Seamless Mobile Computing on Fixed Infrastructure

Internet Suspend/Resume is a thick-client approach to mobility in which hardware virtualization and file caching are the keys to rapid personalization of anonymous hardware for transient use.

Michael Kozuch
Intel Research Pittsburgh

Mahadev Satyanarayanan
Carnegie Mellon University and Intel Research Pittsburgh

Thomas Bressoud
Denison University

Casey Helfrich
Intel Research Pittsburgh

Shafeeq Sinnamohideen
Carnegie Mellon University

The term "mobile computing" typically evokes images of a laptop, handheld, or wearable computer. However, the plummeting cost of hardware suggests that pervasive computing infrastructure could minimize the need to carry such devices in the near future. We envision a world in which computers are provided for public use in locations ranging from coffee shops to medical office waiting rooms. We can even imagine each seat on an aircraft or a commuter train being equipped with a computer for the convenience of the current occupant.

In such a world, personal computing will be available anywhere on demand, like light at the flip of a switch. Only when a user starts to use a computer will it acquire his unique customization and state. When he finishes using the computer, this customization and state will disappear from it. Thus, a user could travel hands-free yet be confident of making productive use of slivers of free time anywhere. For this to be a compelling vision from a user's viewpoint, the customization and state acquisition process must be accurate and nearly instantaneous. For it to be a viable business model, the management and system administration costs of pervasive deployments of machines must be low.

To address these challenges, we have developed *Internet Suspend/Resume* (ISR), a pervasive computing technology that rapidly personalizes and depersonalizes anonymous hardware for transient use (http://info.pittsburgh.intel-research.net/project/isr). As its name implies, ISR mimics the closing and opening of a laptop. A user can suspend work on an ISR machine at one location, travel to another location, and resume work there on any other ISR machine. Of course, he can also resume on the original ISR machine if he travels with it—this just happens to be an important special case of the more general capability.

Hardware virtualization and file caching are the keys to ISR's precise customization and simple administration. ISR layers *virtual machines* on a location-transparent distributed file system that aggressively caches data. Each VM encapsulates distinct execution and user customization state. The distributed file system transports that state across both space (from suspend site to resume site) and time (from suspend instant to resume instant). Because of its precise state capture and its efficient state transport, ISR is a key enabling technology for pervasive computing.

Our approach eliminates the need for modifications to guest applications or guest operating systems. In particular, ISR supports unmodified Microsoft software including any of the Windows operating systems and the Office application suite. The average user can thus benefit immediately from ISR.

ISR's thick-client approach to mobility is fundamentally different from thin-client approaches. Although thin-client solutions work well in some situations, they are frustrating to use in failure-prone, congested, or high-latency networks because graphical user interactions such as scrolling and highlighting are painful. In contrast, ISR offers crisp interactive performance under all networking conditions, including total disconnection.

0018-9162/04/$20.00 © 2004 IEEE

User Mobility Evolution

Internet Suspend/Resume is the latest step in a long historical evolution toward user mobility on fixed infrastructure. The earliest form of user mobility dates back to the early 1960s and was supported by timesharing systems attached to "dumb" terminals, at any of which users could access their personal environments.

Thin-client approaches such as InfoPad,[1] SLIM,[2] VNC,[3] and xmove[4] are the modern-day realization of this capability, providing just enough compute power to support GUIs. These strategies work well in some situations but are frustrating to use in failure-prone, congested, or high-latency networks. In such systems, even simple user interactions such as scrolling and highlighting can be tedious because current GUI designs assume very low end-to-end latency. ISR's thick-client approach to mobility is fundamentally different, offering crisp interactive performance under all networking conditions, including total disconnection.

Thick-client strategies became possible after the birth of personal computing more than two decades ago. The vision of being able to use any machine as your own, which dates back at least to the mid-1980s, motivated both location transparency and client caching in the Andrew File System. According to a 1990 article on AFS,[5] "A user can walk up to any workstation and access any file in the shared name space. A user's workstation is 'personal' only in the sense that he owns it."

However, AFS only saves and restores persistent state; it does not preserve volatile state such as the size and placement of windows. In addition, the user sees the client's native operating system and application environment, which in many cases may not be the user's preferred environment.

