, resources of a utility are shared and managed centrally by experts, thus resulting in a more efficient system. A managed compute utility designed to support global mobility and sharing can provide greater security than our current environment. A global utility gives users efficient access to their computing environment from anywhere in the world.

Capsules are important in ensuring the integrity and manageability of the system, both important qualities of a compute utility. Compute servers in a utility run a small trusted computing base, which serves as a platform that can run multiple capsules. It is easy to guarantee isolation and protection between the capsules because of the well-defined and thin interface between the platform and the capsules. These servers manage compute capsules like a cache; jobs can easily be put onto newly installed machines and taken off machines to be turned off for maintenance. While a user's capsules can be run on any of the compute servers in the utility, interactive performance can be improved by migrating capsules closer to the user.

1.3 Realization of Capsules

Key to the design of first-class capsules is the definition of an interface between capsules and the platform. This interface must be well-defined and preferably narrow. A capsule must be self-contained; all the information required to establish the capsule on another platform must be contained in the capsule. A particular attractive interface is the machine interface. The industry-standard x86 architecture is a well-established and well-defined interface and runs much of today's software. A capsule in this case simply consists of all the activities that normally take place on a machine. These activities include booting up the operating systems, logging in the user, running user programs, performs I/O, and interfaces to the network.

We use virtual machine monitors to make operating systems into first-class capsules. The capsules then correspond to virtual machines. The virtual machine monitor implements docking, undocking and serialization as part of its primitives.

The virtual machine monitor provides the platform to the capsule. The platform provided looks like a plausible hardware platform to the capsule, but the monitor is free to implement the various devices in the hardware platform as it sees fit. It can map the devices directly onto real hardware or simulate the devices in software.

Our prototype is built upon VMware GSX Server from VMware Inc. The GSX Server is a virtual machine monitor that, on a single x86 computer, can host multiple operating systems, each in its own virtual x86 machine. For example, the VMware virtual monitor software allows multiple Windows and Linux systems be run on the same machine [18]. VMware provides another important function. PCs have many different devices, and it is highly desirable that a capsule running on one PC be runnable on another with different devices. VMware supports exporting the same set of virtual devices to the capsule on every platform, even if the hardware under the monitor varies. For the most part, all PCs that can run the monitor can run the capsules. Finally, VMware runs most of the PC software available today; most of the software people use today can be put into capsules.

2. BENEFITS OF CAPSULES

Capsules have some new fundamental properties. These properties include a decoupling of the software from the hardware, the ability to control and isolate the software execution in capsules, the ability to save capsules to disk and transfer them around, and the ability to manipulate capsules like data.

2.1 Freedom from a hardware instance

Putting a computation in a capsule frees the computation from a particular computer and allows the computation to move across computers. This mobility of the software is useful for integrating new machines as well as dealing with failed machine. With capsules, new machines can be brought online and taken down easily by simply migrating the capsules to and from the machines, respectively.

2.2 Isolation

By running multiple capsules on the same machine, we can isolate the computations such that they are protected from each other, they can be given different access rights, and they do not interfere with each other.

2.2.1 Isolation for protection

Users often want environments with different personalities, each of which could be contained within a capsule. For example, a user could have one capsule for normal desktop usage, a fail-safe capsule with a minimal environment and no external dependencies for running untrusted software, capsules for work-related and personal activities, etc. Capsules can also be used to create environments that contain precisely the information the user wishes to pass to application service providers.

A logical machine can be created to contain a unique set of rights, privileges as well as a computing environment. Consider a shared project workspace, such as a development environment or a collaborative system. The project coordinator can create a capsule and grant access rights to the group members. A private project file space can be created, and it will only be accessible to group members. Persons working on the project simply join the group capsule and begin to immediately share resources with the other members. In our experience many difficulties in software development and distribution stem from improper configuration of the computing environment and the unavailability of the right versions of various tools and libraries. Such problems can be eliminated by encapsulating a complete working environment and the necessary tools in a capsule.

2.2.2 Isolation of failures and unwanted interaction

PCs are more reliable if they are dedicated to single applications rather than running many different applications on the same machine. Companies would buy different PCs to perform different functions even though the same machine could have more than adequately handled all the functions. For example, the PCs used in a teaching laboratory at Stanford are used only for one class at a time because it is too unwieldy to maintain different environments needed by different classes. The use of capsules will provide the safety guarantee that allows the same PCs to be used for different functions.

