The Collective: A Cache-Based System Management Architecture

Ramesh Chandra Nickolai Zeldovich Constantine Sapuntzakis Monica S. Lam
Computer Science Department
Stanford University
Stanford, CA 94305
{rameshch, nickolai, csapuntz, lam}@cs.stanford.edu

Abstract

This paper presents the Collective, a system that delivers managed desktops to personal computer (PC) users. System administrators are responsible for the creation and maintenance of the desktop environments, or *virtual appliances*, which include the operating system and all installed applications. PCs run client software, called the *virtual appliance transceiver*, that caches and runs the latest copies of appliances locally and continuously backs up changes to user data to a network repository. This model provides the advantages of central management, such as better security and lower cost of management, while leveraging the cost-effectiveness of commodity PCs.

With a straightforward design, this model provides a comprehensive suite of important system functions including machine lockdown, system updates, error recovery, backups, and support for mobility. These functions are made available to all desktop environments that run on the x86 architecture, while remaining protected from the environments and their many vulnerabilities. The model is suitable for managing computers on a LAN, WAN with broadband, or even computers occasionally disconnected from the network like a laptop. Users can access their desktops from any Collective client; they can also carry a bootable drive that converts a PC into a client; finally, they can use a remote display client from a browser to access their desktop running on a remote server.

We have developed a prototype of the Collective system and have used it for almost a year. We have found the system helpful in simplifying the management of our desktops while imposing little performance overhead.

1 Introduction

With the transition from mainframe computing to personal computing, the administration of systems shifted from central to distributed management. With mainframes, professionals were responsible for creating and maintaining the single environment that all users accessed. With the advent of personal computing, users got to define their environment by installing any software that fit their fancy. Unfortunately, with this freedom also came the tedious, difficult task of system management: purchasing the equipment and software, installing the software, troubleshooting errors, performing upgrades and re-installing operating systems, performing backups, and finally recovering from problems caused by mistakes, viruses, worms and spyware.

Most users are not professionals and, as such, do not have the wherewithal to maintain systems. As a result, most personal computers are not backed up and not up to date with security patches, leaving users vulnerable to data loss and the Internet vulnerable to worms that can infect millions of computers in minutes [16]. In the home, the challenges we have outlined above lead to frustration; in the enterprise, the challenges cost money.

The difficulties in managing distributed PCs has prompted a revival in interest in thin-client computing, both academically [15, 8] and commercially [2]. Reminiscent of mainframe computing, computation is performed on computers centralized in the data center. On the user's desk is either a special-purpose remote display terminal or a general-purpose personal computer running remote display software. Unfortunately, this model has higher hardware costs and does not perform as well. Today, the cheapest thin clients are PCs without a hard disk, but unlike a stand-alone PCs, a thin client also needs a server that does the computation, increasing hardware costs. The service provider must provision enough computers to handle the peak load; users cannot improve their computing experience by going to the store and buying a faster computer. The challenge of managing multiple desktops remains, even if it is centralized in the data center. Finally, remote display, especially over slower links, cannot deliver the same interactivity as local applications.

1.1 Cache-based System Management

This paper presents a cache-based system management model that combines the advantages of centralized management while taking advantage of inexpensive PCs. Our model delivers instances of the same software environments to desktop computers automatically, thereby amortizing the cost of the management. This design trades off users' ability to customize their own environment in return for uniformity, scalability, better security and lower cost of management.

In our model, we separate the state in a computer into two parts: system state and user state. The system state consists of an operating system and all installed applications. We refer to the system state as an *appliance* to emphasize that only the administrator is allowed to modify the system function; thus, to the user the system state defines a fixed function, just like any appliance. Note that these appliances are *virtual appliances* because unlike real appliances, they do not come with dedicated hardware. User state consists of a user's profile, preferences, and data files.

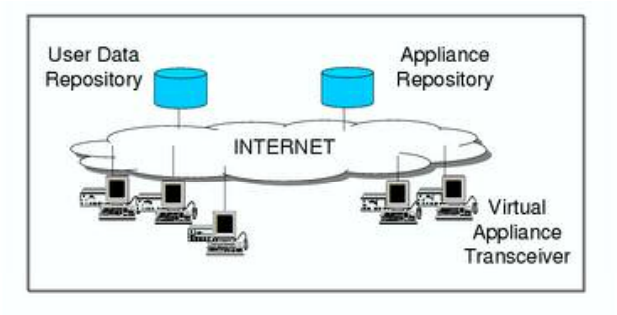

Figure 1: Architecture of the Collective system

In the cache-based system management model, appliances and user state are stored separately in network-accessible *appliance repositories* and *data repositories*, as shown in Figure 1. PCs in this model are fixed-function devices called *virtual appliance transceivers* (VATs).

In this model, a user can walk up to any of these clients, log in, and get access to any appliance he is entitled to. The VAT performs the following functions:

1. authenticates users.
2. fetches and runs the latest copies of appliances locally.
3. backs up user state changes to the data repository continuously; only administrators are allowed to update appliances in the appliance repository.
4. optimizes the system by managing a cache to reduce the amount of the data that needs to be fetched over the network.

Because appliances run locally, rather than on some central server, users experience performance and interactivity similar to their current PC environment; this approach provides the benefits of central management while leveraging commodity PCs.

1.2 System Highlights

We have developed a prototype based on this model which we call the Collective [14, 12]. By using x86 virtualization technology provided by the VMware GSX Server [20], the Collective client can manage and run most software that runs on an x86 PC. We have used the system daily since June 2004. Throughout this period, we have extended our system and rewritten parts of it several times. The system that we are describing is the culmination of many months of experience. This paper presents the design and rationale for cache-based system management, as well as the detailed design and implementation of the Collective. We also measure our prototype and describe our experiences with it.

Our cache-based system management model has several noteworthy characteristics. First, the VAT is a separate layer in the software stack devoted to management. The VAT is protected from the appliances it manages by the virtual machine monitor, increasing our confidence in the VAT's security and reliability in the face of appliance compromise or malfunction. Finally, the VAT automatically updates itself, requiring little to no management on the part of the user.

Second, the system delivers a comprehensive suite of critical system management functions automatically and efficiently, including disk imaging, machine lockdown, software updates, backups, system and data recovery, mobile computing, and disconnected operation, through a simple and unified design based on caching. Our cache design is kept simple with the use of a versioning scheme where every data item is referred to by a unique name. In contrast, these management functions are currently provided by a host of different software packages, often requiring manual intervention.

Third, our design presents a uniform user interface, while providing performance and security, across computers with different network connectivities and even on computers not running the VAT software. The design works on computers connected to a LAN or WAN with broadband bandwidth; it even works when computers are occasionally disconnected. It also enables a new mobility model, where users carry a portable storage device such as an 1.8-inch disk. With the disk, they can boot any compatible PC and run the VAT. Finally, in the case where users can only get access to a conventional PC, they can access their environment using a browser, albeit only at remote access speeds.

In contrast, without a solution that delivers performance across different network connectivities, enterprises often resort to using a complicated set of techniques as a compromise. In a large organization, users in the main campus may use PCs since they have enough IT staff to manage them; remote offices may use thin clients instead, trading performance for reduced IT staff. Employees may manage their home PCs, installing corporate applications, and accessing corporate infrastructure via VPN. Finally, ad hoc solutions may be used to manage laptops, as they are seldom connected to the network unless they are in use.

