Internet Suspend/Resume

Michael Kozuch, Intel Research Pittsburgh
M. Satyanarayanan, Carnegie Mellon University and Intel Research Pittsburgh

Abstract

We identify a new capability for mobile computing that mimics the opening and closing of a laptop, but avoids physical transport of hardware. Through rapid and easy personalization and depersonalization of anonymous hardware, a user is able to suspend work at one machine and to resume it at another. Our key insight is that this capability can be achieved by layering virtual machine technology on a distributed file system. We report on an initial implementation and describe our plans for improving efficiency, portability, and security.

1 Introduction

When you close a laptop, its state is suspended. You can travel a great distance and open the cover many hours or days later; the laptop's execution state is restored precisely as it was at suspension. Can we achieve this capability without physically carrying the laptop? In other words, can I logically suspend a machine at one Internet site, then travel to some other site and resume my work there on another machine? Can state restoration be as swift as it is when you open a laptop? We call this hypothetical capability *Internet Suspend/Resume (ISR).*

If feasible, ISR would support mobile computing without hardware being physically carried. For example, one could pick up a laptop at a car rental agency much as one rents a cell phone today. Or, one could imagine a laptop being built into every seat in an aircraft for use during flights, much as video players are today. The essential capability needed for these scenarios is the ability to effortlessly save and restore computing state — in other words, to rapidly and easily personalize and depersonalize anonymous hardware. Of course, the term "laptop" is used here only for illustration. This work applies equally well to any other form of computing hardware, from desktops to handheld or wearable computers.

The key insight offered by this paper is that ISR can be realized by combining two off-the-shelf technologies: *virtual machine* (VM) technology and *distributed file systems.* We report on a simple proof-of-concept implementation in this paper. We also identify many important improvements, and describe our plans for building a more robust, efficient, and complete prototype.

2 Design Overview

ISR is foreshadowed by location transparent distributed file systems such as AFS [10] and Coda [16]. If a user restricts all his file accesses to such a file system (including placing his home directory in it), he will see identical file state at all clients. He can log in to any client, work for a while, log out, move to any other client, log in, and continue his work. There are at least two ways in which this capability falls short of ISR. First, only persistent state is saved and restored. Volatile state, such as the the execution states of interactive applications, is not preserved. The second shortcoming follows from the first. From a user's viewpoint, the "suspend" and "resume" steps are considerably more complex, heavyweight and slow than closing and opening a laptop.

A *virtual machine monitor (VMM)* [6] cleanly encapsulates all volatile execution state of a VM. The operating system that executes within a VM is referred to as a *guest* operating system. A VMM typically maps the volatile state of its VMs to files in the local file system of its host. When a VM is suspended, the corresponding files are updated to reflect volatile state at the point of suspension. If these files are copied to a remote host with similar hardware architecture, a VMM on that host can resume the VM. In other words, the VM, including the volatile state of the guest OS, has been migrated.

In contrast to the well-known difficulties of process migration [4], VM migration is simpler because volatile execution state is better encapsulated. Because migration takes place at the machine level, some problems that have historically presented challenges for process migration are not present, such as managing open file descriptors cached by applications. Another historic challenge, migrating network connections, is manageable because modern desktop operating systems provide features such as ACPI [1] compliance to accomodate laptop mobility. To the guest OS, a VM mi-

Proceedings of the Fourth IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'02)
0-7695-1647-5/02 $17.00 © 2002 IEEE

gration is similar to closing the lid on a laptop, moving the laptop, and re-opening the lid. Without mobile IP [12] technology the change in network topology is not truly transparent, but the change can be handled smoothly by the guest operating system.

VM migration is also tolerant of greater disparity between the source and target systems across which migration occurs. Independent of our work, Chen and Noble have also made this observation recently [2]. For process migration to succeed, there has to be a very close match between host and target operating systems, language runtime systems, and so on. In contrast, VM migration only requires a compatible VMM and hardware architecture at the target.The price for this greater flexibility is that VM migration may involve the transfer of more state.