2.3 Persistence and Portability

Today, software installations can get quite complex because a package often has many dependencies. For example, an application may require the presence of specific versions of other applications and runtime libraries, or that certain environment variables be set. The requirements of different applications may even conflict with each other. It is not uncommon for software engineers to spend an enormous amount of time to acquire and build the environment they need for their work. It is useful to be able to share their efforts in this regard. The cost, time, hassle, and the potential security risk associated with installing, setting up, and running new software have deterred many from experimenting with new applications.

Instead of distributing code, we can now distribute capsules. Capsules represent the entire environment that is ready to be executed. Capsules can be run on the same machine with all other computations without interference.

The recent emergence of application service providers is in part a reaction to two problems: they provide users hassle-free access to fully configured and supported software, and they provide users with a way to share data. Examples include web sites that file your tax return, and web sites that host your photos. Unfortunately, in this model, users' personal data are scattered across the different web sites and users have little control over the privacy and security of the data. Users are also limited in their ability to manipulate their data.

The use of compute capsules allows us to return to the more natural data-centric mode of operation with the convenience of easy application access. The capsules provide a generic way for users to package together arbitrary applications and data and allow them to share them with others. Individuals can instantiate a compute capsule, prepackaged with all the useful routines, and use it to access their personal data. The capsules together with all the relevant data can be shared between friends or relatives. There are no restrictions on the applications; they will be executed on users home machines or a local node in the utility and can thus have arbitrarily rich interactive interfaces. In contrast to the application service provider model, the computations are distributed and operate independently, and can thus easily be scaled up to serve many millions of users.

2.4 Capsules as Data

We imagine that capsules in the future will be prepared by IT staff of institutions or third party vendors who put together complete and convenient packages that provide some popular functionality. A capsule is a collection of rights to various software, resources and shared data; the right to the entire package is granted as a whole by giving users access to the capsules. Users log onto instances of these capsules, and upon presentation of the right credentials, can apply the programs in the capsules to their personal data. An active capsule, with running processes, is itself a capsule that can be suspended and migrated to other machines. This individualized capsule is normally only accessible to the owner of the instance. An expert user may choose to specialize a capsule, by adding to a capsule rights to programs he owns or access to data he wishes to share, and makes the capsules available to specific parties.

A user can choose to start a new session by logging onto an instance of the standard capsule prepared by the staff, or continue to operate on a previous session. This means that the administration staff can upgrade a system by issuing new capsules without having to worry about interrupting any ongoing work. Individuals may experiment with their environment without the fear that a single misstep can blow away all the system administration effort invested.

3. OPERATIONS OF CAPSULES

Capsules have the following list of basic operations:

- quiesce, which is to put a computation into a state ready to be made into a capsule.
- capsulize, which is to capture the state of the computation in a serialized representation that is portable.
- undock, which is the operation to release a capsule from a platform.
- dock, which is the operation to bind a capsule to a platform.
- clone, which is to replicate a capsule.

3.1 Naming

Traditionally, a computation usually hardwires the names of the hardware resources used into its state, and sometimes even exports them to the outside world. It is these bindings that make computations hard to migrate. Fundamentally, we need to find all these names and rebind them to the new resources on the new machines. There are several main techniques used in handling this rebinding of platform specific names.

Name rewriting. Rewrite the names in the capsule with the specific platform's resources when we dock the capsule and vice versa. If we think of code as a degenerate capsule with no data state, then a relocatable object loader does precisely that. This requires knowing exactly where these names to be rewritten are. This is not doable in a general environment where it is unclear where these names are used.

Virtualization. The most common technique is to introduce a level of indirection between the software and the hardware resource names. The software refers to the resources by a virtual name, which is then mapped into a physical name on the platform. By changing the mapping, we can easily change the bindings to refer to the new resources. This requires intercepting all calls to acquire resources so as to create a mapping between the virtual and the physical names. All subsequent references to the virtual name must then be translated to the physical name. The bulk of the state in a computation is its memory space, whose virtualization was introduced in the early 60s. The same idea applies to pretty much all the resources on the platform. Virtualization relies on translating all uses of the names, which is not possible once the name is exported externally.