The combination of centralized management and cheap PCs, provided by the Collective, offers a number of practical benefits. This approach lowers the management cost at the cost of a small performance overhead. More importantly, this approach improves security by keeping software up to date and locking down desktops. Recovery from failures, errors, and attacks is made possible by continuous backup. And, the user sees the same computing environment at the office, at home, or on the road.

1.3 Paper Organization

The rest of the paper is structured as follows. Section 2 presents an overview of the system. Section 3 discusses the design in more detail. We present quantitative evaluation in section 4 and qualitative user experience in Section 5. Section 6 discusses the related work and Section 7 concludes the paper.

2 System Overview

This section provides an overview of the Collective. We start by presenting the data types in the system: appliances, repositories and subscriptions. We then show how the Collective works from a user's perspective. We describe how the Collective's architecture provides mobility and management functions and present optimizations that allow the Collective to perform well under different connectivities.

2.1 Appliances

An appliance encapsulates a computer state into a virtual machine which consists of the following:

- *System disks* are created by administrators and hold the appliance's operating system and applications. As part of the appliance model, the contents of the system disk at every boot are made identical to the contents published by the administrator. As the appliance runs, it may mutate the disk.
- *User disks* hold persistent data private to a user, such as user files and settings.
- *Ephemeral disks* hold user data that is not backed up. They are used to hold ephemeral data such browser caches and temporary files; there is little reason to incur additional traffic to back up such data over the network.
- A *memory image* holds the state of a suspended appliance.

2.2 Repositories

Many of the management benefits of the Collective derive from the versioning of appliances in network repositories. In the Collective, each appliance has a repository; updates to that appliance get published as a new version in that repository. This allows the VAT to automatically find and retrieve the latest version of the appliance.

To keep consistency simple, versions are immutable. To save space and to optimize data transfer, we use copy-on-write (COW) disks to express the differences between versions.

2.3 Subscriptions

Users have accounts, which are used to perform access control and keep per-user state. Associated with a user's account is the user's Collective profile which exists on network storage. In the user's Collective profile is a list of appliances that the user has access to. When the user first runs an appliance, a *subscription* is created in the profile to store the user state associated with that appliance.

To version user disks, each subscription in the user's Collective profile contains a repository for the user disks associated with the appliance. The first version in the repository is a copy of an initial user disk published in the appliance repository.

Other state associated with the subscription is stored only on storage local to the VAT. When an appliance starts, a COW copy of the system disk is created locally. Also, an ephemeral disk is instantiated, if the appliance requires one but it does not already exist. When an appliance is suspended, a memory image is written to the VAT's storage.

Since we do not transfer state between VATs directly, we cannot migrate suspended appliances between VATs. This is mitigated by the user's ability to carry their VAT with them on a portable storage device.

To prevent the complexity of branching the user disk, the user should not start a subscribed appliance on a VAT while it is running on another VAT. So that we can, in many cases, detect this case and warn the user, a VAT will attempt to acquire and hold a lock for the subscription while the appliance is running.

2.4 User Interface

The VAT's user interface is very simple–it authenticates the user and allows him to perform a handful of operations on appliances. On bootup, the VAT presents a Collective log-in window, where a user can enter his username and password. The system then presents the user with a list of appliances that he has access to, along with their status. Choosing from a menu, the user can perform any of the operations on his appliances:

- *start* boots the latest version of the appliance, or if a suspended memory image is available locally, resumes the appliance.
- *stop* shuts down the appliance.
- *suspend* suspends the appliance to a memory image.

- *reset* destroys all ephemeral disks and the memory image but retains the user disks
- *delete* destroys the subscription including the user disks
- *user disk undo* allows the user to go back to a previous snapshot of their user disk.
- *publish* allows the administrator to save the current version of the system disk as the latest version of the appliance.

When a user starts an appliance, the appliance takes over the whole screen once it runs; at any time, the user can hit a hot key sequence (Ctrl-Alt-Shift) and return to the list of appliances to perform other operations. The VAT user interface also indicates the amount of data that remains to be backed up from the local storage device. When this hits zero, the user can log out of the VAT and safely shift to using the virtual machines on another device.

2.5 Management Functions

We now discuss how the various management functions are implemented using caching and versioning.

2.5.1 System Updates

Desktop PCs need to be updated constantly. These upgrades include security patches to operating systems or installed applications, installations of new software, upgrades of the operating system to a new version, and finally re-installion of the operating system from scratch. All software upgrades, be they small or big, are accomplished in our system with the same mechanism. The system administrator prepares a new version of the appliance and deposits it in the appliance repository. The user gets the latest copy of the system disks the next time they reboot the appliance. The VAT can inform the user that a new version of the system disk is available, encouraging the user to reboot, or the VAT can even force a reboot to disallow use of the older version. From a user's standpoint, upgrade involves minimal work; they just reboot their appliances. Many software updates and installations on Windows already require the computer to be rebooted to take effect.

This update approach has some advantages over package and patch systems like yum [24], RPM [1], Windows Installer [22], and Windows Update. First, patches may fail on some users' computers because of interactions with user-installed software. Our updates are guaranteed to move the appliance to a new consistent state. Users running older versions are unaffected until they reboot. Even computers that have been off or disconnected get the latest software updates when restarted; with many patch management deployments, this is not the case. Finally, our update approach works no matter how badly the software in the appliance is functioning, since the VAT is protected from the appliance software. However, our model requires the user to subscribe to an entire appliance; patching works with individual applications and integrates well in environments where users mix and match their applications.

Fully automating the update process, the VAT itself is managed as an appliance. It automatically updates itself from images hosted in a repository. This is described in more detail in Section 3. By making software updates easy, we expect environments to be much more up to date with the Collective and hence more secure.

2.5.2 Machine Lockdown

Our scheme locks down user desktops because changes made to the system disks are saved in a new version of the disk and are discarded when the appliance is shut down. This means that if a user accidentally installs undesirable software like spyware into the system state, these changes are wiped out.

Of course, undesirable software may still install itself into the user's state. Even in this case, the Collective architecture provides the advantage of being able to reboot to an uncompromised system image with uncompromised virus scanning tools. If run before accessing ephemeral and user data, the uncompromised virus scanner can stay uncompromised and hopefully clean the changed state.

2.5.3 Backup

The VAT creates a COW snapshot of the user disk whenever the appliance is rebooted and also periodically as the appliance runs. The VAT backs up the COW disks for each version to the user repository. The VAT interface allows users to roll back changes made since the last reboot or to return to other previous versions. This allows the user to recover from errors, no matter the cause, be it spyware or user error.

When the user uses multiple VATs to access his appliances without waiting for backup to complete, potential for conflicts in the user disk repository arises. The backup protocol ensures that only one VAT can upload user disk snapshots into the repository at a time. If multiple VATs attempt to upload user disks at the same time, the user is first asked to choose which VAT gets to back up into the subscription and then given the choice of terminating the other backups or creating additional subscriptions for them.