A distributed file system can serve as the transport mechanism for propagating suspended VM state across space and time. The obvious approach would be to place the relevant files in the shared name space of the distributed file system and configure the VMM to access that name space directly. However, we have identified a number of file system enhancements that may prove to be useful in an ISR environment (Section 4). Rather than attempt to implement these enhancements in a distributed file system directly, we have adopted a more flexible approach for early experimentation. In this approach, the VMM only accesses files in the local files system. VM files are explicitly copied into the local file system during resume operations and out of the local file system during suspend operations.

Another benefit of the explicit copyin/copyout approach is that we can experiment with several different distributed file systems and accommodate their various idiosyncrasies. As an example, many distributed file systems do not precisely emulate POSIX file system semantics for reasons of scalability and performance. Consequently, the implementations of many system calls may diverge in subtle ways from their POSIX specifications. Our explicit copyin/copyout approach decouples the implementation complexities of distributed file systems and VMMs.

3 Initial Implementation

We have built an initial proof-of-concept implementation using VMware Workstation 3.0 [7] as the VMM and NFS as the distributed file system. VMware Workstation is a modern, commercial VMM which executes on a PC platform and provides a VM abstraction identical to a PC. VMware Workstation runs within a *host* operating system and relies on it for common system services such as device management. Both Linux and Windows 2000/XP are supported as host operating systems.

VMware Workstation supports many operating systems as guests including Windows 95/98, Windows 2000/XP, and Linux. A user can configure many important parameters that define the VM including the amount of memory, size and arrangement of disk drives, and number of network adapters. The video output of the VM may either appear as a window on the desktop of the host or occupy the entire host screen.

Note that, in our work, the distributed file system is used to transport the VM state files; it is not necessarily visible to the guest operating system. Similarly, while the VMM stores the contents of a VM's virtual disks in local files, it is not capable of interpreting the data stored in those files. The contents of the virtual disks, and hence the guest file system, are opaque from the point-of-view of the VMM.

3.1 Test System

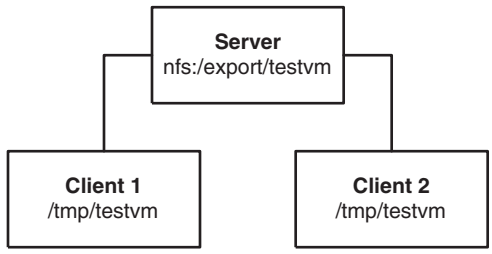
In our test environment we emulate a common VM usage model: the host operating system is Linux and the guest operating system is Windows XP. We have configured our guest with 128 MB of main memory, a single Ethernet card, and a 2 GB virtual disk. We consider this to be a modest configuration. The memory and disk sizes are sufficiently large that Windows XP runs out-of-the-box, but sufficiently small that the host operating system may easily manage the VM state files (Section 3.2).

The test system hardware and arrangement is depicted in Figure 1. Assuming that our experimental VM, *testvm*, is inactive (i.e. it has previously been suspended), the VM state associated with *testvm* is stored on the server in the NFS share, `/export/testvm`. If a user at Client 1 wishes to resume execution of the VM, she invokes a simple Linux script which implements the copyin operation described in Section 2. This script retrieves the VM state stored on the server and reconstructs that state in the local file system of Client 1 under `/tmp/testvm`. After the VM state has been reconstructed, the script launches VMware. We refer to the copyin operation and the application launch together as the "resume event."

When the user is finished with *testvm*, she suspends the VM. When the VMM exits, the copyout script copies the VM state to the server, possibly altering the format of the files in transit. The VM may now be migrated to Client 2, if desired. We use the term "suspend event" to refer to the combination of local VM state save by the VMM and copyout.

In our implementation, the copyout script divides the VM state files into 16 MB chunks and stores them in the NFS share. The copyin script reassembles file chunks into the local file system (`/tmp`) on the client machine. We implemented this chunking operation in anticipation of layering our work on other distributed file systems that do not handle multi-gigabyte files very well.