Proxies. Proxying is the technique of forwarding accesses of the old name to the new name. Consider the case when a program communicates with another process using its IP address. Suppose we wish to migrate the process and give it a new IP address, the ideal would be to inform all communicating processes of the new change. However, this is often impossible. Thus, a common technique, such as the one used in mobile IP [4], is to use a proxy to accept traffic for the old address and forward them to the new one. The use of proxies is not always possible. There may be no network connection to the original platform; the original platform may have ceased to exist; proxying may be too expensive because it may take up too much bandwidth or introduce too much latency.

Dangling bindings. Another approach is to simply eliminate the binding, and rely on the application to re-establish a connection. For example, many applications today are designed to handle TCP/IP connection failures. They will simply note that the connection is no longer active and re-establish the connection.

3.2 Issues

Programs are normally run to completion once they are started. User applications are often written knowing that they are run in a time-shared manner. On the other hand, an operating system is in a sense a real-time system, expecting that the timer interrupts will deliver control back to the operating system at certain intervals. With capsules, we have the option of interrupting the normal course of execution, and this raises a number of interesting issues as discussed below.

Shelf life. A time lag may be introduced from the time the capsule is created to the time it is run. Limitations may be placed on how long capsules can be shelved if certain real-time behavior is expected. Similarly, a program may have connections with the external environment which expects the program to have a continual presence. If the capsules are shelved, locks may get broken by timeouts and the program may miss rounds in distributed algorithms. Thus, some capsules may need to be revived within certain intervals, or they may need to be revived upon the arrival of sime external stimulus.

Idempotency. A capsule can be cloned and re-executed multiple times. This capability may be useful for, say, debugging a long-running program. You may wish to encapsulate a state reached after days of computation, so that you can return to it without re-executing the program. Checkpointing an execution in case a fault occurs is another example. However, it is not always possible to re-start a state. For example, if we save the state of a program that has just obtained a lock on a shared external resource. If one execution releases the lock, then re-executing the same code would not work because the lock is no longer held. We say that a capsule that can be re-executed successfully is *idempotent.*

Allowing multiple identities. Multiple clones of the same capsule may be executed at the same time. Idempotency is necessary but not sufficient. For example, if the operation of a capsule relies on having a unique name, then running replicas of the same capsule will fail. One-to-one connections and sessions, like TCP connections, are problematic.

3.3 Initiation of Capsule Creation

Normally, users are expected to initiate the creation of capsules. The user understands the application and can initiate the capsule creation at a time when the state is compatible with its intended use. For example, if we wish to distribute a capsule, we must ensure that it allows multiple simultaneous executions.

It is also useful to have the platform initiate a capsule creation. An example can be found in the context of a server in a compute utility. Many users may have logged in and have docked a large number of compute capsules on the platform. However, most of these capsules may have been left idle for a long time. To make efficient use of the resources, it is desirable to undock some of these long-idle capsules to free up the resources they acquired. A reasonable interface is to have the platform inform the capsule that it will be undocked for possibly an indefinite amount of time, and that the computation should reach a quiescent state in preparation of the action. Some of the power management features in modern operating systems can be used to implement this functionality. Analogies can be drawn between undocking a computation to hibernating an operating system.

4. VIRTUAL-MACHINE BASED CAPSULES

The lowest level of our platform prototype is built on top of VMware Inc. virtual machine monitor technology. In the following, we first describe the capabilities and attributes of the VMware technology, then followed by a discussion how we can use the technology to implement the basic primitives of capsules.

4.1 VMware Virtual Machine Monitor

Our capsules are implemented as virtual machines using

virtual machine monitor technology taken from VMware's GSX server product. A virtual machine monitor [5] is a software layer than runs directly on the raw hardware and exports an abstraction of a virtual machine that looks enough like a hardware machine that all software (BIOS, operating system, and application programs) run unmodified in the virtual machine.