If an appliance is writing and overwriting large quantities of data, and there is insufficient network bandwidth for backup, snapshots can accumulate at the client, buffered for upload. In the extreme, they can potentially fill the disk. We contemplate two strategies to deal with accumulating snapshots: collapse multiple snapshots to-

gether and reduce the frequency of snapshots. In the worst case, the VAT can stop creating new snapshots and collapse all of the snapshots into one; the amount of disk space taken by the snapshot is then bounded by the size of the virtual disk.

2.5.4 Hardware Management

Hardware management becomes simpler in the Collective because PCs running the VAT are interchangeable; there is no state on a PC that cannot be discarded. Deploying hardware involves loading PCs with the VAT software. Provisioning is easy because users can get access to their environments on any of the VATs. Faulty computers can be replaced without manually customizing the new computer.

2.6 Optimizations for Different Network Connectivities

To reduce network and server usage and improve performance, the VAT includes a large on-disk cache, on the order of gigabytes. The cache keeps local copies of the system and user disk blocks from the appliance and data repositories, respectively. Besides fetching data on demand, our system also prefetches data into the cache. To ensure good write performance even on low-bandwidth connections, all appliance writes go to the VAT's local disk; the backup process described in section 2.5.3 sends updates back in the background.

The cache makes it possible for this model to perform well under different network connectivities. We shall show below how our system allows users to access their environments on any computer, even computers with no pre-installed VAT client software, albeit with reduced performance.

2.6.1 LAN

On low-latency, high bandwidth (e.g., 100 Mbps) networks, the system performs reasonably well even if the cache is empty. Data can be fetched from the repositories fast enough to sustain good responsiveness. The cache is still valuable for reducing network and server bandwidth requirements. The user can easily move about in a LAN environment, since it is relatively fast to fetch data from a repository.

2.6.2 WAN with Broadband

By keeping a local copy of data from the repository, the cache reduces the need for data accesses over the network. This is significant because demand-fetching every block at broadband bandwidth and latency would make the system noticeably sluggish to the user. We avoid this worst-case scenario with the following techniques. First, at the time the VAT client software is installed on the hard disk, we also populate the cache with blocks of the appliances most likely to be used. This way only updates need to be fetched. Second, the VAT prefetches data in the background whenever updates for the appliances in use are available. If, despite these optimizations, the user wishes to access an appliance that has not been cached, the user will find the application sluggish when using a feature for the first time. The performance of the feature should subsequently improve as its associated code and data get cached. In this case, although the system may try the user's patience, the system is guaranteed to work without the user knowing details about installing software and other system administration information.

2.6.3 Disconnected Operation with Laptops

A user can ensure access to a warm cache of their appliances by carrying the VAT on a laptop. The challenge here is that a laptop is disconnected from the network from time to time. By hoarding all the blocks of the appliances the user wishes to use, we can keep operating even while disconnected.

2.6.4 Portable VATs

The self-containment of the VAT makes possible a new model of mobility. Instead of carrying a laptop, we can carry the VAT, with a personalized cache, on a bootable, portable storage device. Portable storage devices are fast, light, cheap, and small. In particular, we can buy a 1.8-inch, 40GB, 4200 rpm portable disk, weighing about 2 ounces, for about $140 today. Modern PCs can boot from a portable drive connected via USB.

The portable VAT has these advantages:

1. *Universality and independence of the computer hosts*. Eliminating dependences on the software of the hosting computer, the device allows us to convert any x86 PC into a Collective VAT. This approach leaves the hosting computer undisturbed, which is a significant benefit to the hosting party. Friends and relatives need not worry about their visitors modifying their computing environments accidentally, although malicious visitors can still wreak havoc on the disks in the computers.

2. *Performance*. The cache in the portable VAT serves as a network accelerator. This is especially important if we wish to use computers on low-bandwidth networks.

3. *Fault tolerance*. Under typical operation, the VAT does not contain any indispensable state when not in use; thus, in the event the portable drive is lost or forgotten, the user gets access to his data by inserting another generic VAT and continuing to work, albeit at a slower speed.

4. *Security and privacy*. This approach does not disturb the hosting computer nor does it leave any trace

of its execution on the hosting computer. Data on the portable drive can be encrypted to maintain secrecy if the portable drive is lost or stolen. However, there is always the possibility that the firmware of the computer has been doctored to spy on the computations being performed. Trusted computing techniques [18, 5] can be applied here to provide more security; hardware could in theory attest to the drive the identity of the firmware.

2.6.5 Remote Display

Finally, in case the users do not have access to any VATs, they can access their environments using remote display. We recreate a user experience similar to the one with the VAT; the user logs in, is presented with a list of appliances and can click on them to begin using them. The appliances are run on a server and a window appears with an embedded Java remote display applet that communicates with the server.

3 Design of the VAT

In this section, we present the design of the appliance transceiver. The VAT's major challenges include running on as many computers as possible, automatically updating itself, authenticating users and running their appliances, and working well on slow networks.

3.1 Hardware Abstraction

We would like the VAT image to run on as many different hardware configurations as possible. This allows users with a bootable USB drive to access their state from almost any computer that they might have available to them. It also reduces the number of VAT images we need to maintain, simplifying our administration burden.

To build a VAT that would support a wide range of hardware, we modified KNOPPIX [6], a *Live CD* version of Linux that automatically detects available hardware at boot time and loads the appropriate Linux drivers. KNOPPIX includes most of the drivers available for Linux today.

KNOPPIX's ability to quickly auto-configure itself to a computer's hardware allows the same VAT software to be used on many computers without any per-computer modification or configuration, greatly simplifying the management of an environment with diverse hardware. We have found only one common situation where the VAT cannot configure itself without the user's help: to join a wireless network, the user may need to select a network and provide an encryption key.

If a VAT successfully runs on a computer, we can be reasonably sure the appliances running on it will. The appliances run by the VAT see a reasonably uniform set of virtual devices; VMware GSX server takes advantage of the VAT's device drivers to map these virtual devices to a wide range of real hardware devices.

3.2 User Authentication

To maintain security, users must identify themselves to the VAT and provide credentials that will be used by the VAT to access their storage on their behalf. As part of logging in, the user enters his username and password. The VAT then uses SSH and the password to authenticate the user to the server storing the user's Collective profile. To minimize the lifetime of the user's password in memory, the VAT sets up a key pair with the storage server so that it can use a private key, rather than a password, to access storage on behalf of the user.

Disconnected operation poses a challenge, as there is no server to contact. However, if a user has already logged in previously, the VAT can authenticate him. When first created, the private key mentioned in the previous paragraph is stored encrypted with the user's password. On a subsequent login, if the password entered successfully decrypts the private key, the user is allowed to login and access the cached appliances and data.

3.3 VAT Maintenance

As mentioned earlier, the VAT is managed as an appliance, and needs zero maintenance from the end user; it automatically updates itself from a repository managed by a VAT administrator.

For most problems with the VAT software, a reboot restores the software to a working state. This is because the VAT software consists largely of a read-only file system image. Any changes made during a session are captured in separate file systems on a ramdisk. As a result, the VAT software does not drift from the published image.

All VATs run an update process which checks a repository for new versions of the VAT software and downloads them to the VAT disk when they become available. To ensure the integrity and authenticity of the VAT software, each version is signed by the VAT publisher. The repository location and the public key of publisher are stored on the VAT disk.