Proceedings of the Fourth IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'02)
0-7695-1647-5/02 $17.00 © 2002 IEEE



All machines are 1.7 GHz Pentium 4, single processor computers with 512 MB DRAM running Red Hat Linux 7.2 and NFS v3. The client machines are also running VMware Workstation 3.0, and the network connections are 100 Mb/s Ethernet.

Figure 1. ISR Test System

3.2 File System Mapping

Figure 2 describes the mapping between VM execution state and files for our VMware-based test environment. In this case, a single directory in the local file system contains all the relevant files. Fortunately, the designers of the VMware Workstation product use a straightforward representation of the VM state required for modern PC hardware.

A single file, `testvm.cfg`, represents the VM configuration. This file describes the types, numbers, and arrangement of the virtual hardware components. Another file, `testvm.log`, serves as a logging mechanism for the VMM.

The non-volatile state of the virtual system is maintained in a handful of files: one for the standard PC non-volatile memory state (`testvm.nvram`), and one for each of the virtual disk drives (`testvm.vmdk`). We configured our example VM with a single virtual disk, but if we had included additional drives the VM state directory would have included multiple virtual disk files. Although a VMM may configure a VM to directly access physical hard drives in the host platform, we only consider VMs with virtual disk drives.

The non-volatile state files provide enough information to restart the VM after it has been powered-off. To successfully suspend and resume a VM without powering it down, the VMM must also capture the volatile state of the VM at the point of suspension. In the example of Figure 2, this file is called `testvm.vmss`. This file is responsible for capturing the volatile state of VM at the time of VM suspension including the processor, devices, and main memory.

For our test VM, the `testvm.vmss` file is 134 MB. The virtual disk representation, `testvm.vmdk`, is only about 1GB even though *testvm* is configured with a 2GB disk. This is because VMware can compactly represent the state of a virtual disk that is not fully used.

Filename	Size	Function
`testvm.cfg`	1KB	Configuration information specifying number and type of virtual hardware components
`testvm.vmdk`	1GB	Data on virtual disk drive (Configured to be 2GB, but only 1GB is currently being used.)
`testvm.nvram`	8KB	State of non-volatile system memory (CMOS RAM)
`testvm.vmss`	134MB	*Only present if VM is suspended.* Stores the current system state of the VM (DRAM, processor, and devices).
`testvm.log`	30KB	Log file used for debugging.

These are the files used by VMware Workstation to represent the suspended state of *testvm,* the VM used in our test system. As mentioned in Section 3.1, *testvm* is configured with 128MB of main memory and a 2GB disk. The guest OS is Windows XP.

Figure 2. Suspended File State of *testvm*

3.3 Evaluation

Our prototype ISR implementation is able to suspend a VM on Client 1, resume it on Client 2, and vice versa. Although many performance optimizations are yet to be implemented, we were curious about the performance of the suspend and resume operations on our prototype. We therefore conducted a set of experiments to measure this performance using our sample VM configuration, *testvm*.

Before each resume operation on a client, we rebooted that client to ensure a cold NFS cache. This represents a worst case scenario. We considered two kinds of suspend events. The first kind, called *warm suspend,* occurs shortly after a resume event. It therefore benefits from a warm NFS cache at the client and a warm I/O buffer cache on the server. In real life, this corresponds to very brief use of machine by a mobile user. To reflect this scenario, we suspended the VM immediately after a resume event. The second kind of suspend event, *cold suspend,* occurs long after a resume event. In real life, this corresponds to extended use of a VM. To reflect this scenario, we ensured cold NFS and I/O buffer caches by rebooting both machines prior to copyout. We also ensured that the contents of the test files remained constant throughout all of our experiments.