Using VMware's virtual machines for capsules offers a number of beneficial capabilities:

Compatibility. The virtual machine abstraction exported by the VMware technology looks enough like standard Intel-based hardware that most x86-based software environments run unmodified. This allows the capsules to run popular software environments such as those using the Windows family of operating systems from Microsoft and as well as open-source operating systems such as Linux. Unlike traditional virtual machine monitors that try to match the native hardware, the VMware's virtual machines exports a standard virtual machine interface regardless of the actual native hardware devices. This capability hides hardware differences between machines allowing capsules to be moved between compatible CPUs regardless of the I/O device configurations of the machines.

High Performance. Like traditional virtual machine monitors, VMware's technology uses direct execution to execute the virtual machine resulting in the performance of software running in the capsule being close to that of the native hardware. Much of the virtual machine execution can be done with same performance with virtualization overheads occurring only when the monitor gets invoked. These virtualization overheads, occurring on traps and exceptions, privileged instruction execution, and I/O accesses range from negligible for CPU bound applications to as much as 30% slowdown for I/O or OS intensive applications.

Safe Hardware Multiplexing. Like traditional virtual machine monitors, VMware's technology can multiplex many virtual machines on to a single computer allowing many capsules to share the same hardware. Furthermore, this sharing can be done in a safe way with the virtual machines isolated from one another so that software running in one capsule cannot harm the monitor or other capsules.

Encapsulation. The virtual machines are fully encapsulated and decoupled from the actual hardware by the monitor. All external communication to and from the virtual machine occurs through virtual I/O devices that can be controlled by the monitor. This capability allows the monitor to manipulate capsules like a multitasking operating system handles processes. The monitor can de-schedule a capsule from a CPU and then reschedule it on that or a different CPU at a later time. The mapping of virtual machine components such as CPU, memory, and I/O devices can be controlled by the monitor dynamically and transparently to the software running in the virtual machines.

The encapsulation features also allow the monitor to serialize virtual machines regardless of the execution state of the software. This feature allows capsules to be saved as collections of bytes and then later restored on the current or some other monitor.

Remote Control. The VMware GSX server platform allows the virtual machines to be configured and controlled remotely. This includes the scheduling and running of virtual machines as well as the binding of virtual I/O devices to physical devices. Operations like starting and stopping virtual machines, suspending machines can be done with a remote protocol allowing remote control of capsules. Furthermore, the technology allows virtual I/O devices to be bound to remote I/O devices over a network. For example, the disk or video card of virtual machines can be bound to access a remote disk or display of a user. This capability allows the full control of the external connections of capsules.

4.2 Different Kinds of Capsules

A VM capsule, in its full generality, consists of the disk image of all the system and user software, and user data, along with the state of the running OS and all the user's processes. More specifically, different capsules can be captured at different points:

- *Boot capsules.* This is a degenerate computational state that has absolutely no reference to any of the objects on a platform. It clearly represents an idempotent capsule that can be run on arbitrarily many different platforms at the same time.
- *Log-in capsules.* The operating system is ready to be logged onto by any authorized user. Here, saving this state means that the user does not have to wait for the boot up sequence. Moreover, this state is still relatively clean and can easily be executed on different platforms at the same time.
- *User capsules.* After a user logged in, the computation is now operating with the rights, privileges of the user. Saving the state at this point allows the person accessing the capsule to have full access to the user's environment. Whether the capsule created is idempotent, and whether it can be run simultaneously depend greatly on the state in which the capsule is created.

In the case where a logged-in user operates as *root*, he can modify the persistent state of the operating system. The changes are obviously reflected in the capsule containing the full user state. The user can log out, thus cleaning out the user state, then save a capsule at the log-in state with operating system modifications. Finally, the user can even initiate a shut-down which would then create a boot image with modifications.

In this section, we will discuss how we implement the various primitives as discussed in Section 3. The discussion on the policies will be discussed later in Section 5.

4.3 Implementing the Capsule Primitives

4.3.1 Creation, docking and undocking

Virtualization of practically all the platform resources names is supported directly by the VMM. These include the various i/o devices and network connections. More problematic are the network connections of applications in the capsule. Approaches to preserving network names across migration

include DHCP with NAT, and Mobile IP. DHCP with NAT allows rebinding of the IP address of the VM capsule to a new address, but all the network connections are torn down and will have to be re-established. Mobile IP [4], on the other hand, allows rebinding of both the IP address and network connections (using a proxy), and so does not need the network connections to be torn down. Currently our implementation uses the former technique.