After downloading the updated image, the update process verifies the signature and atomically changes the boot sector to point to the new version. To guarantee progress, we allocate enough room for three versions of the VAT image: the currently running version, a potentially newer version that is pointed to by the boot sector which will be used at next reboot, and an even newer, incomplete version that is in the process of being downloaded or verified.

The VAT image is about 350MB uncompressed; downloading a completely new image wastes precious network capacity on broadband links. To reduce the size to about 160MB, we use the `cloop` tools that come with KNOPPIX to generate a compressed disk image; by using the `cloop` kernel driver, the VAT can mount the root file system from the compressed file system directly. Even so, we expect most updates to touch only a few files; transferring

an entire 160MB image is inefficient. To avoid transferring blocks already at the client, the update process uses rsync; for small changes, the size of the update is reduced from 160MB to about 10MB.

3.4 Storage Access

Our network repositories have the simplest layout we could imagine; we hope this will let us use a variety of access protocols. Each repository is a directory; each version is a subdirectory whose name is the version number. The versioned objects are stored as files in the subdirectories. Versions are given whole numbers starting at 1. Since some protocols (like HTTP) have no standard directory format, we keep a `latest` file in the repository's main directory that indicates the highest version number.

To keep consistency simple, we do not allow a file to be changed once it has been published into a repository. However, it should be possible to reclaim space of versions that are old; as such, files and versions can be deleted from the repository. Nothing prevents the deletion of an active version; the repository does not keep track of the active users.

When reading from network storage, we wanted a simple, efficient protocol that could support demand paging of large objects, like disk images. For our prototype, we use NFS. NFS has fast, reliable servers and clients. To work around NFS's poor authentication, we tunnel NFS over SSH.

While demand paging a disk from the repository, we may become disconnected. If a request takes too long to return, the disk drivers in the appliance's OS will timeout and return an I/O error. This can lead to file system errors which can cause the system to panic. In some cases, suspending the OS before the timeout and resuming it once we have the block can prevent these errors. A better solution is to try to make sure this does not happen in the first place by aggressively caching blocks; we will discuss this approach more in Section 3.6.

We also use NFS to write to network storage. To atomically add a new version to a repository, the VAT first creates and populates the new version under a one-time directory name. As part of the process, it places a nonce in the directory. The VAT then renames the directory to its final name and checks the nonce to see if it succeeded.

We would also like to set the priority of the writes to user data repositories with respect to other network traffic. Currently, the VAT approximates this by mounting the NFS server again on a different mount point; this in turn uses a separate TCP connection, which is given a different priority. Another consideration is that when an NFS server is slow or disconnected, the NFS in-kernel client will buffer writes in memory, eventually filling memory with dirty blocks and degrading performance. To limit the quantity of dirty data, the VAT performs an fsync after every 64 kilobytes of writes to the user data repository.

3.5 Caching

Our cache is designed to mask the high latency and low bandwidth of wide-area communication by taking advantage of large, persistent local storage, like hard disks and flash drives. At the extreme, the cache allows the client to operate disconnected.

COW disks can be gigabytes in size; whole file caching them would lead to impractical startup times. On the other hand, most of the other data in our system, like the virtual machine description, is well under 25 kilobytes. As a result, we found it easiest to engineer two caches: a small object cache for small data and meta-data and a COW cache for COW disk blocks.

To simplify disconnected operation, small objects, like the user's list of appliances or the meta-data associated with a repository, are replicated in their entirety. All the code reads the replicated copy directly; a background process periodically polls the servers for updates and integrates them into the local replica. User data snapshots, which are not necessarily small, are also stored in their entirety before being uploaded to the server.

The COW cache is designed to cache immutable versions of disks from repositories; as such the name of a disk in the cache includes an ID identifying the repository (currently the URL), the disk ID (currently the disk file name), and the version number. To name a specific block on a disk, the offset on disk is added. Since a data block can change location when COW disk chains are collapsed, we use the offset in the virtual disk, not the offset in the COW disk file.

One of the challenges of on-disk data structures is dealing with crashes, which can happen at any time, leading to partial writes and random data in files. With a large cache, scanning the disk after a crash is unattractive. To cope with partial writes and other errors introduced by the file system, each 512-byte sector stored in the cache is protected by an MD5 hash over its content and its address. If the hash fails, the cache assumes the data is not resident in the cache.

A traditional challenge with file caches has been invalidation; however, our cache needs no invalidation protocol. The names used when storing and retrieving data from the cache include the version number; since any given version of an object is immutable, no invalidation is necessary.

Our cache implementation does not currently make an effort to place sequential blocks close to each other on disk. As a result, workloads that are optimized for sequential disk access perform noticeably slower with our cache, due to the large number of incurred seeks. One such common workload is system bootup; we have implemented a *bootup block optimization* for this case. Since the block

access pattern during system bootup is highly predictable, a trace of the accessed blocks is saved along with each virtual appliance. When an appliance is started, the trace is replayed, bringing the blocks into the buffer cache before the appliance OS requests them. This optimization significantly reduces bootup time. A more general version of this technique can be applied to other predictable block access patterns, such as those associated with starting large applications.

3.6 Prefetching

To minimize cache misses, the VAT runs a prefetcher process to fetch useful appliance blocks in the background. The prefetcher checks for updates to appliances used by the VAT user, and populates the cache with blocks from the updated appliances. One optimization is to prioritize the blocks using access frequency so that the more important data can be prefetched first.

The user can use an appliance in disconnected mode by completely prefetching a version of an appliance into the cache. The VAT user interface indicates to the user what versions of his appliances have been completely prefetched. The user can also manually issue a command to the prefetcher if he explicitly wants to save a complete version of the appliance in the cache.

The prefetcher reduces interference with other processes by rate-limiting itself. It maintains the latencies of recent requests and uses these to determine the extent of contention for network or disk resources. The prefetcher halves its rate if the percentage of recent requests experiencing a high latency exceeds a threshold; otherwise, it doubles its rate when it finds a large percentage experience a low latency. If none of these apply, it increases or decreases request rate by a small constant based on the latency of the last request.

Prefetching puts spare resources to good use by utilizing them to provide better user experience in the future: when a user accesses an appliance version for the first time, it is likely that the relevant blocks would already be cached. Prefetching hides network latency from the appliance, and better utilizes network bandwidth by streaming data rather than fetching it on demand.

4 Evaluation

We provide some quantitative measurements of the system to give a sense of how the system behaves. We perform four sets of experiments. We first use a set of benchmarks to characterize the overhead of the system and the effect of using different portable drives. We then present some statistics on how three appliances we have created have evolved over time. Next, we evaluate prefetching. We show that a small amount of data accounts for most of the accesses, and that prefetching can greatly improve the responsiveness of an interactive workload. Finally, we study the feasibility of continuous backups.