The second column in Figure 3 reports the measured resume and suspend times in our experiments. A cold resume takes roughly two minutes, and the suspend times range from slightly under two minutes to well over two minutes. These times are probably tolerable for some users but are much longer than the typical suspend/resume times of mod-

Proceedings of the Fourth IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'02)
0-7695-1647-5/02 $17.00 © 2002 IEEE

Event	No Compression	With Compression
(Cold) Resume	125 (0.2)	73 (4.3)
Warm Suspend	114 (5.1)	158 (1.4)
Cold Suspend	146 (19.6)	158 (0.9)

This table shows the average time, in seconds, of suspend and resume operations under cold and warm file cache conditions. All experiments were repeated three times, and the observed standard deviations are shown in parentheses.

Figure 3. Suspend/Resume Times in *testvm*

ern laptops.

We hypothesize that users perceive resume latency more acutely than suspend latency because the suspend operation may overlap other user activity. For example, the user can depart the work site immediately after suspending a VM and allow the operation to complete while travelling to the airport. Of course, attempts to resume the machine from another site will block until the suspend operation completes.

By introducing simple file compression, we are able to take advantage of this asymmetry between user perception of suspend and resume times. We introduced additional steps in the suspend and resume scripts to compress file chunks during copyout and to decompress them during copyin. Compression was done using `gzip` with the `--fast` command-line option, achieving an average compression ratio of about 49%. The third column of Figure 3 reports the effect of compression on performance. Resume time is reduced by nearly 40% to about 73 seconds, while warm suspend time is increased by about 38% and cold suspend time by 8%. Although the resume time of slightly over a minute may be acceptable in many scenarios we believe we can further reduce it as discussed in Section 4.

3.4 Shortcomings of Test System

While constructing our prototype, we observed three usability shortcomings. All of these arise from minor limitations in the current version of VMware Workstation. They do not pose significant obstacles to ISR, nor are they difficult to eliminate.

The first shortcoming stems from an imperfect abstraction of the host platform. In our first set of trials, we had configured both Client 1 and Client 2 (running on identical hardware including the monitors) with the same X display settings. However, we noticed that when the *testvm* was suspended on Client 1, it would fail to resume on Client 2. The error message would indicate that the screen resolution on Client 2 did not match that of *testvm* at suspend. The only difference between the two machines was that Client 1 was connected to its monitor through a KVM (keyboard/video/mouse) switch, while Client 2 was connected directly to its monitor. Removing the KVM eliminated the problem. Since the problem arises only on suspend/resume and not on power-down/power-up, we infer that display information is saved in `testvm.vmss` but not in the other files.

The second shortcoming relates to file location. We noticed that a machine suspended on Client 1 could not be resumed in a different directory on Client 2. Again, we only encountered this limitation when *testvm* was suspended, not when it was powered-down. This leads us to believe that `testvm.vmss` embeds absolute pathnames of the other VM state files. Our workaround was to ensure that copyin and copyout use identical directory names on the resume and suspend machines.

The third shortcoming is an omission. To our knowledge, the only way to suspend a running VM is through a mouse click in its control window. A mechanism for remote triggering of suspend would be valuable. For example, suppose a user forgets to suspend his VM before leaving work. On reaching home, he would like to be able to migrate that VM. At present, the only option is to ask someone back at work to manually click on the suspend button. A means of accomplishing this remotely (for example, through a "suspend and exit" signal) would be helpful. Of course, there are important security issues that would have to be addressed in implementing such a mechanism.

4 Improvements and Future Work

4.1 Improved Security

By the very nature of ISR, the VM state will often be transferred over insecure channels. In such an environment, properly protecting the contents of the VM state files is an important consideration because volatile state may contain highly sensitive information such as passwords in the clear. To address this problem, we plan to encrypt data on copyout and decrypt it on copyin. This means that no VM state is ever stored in the clear in the distributed file system. Explicit encryption/decryption allows the security of ISR to be decoupled from that of the underlying distributed file system.

Another important dimension to security is the need for mutual validation of a user and the host system at resume time. This is clearly a problem that must be addressed before large-scale deployment of ISR using anonymous hardware is advisable. In this area, we hope to leverage the work of others, such as the TCPA [18].