In the case where the platform initiates a capsule creation, the platform should send a signal to the operating system to "hibernate", not unlike a machine sending a signal to the OS for power management.

The operations to undock a computation and write out a capsule map directly to the "suspend" operation provided by GSX. The suspend feature writes out the physical memory and hardware state of the VM capsule to stable storage and stops the execution of the VM. The resume feature reloads the state of the capsule from the stable storage and continues execution. If we wish to just write out a capsule, without stopping the computation, we need to initiate a "suspend" operation and follow it with a "resume" operation on the same platform immediately. Finally, we can dock a capsule by invoking "resume" operation on the desired platform.

4.3.2 Cloning

We envision that many users will be running the same login-capsule prepared by some professional. Especially in the case of a PC operating system, where a user has to be given administrator privileges to accomplish normal computational tasks, each will be given a clone of the original so that they can modify the state for their purposes. Cloning is therefore a common operation. We optimize the cloning operation by using the copy-on-write(COW) feature of virtual disks in the VMM. Thus, we only have to keep the differences between the contents when cloning.

4.4 Performance

We performed preliminary measurements of the performance of VM capsules (with VMware VMM as the platform) using some simple application benchmarks. These measurements are taken from a PC with a 2 GHz Pentium 4 and 2 GBytes of memory. These benchmarks consist of a few of the common tasks that are performed on a day-to-day basis inside a Linux environment. These tasks range from booting the Linux OS to downloading a file over the network. The complete list of tasks is shown in Table 4.4. To perform these measurements, exactly the same Linux environment was setup on a machine as well inside a VM capsule. The VM capsule was executing on hardware identical to the machine, and the systems were isolated from the network (except in the case of file download). The results are shown in Table 4.4.

From the above results, we can observe that the tasks in the VM capsule execute on an average between 30% to 70% slower (except for shutdown and search). We believe that advantages of easier maintenance, mobility, and security of compute capsules, along with the rapid advances in hardware performance outweigh the above slowdown caused by virtualization.

Application benchmark	Hardware (seconds)	VM Capsule (seconds)
boot linux	25.93	44.91
untar linux src	3.79	6.74
search linux src	14.06	43.22
compile linux src	268.74	349.31
start gnome wm	7.93	10.55
start mozilla browser	3.83	5.41
shutdown linux	15.26	16.91
download large file	100.4	155.2

Table 1: Application Benchmark Results

5. THE COLLECTIVE SYSTEM

The concept of capsules can be used to immediate advantage in our current computing environment, which consists of mainly workstations and laptops. We can use the idea to manage our home machines or our laptops, or to create a set of distributed servers out of our current desktop machines. Our ultimate vision for capsules is to use them to develop a global compute utility, which we call the Collective.

5.1 System Architecture

The design of the Collective System consists of the following components: compute servers, storage servers, brokers, authentication servers and user terminals. A user accesses the utility through any user terminal connected to the internet, presents his credentials to the authentication server, then contacts a broker which assigns the user's task to a compute server. The user's data are stored in storage servers. Each of these components may run on a separate machine, as would be the case in an compute utility design; however, they may also all run on the same machine, say, in the case of managing a laptop. We will further elaborate on each of these components below.

Authentication servers. The authentication servers securely introduce various principals in the system to each other. The parties can include servers as well as users. The system will support on-line authentication services like Kerberos. This allows the Collective to integrate into current security systems and use current principals. The Collective will also support public key certificates. Certificates have a potential to decrease the load on the authentication service by allowing secure connections to be established without communication to an intermediary.

Brokers. User requests to run capsules get sent to brokers, which are in charge of dispensing tasks to the servers it oversees. The brokers try to distribute load evenly across the servers and implement admission control if so desired. Brokers may operate hierarchically, querying other brokers for resources. For example, users may run a broker that schedules capsules according to user's needs on price, network latency, network bandwidth, CPU speed, etc. We envision the user's broker negotiating with various services' brokers for the best deal.