4.1 Run-Time Performance

We first establish some basic parameters of our system by running a number of benchmarks under different conditions. All of the experiments, unless noted otherwise, are run on 2.4GHz Pentium IV machines with 1GB of memory and a 40GB Hitachi 1.8" hard drive connected via Prolific Technology's PL-2507 USB-to-IDE bridge controller. VAT software running on the experimental machines is based on Linux kernel 2.6.11.4 and VMware GSX server version 3.1. The file server is a 2.4GHz Pentium IV with 1GB of memory and a Linux software RAID, consisting of four 160GB IDE drives. We use FreeBSD's *dummynet* [11] network simulator to compare the performance of our system over a 100 Mbps LAN to that over a 1.5 Mbps downlink / 384 Kbps uplink DSL connection with 40 msec round-trip delay.

4.1.1 Effects of Caching

To evaluate caching, we use three repeatable workloads: bootup and shutdown of a Linux VM, bootup and shutdown of a Windows XP VM, and building the Linux 2.4.23 kernel in a VM. The runtime of each workload is measured in different network and cache configurations to illustrate how caching and network connectivity affect performance. All workloads are run with both an empty and a fully prefetched initial cache. We also repeat the workloads with a fully prefetched cache but without the bootup block optimization, to show the optimization's effect on startup performance.

By running the same virtual machine workloads on an unmodified version of VMware's GSX server, we quantify the benefits and overheads imposed by the Collective caching system. In particular, we run two sets of experiments using unmodified VMware without the Collective cache:

- *Local*, where the entire VM is copied to local disk and executes without demand-fetching. The COW disks of each VM disk are collapsed into a flat disk for this experiment. We expect this to provide a bound on VMware performance.

- *NFS*, where the VM is stored on an NFS file server and is demand-fetched by VMware without additional caching. This is expected to be slow in the DSL case and shows the need for caching.

Figure 2 summarizes the performance of these benchmarks. Workloads running with a fully prefetched cache are slower than the *local* workload, due to additional seek overhead imposed by the layout of blocks in our cache. The bootup block prefetching optimization, described in

Section 3.5, largely compensates for the suboptimal block layout.

As expected, the performance of our workloads is bad in both the NFS and the empty cache scenario, especially in the case of a DSL network, thus underscoring the need for caching.

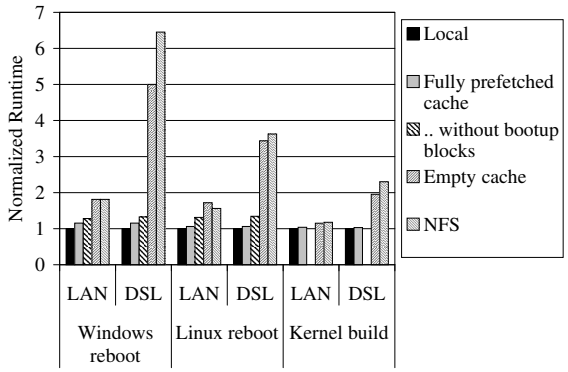

Figure 2: Runtime of workload experiments on different cache configurations when run over a 100 Mbps LAN, and a simulated DSL link with 1.5 Mbps downlink / 384 Kbps uplink and 40 msec RTT latency. The runtimes are normalized to the runtime in the local experiment. The local runtimes are 64 sec, 32 sec, and 438 sec, respectively for the Windows reboot, Linux reboot, and Linux kernel build experiments.

4.1.2 Effects of disk performance

As a first test to evaluate the performance of different disks, we measured the time taken to boot the VAT software on an IBM Thinkpad T42p laptop, since our standard experimental desktop machine did not have USB 2.0 support in its BIOS. The results, shown in the first column of Figure 3 indicate that the VAT boot process is reasonably fast across different types of drives we tried.

For our second test, we run the same micro-benchmark workloads as above; to emphasize disk performance rather than network performance, the VMs are fully prefetched into the cache, and the machines are connected over a 100Mbps LAN. The results are shown in Figure 3. The flash drive performs well on this workload, because of its good read performance with zero seek time, but has limited capacity, which would prevent it from running larger applications well. The microdrive is relatively slow, largely due to its high seek time and rotational latency. In our opinion, the 1.8" hard drive offers the best price / performance / form factor combination.

4.2 Maintaining Appliances

We have created and maintained three virtual machine appliances over a period of time:

- a Windows XP environment. Over the course of half a year, the Windows appliance has gone through two service packs and many security updates. The appliance initially contained Office 2000 and was upgraded to Office 2003. The appliance includes a large number of applications such as Adobe PhotoShop, FrameMaker, and Macromedia DreamWeaver.

- a Linux environment, based on Red Hat's Fedora Core, that uses NFS to access our home directories on our group file server. Over a period of eight months, the NFS Linux appliance required many security updates, which replaced major subsystems like the kernel and X server. Software was added to the NFS Linux appliance as it was found to be needed.

- a Linux environment also based on Fedora, that stores the user's home directory in a user disk. This Linux appliance included all the programs that came with the distribution and was therefore much larger. We used this appliance for two months.

	VAT startup	Windows reboot	Linux reboot	Kernel build
Lexar 1GB Flash Drive	53	129	42	455
IBM 4GB Microdrive	65	158	53	523
Hitachi 40GB 1.8" Drive	61	84	43	457
Fujitsu 60GB 2.5" Drive	52	65	40	446

Figure 3: Performance characteristics of four different VAT disks. All the numbers are in seconds. The first column shows the VAT boot times on an IBM Thinkpad T42p, from the time BIOS transfers control to the VAT's boot block to the VAT being fully up and running. In all cases the BIOS takes an additional 8 seconds initializing the system before transferring control to the VAT. The rest of the table shows results for the micro-benchmarks run with fully-primed caches when run over a 100 Mbps network.

Some vital statistics of these appliances are shown in Figure 4. We show the number of versions created, either due to software installations or security patches. Changes to the system happen frequently; we saved a lot of time by having to just update one instance of each appliance.

Appliance	Number of versions	Total size	Active size	Cache size
Windows XP	31	16.5	4.5	3.1
NFS Linux	20	5.7	2.8	1.4
User-disk Linux	8	7.0	4.9	3.7

Figure 4: Statistics of three appliances. Sizes are in GB.

We also measure the size of all the COW disks for each appliance ("Total size") and the size of the latest version ("Active size"). The last column of the table, "Cache size", shows an example of the cache size of an active user of each appliance. We observe from our usage that the cache size grows quickly and stabilizes within a short amount of time. It grows whenever major system updates are performed and when new applications are used for the first time. The sizes shown here represent all the blocks ever cached and may include disk blocks that may have

since been made obsolete. We have not needed to evict any blocks from our 40GB disks.

4.3 Effectiveness of Prefetching

In the following, we first measure the access profile to establish that prefetching a small amount of data is useful. Second, we measure the effect of prefetching on the performance of an interactive application.

4.3.1 Access Profile

In this experiment, we measure the access profile of appliance blocks, to understand the effectiveness of prefetching based on the popularity of blocks. We took 15 days of usage traces from 9 users using the three appliances described above in their daily work. Note that during this period some of the appliances were updated, so the total size of data accessed was greater than the size of a single active version. For example, the Windows XP appliance had an active size of 4.5 GB and seven updates of 4.4 GB combined, for a total of 8.9 GB of accessible appliance data.

Figure 5 shows each appliance's effective size, the size of all the accesses to the appliance in the trace, and the size of unique accesses. The results suggest that only a fraction of the appliance data is ever accessed by any user. In this trace, users access only 10 to 30% of the accessible data in the appliances.