Proceedings of the Fourth IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'02)
0-7695-1647-5/02 $17.00 © 2002 IEEE

4.2 Smart Copying

An obvious limitation of our current implementation is the large amount of data that has to be copied from the suspend site to the resume site. The main components of suspended VM state include memory and virtual disks. These can total hundreds of megabytes to tens of gigabytes in typical systems today. Depending on the VMM, this data may be mapped to a single file or multiple files. In either case, the files are very large — at least a hundred megabytes in typical cases. We are exploring a number of different techniques for reducing the amount of data that must be transmitted between suspend and resume.

One approach is to exploit locality. People often move between the same small number of locations, such as home and work. In such a scenario, the file cache state at one location is likely to have considerable overlap with the file cache states at the other locations. It will differ only by the amount of work done since the VM was last suspended at the first location. If one can restrict transmission to just this difference, considerable reduction in data volume may result. One way to achieve this is to use differential file transmission techniques such as those described by Tridgell [19] for `rsync`, or by Muthitacharoen et al [11] for their low-bandwidth file system. Another way is to map a large file in the local file system into a tree of small files in the distributed file system. At the resume site, the large local file is reassembled from the tree — caching will ensure that only those (small) files that have changed will actually have to be transmitted. This is especially useful with distributed file systems like AFS and Coda that propagate changes at whole-file granularity.

We may also exploit temporal locality in situations in which the user wishes to work with a VM on a "new" physical machine (a machine that has never hosted that VM previously). In a typical computing session, a user may not access most of the virtual disk image. If the disk data is arranged in a tree of small files as described above, only a small number of these files will need to be transmitted to the new machine. Further, because the number of reads typically exceeds the number of writes, an even smaller number of files will need to be transmitted from the new machine to the distributed file system. This optimization may apply primarily to the VM disk files (`testvm.vmdk` in our example). This optimization may not prove to be as beneficial when handling VM memory state (`testvm.vmss` in our example).

A different approach is to synthesize much of the necessary state at the resume site. This can be effective even when there is no locality to exploit; for example, in the car rental and air travel scenarios of Section 1. On many systems for personal use, a large fraction of disk content consists of standard operating system and application software (e.g., Windows 2000 and the Microsoft Office suite); user-specific files may only be a very small fraction of disk size. Note that user-specific files include not just files in home directories, but also customized files scattered throughout the file system. If disk images of the standard software are available at the resume site, suspended state can be reconstructed by first applying those disk images and then applying customizations transmitted from the suspend site. This technique can be extended to situations where network bandwidth is highly uneven across sites. The large standard disk images can be widely distributed using peer-to-peer techniques such as Gnutella [8] and accessed at high bandwidth from a nearby site. Only the much smaller user-specific state needs to be transmitted at low bandwidth from the suspend site.

Our ongoing research will explore the relative merits of these alternatives for ISR. As mentioned in Section 2, the use of explicit copyin/copyout allows us to do this exploration without making changes to the underlying distributed file system.

4.3 Proactive State Transfer

It may sometimes be possible to predict the resume site with reasonable confidence. In the car rental and air travel scenarios of Section 1, for example, there is likely to be reservation or seat assignment information to use in prediction. In work-to-home and home-to-work scenarios, there may be extensive history on which the system can base its prediction. Context-aware pervasive computing systems such as Aura [14] may exploit other sources of knowledge such as a user's planned schedule, current location, and time.

By acting on such a prediction, the system can ensure that suspended state is available at the resume site in advance of demand — that is, before the user "opens the laptop." This, in turn, will lower resume latency. The proactive actions required may include some combination of file cache warming and state synthesis, depending on which of the many possibilities discussed in Section 4.2 are used. Coda's support for user-directed cache management, known as *hoarding,* can be especially useful in this application.

The consequences of acting on an erroneous prediction are relatively mild. Typically, useful file cache state is flushed and there is some wastage of memory, network bandwidth and CPU cycles in useless state synthesis. Both of these may result in performance degradation but no loss of correctness. In particular, no critical cleanup actions are needed to ensure correctness. If the precision of prediction is only to a small set of sites rather than exactly one, state can be proactively transferred to all those sites.