Compute servers. Compute servers receive requests to run capsules from a broker. Compute servers besides running the capsules, provide all the capsule manipulation operations. They include creating, undocking, and cloning capsules. All I/O from user devices (keyboard, display, mouse, CD-ROM drive) is directed from user terminals to compute servers.

Storage servers. Storage is centralized in our system architecture for manageability. The data stored contains not only just usage data but also the capsules.

User I/O. Users can remotely access the Collective via any machine connected to the network. It has been demonstrated that remote high-fidelity multimedia displays can be achieved on a switched commodity LAN network[15]. Various remote display protocols have been developed and tailored to different network bandwidths [2]. The use of capsules in a utility offers us the possibility of migrating capsules to a server close to the user, thus offering better interactive response.

5.2 Advantages of Capsules in the Collective

In addition to improving the integrity and manageability of the utility, capsules also accrue the following benefits to the Collective.

5.2.1 Scalability and reliability

The use of capsules is critical to the scalability and reliability of the utility. The individual capsules are isolated from each other, thus failure in one capsule will not affect the other. Changes to the system are made only on a cloned version of the master copy, and thus a user's customizations and modifications of the system do not affect another's.

Because each capsule's name space is self-contained, the collection of servers work in a loosely-coupled manner. This is very different from previous research on distributed operating systems, where the system presents a single-system image to the user. As such, the machines in the system are much more tightly-coupled, working on the same global name space, thus limiting their scalability and reliability. Despite all the effort in this area, distributed operating systems are not widely used in practice.

5.2.2 Hardware management

Since compute servers are managed as a cache of capsules, new machines can be added as servers by simply by installing a uniform trusted computing base that runs capsules. Capsules can be migrated onto new machines as each capsule holds the entire environment that needs to be run. We can take machines down for maintenance easily by simply migrating the capsules to other machines.

Like a cache, a compute server under heavy load can "discard" a compute capsule, by first serializing its state and saving the state away. The compute capsule may by resumed by another compute server. And if the computation has been idle for a long time, then the capsule may simply be left in storage.

5.2.3 Supporting mobility

In a global utility model, a user's capsule may be moved to a compute server close to the user, which may be a local node in the utility, a home machine or a laptop. Thus, even highly interactive graphical applications can be run on the utility with excellent interactive response time. If the user moves from one location to another, as is the case for example, when he commutes to and from work, then the capsule can be transferred while he is in transit. If the capsule cannot be prefetched, it can be demand paged in to minimize the startup overhead.

5.2.4 Fault tolerance

The ability to write out a current state of the computation makes it possible for a utility to provide fault tolerance in an application-independent manner. As noted above, if we wish to undo an operation, it is important the capsule is created at a point where it can be re-executed. Thus, we leave it to the user to initiate these checkpoints. Once the capsule is created, the user can then choose to roll back the computation by re-executing the appropriate capsule.

5.3 Implementation

We have developed a prototype of a compute utility, which we call the Collective System. The system provides a web interface to the user for authentication and management of his capsules. The system includes a broker that uses a simple heuristic to dispense capsules to one of the two servers we have in the system. The storage server is an NFS server, and currently the web interface uses a simple password authentication. It will be replaced by Kerberos in the future. For remote display, we are using the VMware remote console technology. Our system for the most part has been implemented in Java. Via the web interface, the user can dock a capsule (on a particular server if desired), create a capsule of the current computation, clone a capsule, and undock a capsule.

The implementation of cloning makes use of the copy-on-write feature provided by VMware virtual disks. The transfer of state for migration is performed via NFS (i.e., demand paging with no caching). This unoptimized implementation of migration takes about 30 seconds to migrate a running a Linux log-in capsule between two compute servers on the same 100Mbps LAN. This performance seems reasonable and we expect it to improve in the future with further optimizations.

5.4 Preliminary Experience

We have started to use the Collective system in our research projects, and have found the system useful. For example, this addresses one of the problems that the Stanford SUIF compiler research group has encountered. While most of the group members use Unix, a couple of the projects (e.g. the java joeq virtual machine project) have been developed on the Windows environment. By creating capsules for these projects in the utility, other group members can easily try out the system without having to first install all the necessary software on their own machine.