Appliance	Accessible Size	Accesses in Traces	Unique data Accessed
Windows XP	8.9 GB	31.1 GB	2.4 GB
NFS Linux	3.4 GB	6.8 GB	1.0 GB
User-disk Linux	6 GB	5.9 GB	0.5 GB

Figure 5: Statistics of appliances in the trace.

Figure 6 shows the percentage of accesses that are satisfied by the cache (Y-axis) if a given percentage of the most popular blocks are cached (X-axis). The results show that a large fraction of data accesses are to a small fraction of the data. For example, more than 75% of data accesses in the Windows XP appliance are to less than 20% of the accessed data, which is about 5% of the total appliance size. These preliminary results suggest that popularity of accessed appliance data is a good heuristic for prefetching, and that prefetching a small fraction of the appliance's data can significantly reduce the chances of a cache miss.

4.3.2 Interactive Performance

The responsiveness of an interactive application can be severely affected by cache miss delays. Our next experiment attempts to measure the effects of prefetching on an application's response time.

To simulate interactive workloads, we created a VNC [10] *recorder* to record user mouse and keyboard input events, and a VNC *player* to play them back to reproduce user's actions [25]. Using VNC provides us with

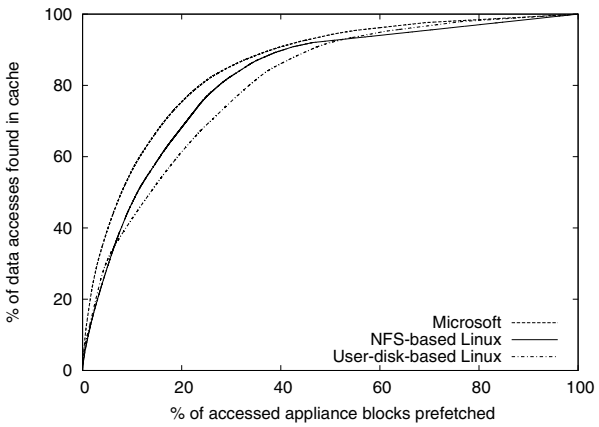

Figure 6: Block access profile: cache hit rate as a function of prefetched appliance data. Most frequently used appliance data is prefetched first.

a platform-independent mechanism for interacting with the desktop environment. Furthermore, it allows us to use VMware's built-in VNC interface to the virtual machine console.

Other tools [19, 23] try to do this, but play back is not always correct when the system is running significantly slower (or faster) than during recording. This is especially true for mouse click events. To reliably replay user actions, our VNC recorder takes screen snapshots along with mouse click events. When replaying input events, the VNC player waits for the screen snapshot taken during recording to match the screen contents during replay before sending the mouse click.

Our replay works only on systems with little or no nondeterministic behavior. Since we use virtual machines, we can easily ensure that the initial state is the same for each experiment.

We use the Windows XP appliance to record a VNC session of a user creating a PowerPoint presentation for approximately 8 minutes in a LAN environment. This session is then replayed in the following experimental configurations:

- Local: the entire appliance VM is copied to the VAT disk and executed with unmodified VMware, without demand-fetching or caching.
- Prefetched: some of the virtual machine's blocks are prefetched into the VAT's cache, and the VM is then executed on top of that cache. The VAT is placed behind a simulated 1.5 Mbps / 384 Kbps DSL connection.

For the prefetched experiments, we asked four users to use various programs in our appliance, to model other people's use of the same appliance; their block access patterns are used for prefetching blocks in the experiment.

Prefetching measures the amount of data transferred over the network; due to compression, the amount of raw disk data transferred is approximately 1.6 times more. The amount of prefetching goes up to a maximum of 420 MB, which includes all of the blocks accessed in the appliance by our users.

The total runtimes for the replayed sessions are within approximately 5% of each other – the additional latency imposed by demand-fetching disk blocks over DSL is absorbed by long periods of user think time when the system is otherwise idle. To make a meaningful comparison of the results, we measure the response time latency for each mouse click event, and plot the distribution of response times over the entire workload in Figure 7. For low response times, the curves are virtually indistinguishable. This region of the graph corresponds to events that do not result in any disk access, and hence are quick in all the scenarios. As response time increases, the curves diverge; this corresponds to events which involve accessing disk – the system takes noticeably longer to respond in this case, when disk blocks need to be demand-fetched over the network. The figure shows that PowerPoint running in the Collective is as responsive as running in a local VM, except for times when new features have to be loaded from disk – similar to Windows taking a while to start any given application for the first time.

The most commonly accessed blocks are those used in the bootup process. This experiment only measures the time taken to complete the PowerPoint workload after the system has been booted up, and therefore the benefit of prefetching the startup blocks is not apparent in the results shown in the figure. However, prefetching the startup blocks (approximately 100 MB) improves startup time from 391 seconds in the no prefetching case to 127 seconds when 200 MB of data is prefetched.

The results show that prefetching improves interactive performance. In the case of full prefetching, the performance matches that of a local VM. Partial prefetching is also beneficial – we can see that prefetching 200 MB significantly improves the interactive performance of PowerPoint.

4.4 Feasibility of Online Backup

Ideally, in our system, user data should always be backed up onto network storage. To determine whether online backup works for real workloads, we collected usage traces for three weeks on personal computers of ten users running Windows XP. These users included office workers, home users, and graduate students. The traces contain information on disk block reads and writes, file opens and start and end of processes. We also monitored idle times of keyboard and mouse; we assume the user to be idle if the idle time exceeds five minutes.

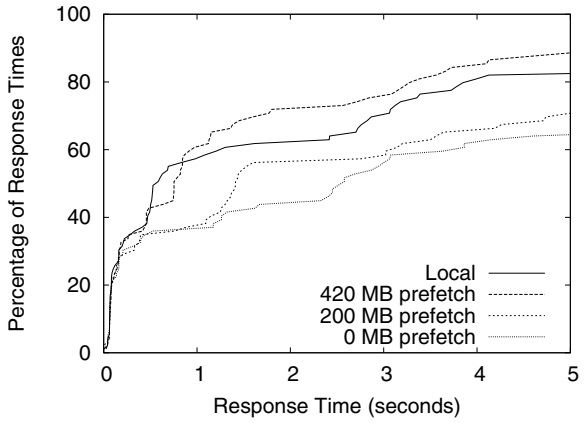

Figure 7: CDF plot of response times observed by the user during a PowerPoint session, for different levels of prefetching.

We expect that in our system the user would log out and possibly shut down his VAT soon after he completes his work. So, the measure we are interested in is whether there is any data that is not backed up when he becomes idle. If all the data is backed up, then the user can log in from any other VAT and get his most recent user data; if the user uses a portable VAT, he could lose it with no adverse effects.