Proceedings of the Fourth IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'02)
0-7695-1647-5/02 $17.00 © 2002 IEEE

4.4 Multiple Back Ends

As mentioned in Section 2, distributed file systems rarely offer precise POSIX semantics. Rather, they relax those semantics to improve scalability and performance in a large distributed system. The details of approximation vary across systems. Consistency is the most common area of approximation. Protection, locking, and recording of access times are other common areas.

To allow flexibility in the choice of distributed file system, we plan to restructure ISR software so that most of the code is written to an abstract file system interface. The binding of this abstract interface to a specific file system will be encapsulated in a small back-end code component. This code component will also determine how VMM files are mapped to files in the distributed file system.

4.5 Incremental Reconstruction

Another way to reduce resume latency is to allow the VMM to begin execution before the reconstruction of its input files is complete. Reconstruction can then be overlapped with the resumed execution of the VM. If the VMM accesses a region of a file that is yet to be reconstructed, the access has to block until reconstruction of that region is complete. This effectively combines demand reconstruction with proactivity. Our implementation will use a stackable file system approach [9] with the reconstruction manager residing in user space to simplify experimentation. A modified Coda kernel module [15] with appropriate extensions will transparently redirect the VMM's file accesses to the reconstruction manager.

5 Alternatives to ISR

This technology addresses the issue of user mobility. During the 1980's and 1990's typical computer usage evolved from a model in which most users interacted with managed computer systems to a model in which many users interact with systems that are either (a) administrated by that user or (b) customized for that user. Many computing environments have become funadamentally *personal*. ISR attempts to provide a mechanism through which a user may interact with their personalized computing environment from different physical locations by leveraging VM technology.

However, at least two alternative technologies also address the issue of user mobility. The first alternative is remote user-interface (UI) technology as employed by VNC [13] and X-Move [17]. The basic tradeoff between remote UI technology and VM migration is that remote UI technology requires that a fairly steady stream of UI-update information be transmitted over the network, while VM migration requires a fairly large state transfer prior to resuming the VM but very little subsequent state transfer until the time of suspend. VM migration provides a clear advantage in scenarios in which the network may become disconnected (such as while travelling by air) or in which network latency may be high (otherwise, the user experience may be very poor). Remote UI technology provides a clear advantage when the user interacts with the system for a very short period of time.

The second alternative is distributed file system (DFS) technology as mentioned in Section 2. Traditionally, users have approximated ISR by storing session resume information in their home directories on a DFS. Perhaps, many users will find the ideal mobility solution a combination of ISR and DFS. In this solution, the OS and related files are stored in a disk image and the user's home directory is stored in the DFS.

6 Conclusion

The concept of ISR is obvious once it is described. Yet, to the best of our knowledge, we are the first to identify its significance and to demonstrate a working implementation. It is instructive to ask why this is the case. After all, IBM had a VM product for its mainframes by the early 1970s. Early distributed file systems, such as AFS and NFS, emerged by the mid-1980s. Why has it taken so long to compose the two technologies?

We conjecture that three factors are responsible for the late emergence of ISR. First, VMs were not available on hardware typically used by mobile computing researchers until recently. Only in the late 1990's, with the founding of VMware, has VM technology become available on this class of hardware. Second, most academic research has tended to focus on process migration as the mechanism for transferring live execution state. Experimental systems such as Demos, V, Mach and Sprite have successfully demonstrated process migration. But there has been no significant growth of a user community dependent on process migration. Indeed, there has been a persistent suspicion in the research community that process migration may be a solution in search of a problem! Third, the growing dominance of the Microsoft Office application suite has forced researchers to ask how those applications can be supported in a mobile computing environment. Since they run only on closed-source Windows operating systems, solutions that require operating system modifications are not feasible. A middleware approach such as Puppeteer [3, 5] is one solution. Using VMs, as described in this paper, is another solution.