We have encountered a few interesting issues while setting up capsules. First, since we are using NAT for networking the capsules, mounting NFS over NAT into the capsules posed a problem. This is because NFS expects client requests from a privileged port, while the NAT gateway would not always assign a privileged port to the forwarded client connections. To circumvent this, we had to modify the NAT code in the Linux kernel to assign privileged ports.

Second, we wanted the user to go through the authentication server just once and not have to log into the individual capsules. To do this for Linux, we injected the user information into the capsule using NFS. We also modified the Linux login program to read this information, and log the user in without the need for a second authentication.

Third, we found that the virtualization of the timer interrupt by VMware causes the interrupts to be delayed or lost under heavy load. This could cause the capsule's OS

to "lose" time, causing the timing measurements using the OS' system calls to be inaccurate. To get around this problem, we measured all times using the Pentium cycle counter, since VMware does not virtualize the instruction to read the cycle counter.

Here are some preliminary measurements obtained on the Collective System. The size of the main virtual disk for a Linux capsule is about 1.2GB, while the minimum size of the copy-on-write disk is 3MB. The latency of migration of a log-in Linux capsule with 128MB main memory and 154MB disk, between two machines on the same 100Mbps LAN, is about 28.9 seconds. This time is divided into 13.0 seconds for undocking the capsule, and 15.9 seconds for docking it on the new machine.

6. RELATED WORK

Capsules can be implemented at various levels, at the operating system, the process level, as well as the object level. We briefly discuss each of these different approaches.

6.1 Operating System Level Migration

The term *capsule* was introduced earlier by one of the authors and Brian Schmidt[14]. In this work, capsules were implemented at the operating system. Hardware resources were virtualized by modifying the operating system kernel. The advantage is that the capsules are much lighter weight than virtual-machine based capsules. The disadvantage, however, is that the operating system interface is not well-defined, causing it a non-trivial engineering task to ensure that all resources are virtualized properly. In addition, unlike the virtual-machine level, the same work has to be applied to each operating system we are interested in. For example, since the prototype developed was based on Solaris, it is not possible to run PC software under this model.

6.2 User-Level Process Migration

Several systems have been developed to support process migration at the user level. Examples include Condor [12], libckpt [11], and CoCheck [17]. These systems are primarily intended to support batch applications on a cluster of machines. An active process can migrate throughout the system, and in some cases a checkpoint facility provides a mechanism for restarting it after a failure. These systems have the benefit of running on top of unmodified operating systems, but other than migration, they do not provide the functionality we require. In addition, because there is no kernel support for process migration, these systems offer incomplete solutions. Migratory processes are required to be well-behaved, which means that they cannot use host-dependent services (such as process identifiers), inter-process communication, or special devices. This severely restricts the class of applications that can utilize such systems.

6.3 Object-Based Approaches

A variety of systems have been developed to support distributed computations using migrating objects and remote method invocation. Examples include Legion [6], Emerald [9], and Rover [8]. These systems are implemented as programming languages or middleware toolkits. Some systems support active objects, which include a thread of control. These approaches require explicit programmer control to use the mobility features, i.e. checkpoint/restart methods must typically be provided by the programmer. Although this offers greater efficiency in the state that is recorded, it duplicates functionality in every program, and it requires all applications to be re-written in the new programming style. This means that there is no legacy support, which is one of our major requirements.

6.4 Machine Clusters with a Single System Image

Many distributed operating systems have been developed which provide a single system image across a cluster of machines, i.e. many machines appear as a single unit with a single operating system. Examples include Sprite [13], V [1], Amoeba [16], and Solaris MC [10]. With varying levels of completeness, these systems provide facilities for migrating individual processes between physical machines in the cluster. This is accomplished by carefully designing the kernels to provide a global namespace and to support location-transparent execution. When a process is migrated, the memory image is moved to the target system, and a variety of optimization techniques are employed. Other process state (such as IPC, device status, and open files) is typically handled by forwarding requests to a home node on which the process originated. Message-based systems, such as V, leave forwarding addresses behind so that communication with system services will remain intact.

A single system image across machines simplifies system management and reduces administration cost, but these types of systems do not scale to Internet dimensions or easily support multiple administrative domains. Although individual processes can be migrated, groups cannot be treated as a unit for mobility or resource scheduling. Further, there are no mechanisms for suspending processes to off-line storage, and there is no isolation between users.