To quantify this measure we simulated the usage traces on our cache running over a 384 Kbps DSL uplink. To perform the simulation we divided the disk writes from the usage data into writes to system data, user data, and ephemeral data. These correspond to the system disk, user disk, and ephemeral disk that were discussed earlier. System data consists of the writes that are done in the normal course by an operating system that need not be backed up. Examples of this include paging, defragmentation, NTFS metadata updates to system disk, and virus scans. User data consists of the data that the user would want to be backed up. This includes email documents, office documents, etc., We categorize internet browser cache, and media objects such as mp3 files, that are downloaded from the web as ephemeral data and do not consider them for backup. In our traces there were a total of about 300GB worth of writes of which about 3.3% were to user data, 3.4% were to ephemeral data and the rest to program data. Users were idle 1278 times in the trace, and in our simulation, backup stops during idle periods. We estimate the size of dirty data in the cache when users become idle.

The results are presented in Figure 8. The x-axis shows the size of data that is not backed up, and the y-axis shows the percentage of idle periods. From the figure we see that most of the time there is very little data to be backed up by the time the user becomes idle. This suggests that interactive users have large amounts of think time and generate little backup traffic. This also shows that online backup,

as implemented in the Collective, works well even on a DSL link. Even in the worst case, the size of dirty data is only about 35 MB, which takes less than 15 minutes to backup on DSL.

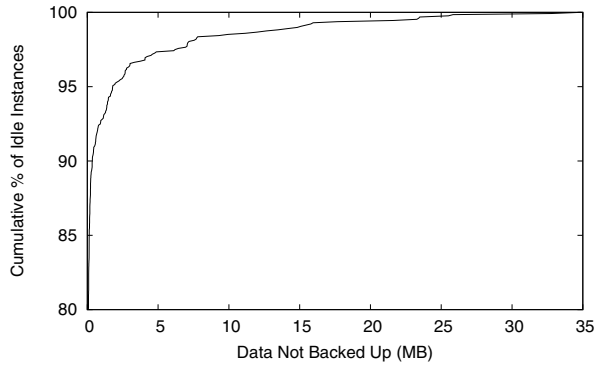

Figure 8: Size of data that is not backed up when a user becomes idle. The graph shows the fraction of times the user would have less than a certain amount of dirty data in his cache at the end of his session.

The results presented in this section illustrate that the system performs well over different network connections, and that it provides a good interactive user experience. Further, the results support our use of prefetching for reducing cache misses, and show that continuous backup is feasible for most users.

5 Experiences

We have been using the Collective for our daily work since June 2004. Based on this and other experiences, we describe how the Collective helped reduce the burden of administering software and computers.

5.1 Uses of the System

At first, members of our research group were using the prototype for the sake of understanding how our system behaves. As the system stabilized, more people started using the Collective because it worked better than their current setup. The following real-life scenarios we encountered illustrate some of the uses of our system:

Deploying new equipment. Before the Collective, when we needed to set up a new desktop or laptop, it would take a couple of hours to install the operating system, applications, and configure the computer. By plugging in and booting from USB disk containing the VAT, we were able to start using the computer immediately, starting up appliances we had previously used on other computers. We also used the VAT to configure the new computer's internal hard drive to be a VAT; all it takes is one user command and, in less than 5 minutes, the computer is assimilated into the Collective.

Fixing broken software setups. In one case, a student adopted the Collective after he botched the upgrade of the Linux kernel on his laptop. As a result of the failed upgrade, the laptop did not even boot. In other cases, we had lent machines to other groups and received them back with less than useful software setups. The Collective allowed us to resume work quickly by placing a VAT on the computer.

Distributing a complex computing environment. Over the summer, two undergraduates participated in a compiler research project that required many tools including Java, Eclipse, the JoeQ Java compiler, BDD libraries, etc. Since the students were not familiar with those tools, it would have taken each of the students a couple of days to create a working environment. Instead, an experienced graduate student created an appliance that he shared with both students, enabling both of them to start working on the research problems.

Using multiple environments. Our Linux appliance users concurrently start up a Windows appliance for the occasional tasks, like visiting certain web pages and running Powerpoint, that work better or require using Windows applications.

Distributing a centrally maintained infrastructure. Our university maintains a pool of computers that host the software for course assignments. Towards the end of the term, these computers become over-subscribed and slow. While the course software and the students' home directories are all available over a distributed file system (AFS), most students do not want to risk installing Linux and configuring AFS on their laptops. We gave students external USB drives with a VAT and created a Linux appliance that uses AFS to access course software and their home directories. The students used the VAT and the appliance to take advantage of the ample cycles on their laptops, while leaving the Windows setup on their internal drive untouched.

5.2 Lessons from our Experience

We appreciate that we only need to update an appliance once and all of the users can benefit from it. The authors would not be able to support all the users of the system otherwise.

The design of the VAT as a portable, self-contained, fixed-function device contributes greatly to our ability to carry out our experiments.

1. Auto-update. It is generally hard to conduct experiments involving distributed users because the software being tested needs to be fixed and improved frequently, especially at the beginning. Our system automatically updates itself allowing us to make quick iterations in the experiment without having to recall the experiment. The user needs to take no action, and the system has the appearance of healing itself upon a reboot.

2. Self-containment. It is easy to get users to try out the system because we give them an external USB drive from which to boot their computer. The VAT does not disturb the computing environment stored on their internal hard drive.

The system also makes us less wary of taking actions that may compromise an appliance. For example, we can now open email attachments more willingly because our system is up to date with security patches, and we can roll back the system should the email contain a new virus. As a trial, we opened up a message containing the BagleJ email virus in a system that had not yet been patched. Because BagleJ installed itself onto the system disk, it was removed when we rebooted. We have had similar experiences with spyware; a reboot removes the spyware executables, leaving only some icons on the user's desktop to clean up.

We observed that the system can be slow when it is used to access appliance versions that have not yet been cached. This is especially true over a DSL network. Prefetching can be useful in these cases. Prefetching on a LAN is fast; on a DSL network, it is useful to leave the computer connected to the network even when it is not in use, to allow prefetching to complete. The important point to note here is that this is fully automatic and hands-free, and it is much better than having to baby-sit the software installation process. Our experience suggests that it is important to prioritize between the different kinds of network traffic performed on behalf of the users; background activities like prefetching new appliance versions or backing up user data snapshots should not interfere with normal user activity.

We found that the performance of the Collective is not satisfactory for I/O intensive applications such as software builds, and graphics intensive applications such as video games. The virtualization overhead, along with the I/O overhead of our cache makes the Collective not suitable for these applications.

Finally, many software licenses restrict the installation of software to a single computer. Software increasingly comes with activation and other copy protection measures. Being part of a large organization that negotiates volume licenses, we avoided these licensing issues. However, the current software licensing model will have to change for the Collective model to be widely adopted.

6 Related Work

To help manage software across wide-area grids, GVFS [26] transfers hardware-level virtual machines. Their independent design shares many similarities to our design, including on-disk caches, NFS over SSH, and VMM-specific cache coherence. The Collective evaluates a broader system, encompassing portable storage, user data, virtual appliance transceiver, and initial user experiences.