Our initial prototype shows that ISR can be successfully implemented on today's hardware. We plan to improve the security, efficiency, and portability of this prototype using

Proceedings of the Fourth IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'02)
0-7695-1647-5/02 $17.00 © 2002 IEEE

the techniques discussed in Section 4. With these refinements, we are confident that ISR will play an important role in mobile computing.

7 Acknowledgments

AFS is the trademark of IBM Corporation. Linux is the trademark of Linus Torvalds. Microsoft Office and Windows are trademarks of Microsoft Corporation. Pentium is the trademark of Intel Corporation. Any other unidentified trademarks used in this paper are properties of their respective owners.

References

[1] ACPI. *Advanced Configuration and Power Interface Specification: Revision 2.0a*, March 2002. http://www.acpi.info/.

[2] Chen, P.M. and Noble, B.D. When Virtual is Better Than Real. In *Proceedings of the 8th IEEE Workshop on Hot Topics in Operating Systems*, Schloss Elmau, Germany, May 2001.

[3] de Lara, E., Wallach, D.S., and Zwaenepoel, W. Puppeteer: Component-Based Adaptation for Mobile Computing. In *Proceedings of the 3rd USENIX Symposium on Internet Technologies and Systems*, San Francisco, CA, March 2001.

[4] Douglis, F. and Ousterhout, J.K. Transparent Process Migration: Design Alternatives and the Sprite Implementation. *Software Practice and Experience*, 21(8), 1991.

[5] Flinn, J., de Lara, E., Satyanarayanan, M., Wallach, D.S., and Zwaenepoel, W. Reducing the Energy Usage of Office Applications. In *Proceedings of Middleware 2001: IFIP/ACM International Conference on Distributed Systems Platforms*, Heidelberg, Germany, November 2001.

[6] Goldberg, R.P. Survey of Virtual Machine Research. *IEEE Computer*, 7(6), June 1974.

[7] Grinzo, L. Getting Virtual with VMware 2.0. *Linux Magazine*, June 2000.

[8] Harris, R. Gnutella: a Net autopirate. *Seattle Post-Intelligencer*, April 13, 2000.

[9] Heidemann, J.S. and Popek, G.J. File System Development with Stackable Layers. *ACM Transactions on Computer Systems*, 12(1), February 1994.

[10] Howard, J.H., Kazar, M.L., Menees, S.G., Nichols, D.A., Satyanarayanan, M., Sidebotham, R.N., and West, M.J. Scale and Performance in a Distributed File System. *ACM Transactions on Computer Systems*, 6(1), February 1988.

[11] Muthitacharoen, A., Chen, B., and Mazieres, D. A Low-Bandwidth Network File System. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles*, Chateau Lake Louise, Banff, Canada, Oct. 2001.

[12] Perkins, C.E. Mobile Networking Through Mobile IP. *IEEE Internet Computing*, January 1998.

[13] Richardson, T., Stafford-Fraser, Q., Wood, K. R., and Hopper, A. Virtual Network Computing. *IEEE Internet Computing*, 2(1), Jan/Feb 1998.

[14] Satyanarayanan, M. Pervasive Computing: Vision and Challenges. *IEEE Personal Communications*, 8(4), August 2001.

[15] Satyanarayanan, M. The Evolution of Coda. *ACM Transactions on Computer Systems*, 20(2), May 2002.

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

[17] Solomita, E., Kempf, J., and Duchamp, D. XMOVE: A pseudoserver for X window movement. *The X Resource*, 11(1), 1994.

[18] TCPA. *Trusted Computing Platform Alliance: Main Specification Version 1.1a*, November 2001. http://www.trustedpc.org.

[19] Tridgell, A. and Mackerras, P. The rsync algorithm. Technical Report TR-CS-96-05, Department of Computer Science, The Australian National University, Canberra, Australia, 1996.

Proceedings of the Fourth IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'02)
0-7695-1647-5/02 $17.00 © 2002 IEEE