7. CONCLUSIONS

This paper argues for the abstraction of a compute capsule, which is an encapsulation of software, in both actively running and non-active environments. A compute capsule has several important properties: it is not tied to a machine it is run on; it is persistent and portable, and can be treated just like data; capsules are isolated from each other because they have separate name spaces. By creating capsules using virtual machine technology, operating systems are encapsulated just like any other user software. Such capsules can be used to reduce system administration cost, enhance security and support mobility in existing computing environments.

We have presented the design of a compute utility, which is a new system architecture based on the concept of capsule. The properties of capsules make it feasible to develop a reliable and scalable compute utility: Failures in users' capsules are naturally isolated from each other; capsules can be migrated easily to support mobility, load distribution, as well as maintenance; capsules can also be used as checkpoints for fault recovery. We have prototyped some of the basic functions in the Collective System.

8. REFERENCES

[1] D. R. Cheriton. The V distributed system. *Communications of the ACM*, 31(3):314–333, 1988.

[2] B. O. Christiansen, K. E. Schauser, and M. Munke. TCC: Thin client compression. In *Proceedings of the IEEE Data Compression Conference*, March 2001.

[3] F. Douglis and J. Ousterhout. Transparent process migration: Design alternatives and the sprite implementation. *Software – Practice and Experience*, 21(8):757–785, August 1991.

[4] C. Perkins (Editor). "IP mobility support" internet request for comments 2002, October 1996.

[5] R. P. Goldberg. Survey of virtual machine research. *Computer*, 7(6):34–45, June 1974.

[6] A. Grimshaw, A. Ferrari, F. Knabe, and M. Humphrey. Legion: An operating system for wide-area computing. Technical Report CS-99-12, Dept. of Computer Science, University of Virginia, March 1999.

[7] IA-32 Intel architecture software developer's manual volumes 1-3. http://developer.intel.com/design/pentium4/manuals/.

[8] A. Joseph, J. Tauber, and M. Kaashoek. Mobile computing with the rover toolkit. *IEEE Transactions on Computers*, 46(3):337–352, March 1997.

[9] E. Jul, H. Levy, N. Hutchinson, and A. Black. Fine-grained mobility in the emerald system. *ACM Transaction on Computer Systems*, 6(1):109–133, February 1988.

[10] Y. Khalidi, J. Bernabeu, V. Matena, K. Shirriff, and M. Thadani. Solaris mc: A multi-computer os. In *Proceedings of the USENIX 1996 Annual Technical Conference*, pages 191–203, January 1996.

[11] J. Plank M. Beck G. Kingsley and K. Li. Libckpt: Transparent checkpointing under unix. In *Proceedings of the USENIX Winter 1995 Technical Conference*, pages 213–224, January 1995.

[12] M. Litzkow, M. Livny, and M. Mutka. Condor – a hunter of idle workstations. In *Proceedings of the 8th International Conference on Distributed Computing Systems*, pages 104–111, June 1988.

[13] J. Ousterhout, A. Cherenson, F. Douglis, M. Nelson, and B. Welch. The sprite network operating system. *IEEE Computer*, 21(2):23–36, February 1988.

[14] B. K. Schmidt. *Supporting Ubiquitous Computing with Stateless Consoles and Computation Caches*. PhD thesis, Computer Science Department, Stanford University, August 2000.

[15] B. K. Schmidt, M. S. Lam, and J. D. Northcutt. The interactive performance of SLIM: a stateless, thin-client architecture. In *Proceedings of the 17th ACM Symposium on Operating Systems Principles*, pages 32–47, December 1999.

[16] C. Steketee, P. Socko, and B. Kiepuszewski. Experiences with the implementation of a process migration mechanism for amoeba. In *Proceedings of Australasian Computer Science Conference*, pages 140–148, January 1996.

[17] G. Stellner. CoCheck: Checkpointing and process migration for MPI. In *Proceedings of the 10th International Parallel Processing Symposium*, pages 526–531, April 1996.

[18] "GSX server", white paper. http://www.vmware.com/pdf/gsx_whitepaper.pdf.

[19] Wintel architecture specifications. http://www.microsoft.com/hwdev/specs/.