Internet Suspend/Resume (ISR) [7] uses virtual machines and a portable cache to provide mobility; the Collective architecture also provides management features like rollback and automatic update, in addition to mobility. Similar to our previous work [13], ISR uses a cache indexed by content hash. In contrast, the current Collective prototype uses COW disks and a cache indexed by location. We feel that any system like the Collective needs COW disks to succinctly express versions; also, indexing the cache by location was straightforward to implement. Index by hash does have the advantage of being able to use a cached block from an unrelated disk image. Our previous work [13] suggests that there is promise in combining COW disks and index by hash. In the case a user does not wish to carry portable storage, ISR also implements *proactive* prefetching, which sends updated blocks to the computers the user uses commonly in anticipation of the user arriving there. The Collective uses prefetching of data from repositories to improve the performance at VATs where the user is already logged in. The two approaches are complementary.

Managing software using disk images is common; a popular tool is Symantec Ghost [17]. Unlike our system, a compromised operating system can disable Ghost since the operating system has full access to the raw hardware. In addition, since Ghost does not play copy-on-write tricks, roll back involves rewriting the whole partition. This potentially lengthy process limits the frequency of ghosting. Finally, Ghost leaves it to the administrator and other tools to address how to manage user data.

Using network repositories for disk images and expressing updates compactly using differences are explored by Rauch et al [9]. A different way of distributing disk images is Live CDs, bootable CDs with a complete software environment. Live CDs provide lock down and can easily roll back changes to operating systems. However, they do not provide automatic updates and management of user data.

Various solutions for transparent install and update exist for platforms other than x86 hardware. Java has Java Web Start [21]; some Windows games use Valve Steam; Konvalo and Zero Install manage Linux applications. The Collective uses virtual machine technology and an automatically updating virtual appliance transceiver to manage the entire software stack.

Like the Collective, MIT's Project Athena [4] provides the management benefits of centralized computing while using the power of distributed desktop computers. In Athena, management is a service that runs alongside applications; in contrast, the Collective's management software are protected from the applications by a virtual machine monitor. The Collective uses a disk-based ab-

straction to distribute software and user data; in contrast, Athena uses a distributed file system. By explicitly exposing multiple versions of disk images through repositories, the Collective can provide consistent snapshots of software and does not force users to start using the new version immediately. In contrast, software run from a network file system must be carefully laid out and managed to provide similar semantics. In Athena, users can mix and match software from many providers; in our model, an appliance is a monolithic unit created and tested by an administrator.

Candea et al [3] have explored rebooting components of a running system as a simple, consistent, and fast method of recovery. The Collective uses reboots to rollback changes and provide upgrades, providing similar advantages.

7 Conclusions

This paper presents the Collective, a prototype of a system management architecture for managing desktop computers. This paper concentrates on the design issues of a complete system. By combining simple concepts of caching, separation of system and user state, network storage, and versioning, the Collective provides several management benefits, including centralized management, atomic updates, and recovery via rollback.

Our design of a portable, self-managing virtual appliance transceiver makes the Collective infrastructure itself easy to deploy and maintain. Caching in the Collective helps provide good interactive performance even over wide-area networks. Our experience and the experimental data gathered on the system suggest that the Collective system management architecture can provide a practical solution to the complex problem of system management.

8 Acknowledgments

The work for this paper was funded in part by the National Science Foundation under Grant No. 0121481. We would like to thank Ben Pfaff, Chris Unkel, Emre Kıcıman, George Candea, Arvind Arasu, Mendel Rosenblum, Eu-jin Goh, our shepherd Ed Lazowska, and the anonymous reviewers for their comments.

References

[1] E. Bailey. *Maximum RPM*. Sams, 1st edition, 1997.

[2] W. M. Bulkeley. The office PC slims down. *The Wall Street Journal*, January 2005.

[3] G. Candea, S. Kawamoto, Y. Fujiki, G. Friedman, and A. Fox. Microreboot – a technique for cheap recovery. In *Proceedings of the 6th Symposium on Operating Systems Design and Implementation*, pages 31–44, December 2004.

[4] G. Champine, J. Daniel Geer, and W. Ruh. Project Athena as a distributed computer system. *IEEE Computer Magazine*, 23(9):40–51, September 1990.

[5] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh. Terra: A virtual machine-based platform for trusted computing. In *Proceedings of the 19th Symposium on Operating System Principles(SOSP 2003)*, pages 193–206, October 2003.

[6] KNOPPIX Live CD Linux distribution. `http://www.knoppix.org/`.

[7] M. Kozuch, M. Satyanarayanan, T. Bressoud, C. Helfrich, and S. Sinnamohideen. Seamless mobile computing on fixed infrastructure. Technical Report 28, Intel Research Pittsburgh, 2004.

[8] J. Nieh, S. J. Yang, and N. Novik. Measuring thin-client performance using slow-motion benchmarking. *ACM Transactions on Computer Systems*, 21(1), February 2003.

[9] F. Rauch, C. Kurmann, and T. Stricker. Partition repositories for partition cloning—OS independent software maintenance in large clusters of PCs. In *Proceedings of the IEEE International Conference on Cluster Computing 2000*, pages 233–242, November/December 2000.

[10] T. Richardson, Q. Stafford-Fraser, K. R. Wood, and A. Hopper. Virtual network computing. *IEEE Internet Computing*, 2(1):33–38, January/February 1998.

[11] L. Rizzo. Dummynet: a simple approach to the evaluation of network protocols. *ACM Computer Communication Review*, 27(1):31–41, January 1997.

[12] C. Sapuntzakis, D. Brumley, R. Chandra, N. Zeldovich, J. Chow, J. Norris, M. S. Lam, and M. Rosenblum. Virtual appliances for deploying and maintaining software. In *Proceedings of Seventeenth USENIX Large Installation System Administration Conference*, pages 181–194, October 2003.

[13] C. Sapuntzakis, R. Chandra, B. Pfaff, J. Chow, M. Lam, and M. Rosenblum. Optimizing the migration of virtual computers. In *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation*, pages 377–390, December 2002.

[14] C. Sapuntzakis and M. Lam. Virtual appliances in the collective: A road to hassle-free computing. In *Workshop on Hot Topics in Operating Systems*, pages 55–60, May 2003.

[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 System Principles*, pages 32–47, December 1999.

[16] S. Staniford, V. Paxson, and N. Weaver. How to 0wn the Internet in your spare time. In *Proceedings of the 11th USENIX Security Symposium*, pages 149–167, August 2002.

[17] Symantec Ghost. `http://www.ghost.com/`.

[18] TCPA. `http://www.trustedcomputing.org/`.

[19] Rational VisualTest. `http://www.ibm.com/software/awdtools/tester/robot/`.

[20] VMware GSX server. `http://www.vmware.com/products/server/gsx_features.html`.

[21] Java web start. `http://java.sun.com/j2se/1.5.0/docs/guide/javaws/`.

[22] P. Wilson. *Definitive Guide to Windows Installer*. Apress, 1st edition, 2004.

[23] Xnee home page. `http://www.gnu.org/software/xnee/www`.

[24] Yellowdog updater modified (yum). `http://linux.duke.edu/projects/yum/`.

[25] N. Zeldovich and R. Chandra. Interactive performance measurement with VNCplay. In *Proceedings of the FREENIX Track: 2005 USENIX Annual Technical Conference*, pages 153–162, April 2005.

[26] M. Zhao, J. Zhang, and R. Figueiredo. Distributed file system support for virtual machines in grid computing. In *Proceedings of the Thirteenth IEEE Symposium on High-Performance Distributed Computing*, June 2004.