ISR closely resembles process migration. The key difference is that ISR operates as a hardware-level abstraction, while process migration operates as an OS-level abstraction. In principle, ISR would seem to be at a disadvantage because hardware state is much larger. However, in practice, the implementation complexity and software engineering concerns of process migration have proved to be greater practical challenges. Successful implementations of process migration have been demonstrated, but no widely used OS currently supports it as a standard capability.

References

1. T.E. Truman et al., "The InfoPad Multimedia Terminal: A Portable Device for Wireless Information Access," *IEEE Trans. Computers,* vol. 47, no. 10, 1998, pp. 1073-1087.
2. B.K. Schmidt, M.S. Lam, and J.D. Northcutt, "The Interactive Performance of SLIM: A Stateless, Thin-Client Architecture," *Proc. 17th ACM Symp. Operating Systems and Principles,* ACM Press, 1999, pp. 32-47.
3. T. Richardson et al., "Virtual Network Computing," *IEEE Internet Computing,* Jan./Feb. 1998, pp. 33-38.
4. E. Solomita, J. Kempf, and D. Duchamp, "Xmove: A Pseudoserver for X Window Movement," *The X Resource,* Nov. 1994, pp. 143-170.
5. M. Satyanarayanan, "Scalable, Secure, and Highly Available Distributed File Access," *Computer*, May 1990, pp. 9-21.

DESIGN OVERVIEW AND RATIONALE

Simplicity was the driving force behind our decision to use a VM-based approach for state encapsulation. In 2001, Peter Chen and Brian Noble[1] first suggested that a VM's clean encapsulation of state might simplify state migration. In 2002, we reported the first experimental validation of this hypothesis.[2] Since then, a number of academic and commercial efforts—including Zap,[3] work by Constantine Sapuntzakis and colleagues,[4] and GoToMyPC (www.gotomypc.com)—have explored the problem of transferring user customization across machines.

Our implementation uses VMware Workstation (henceforth just "VMware"), a commercial virtual machine monitor (VMM) for the Intel x86 architecture that operates in conjunction with a host operating system and relies on it for services such as device management. VMware runs on several host operating systems and supports a wide range of guest operating systems including Windows 95/98, Windows 2000/XP, and Linux. VMware maps each supported VM's state to host files. After suspension, these files can be copied to another machine with a similar hardware architecture, and VMware can resume execution of the VM there.

Easy deployment

As the "User Mobility Evolution" sidebar describes, ISR bears some resemblance to a system based on process migration. However, a VM-based system is easier to deploy because it better encapsulates volatile execution state. In addition, a VM-based system tolerates greater disparity between the source and target systems across which migration occurs. Successful process migration requires a close match between host and target operating systems, language runtime systems, and so on. In contrast, VM migration only requires a compatible VMM at the target.

If the resume machine is known with certainty at suspend, direct data transfer can be used to transport VM state. However, users may not resume for many hours or days, and the resume machine may not be the one originally anticipated. In the face of such uncertainty, direct data transfer would require the suspend machine to preserve the user's VM file state for an extended period of time. Further, the suspend machine cannot be turned off or unplugged from the network until resume occurs. This violates our goal of granting ISR sites full autonomy to simplify their management—as long as there is no active user at an ISR machine, unplugging it or turning it off should cause no disruption.

These considerations led us to a design in which the Internet is the true home of VM state, and ISR machines are merely temporary usage points of that state. Because VMware stores state in files, a distributed file system is the obvious choice for ISR Internet storage. Distributed file systems are a mature technology, with designs such as the Andrew File System (AFS) that aggressively cache data at clients for performance and scalability.

Using a distributed file system, with each ISR machine configured as a client, is the key to mobility. Demand caching ensures that relevant parts of VM state follow a user from suspend to resume. Using a distributed file system also simplifies site management. Because an ISR machine holds user-specific state only while active, it can be treated like an appliance. The owner of an ISR site can turn an unused machine on or off, move it, or discard it at will without any centralized coordination or notification. The classic client-server model is thus a better match for ISR than a peer-to-peer design, even though it transfers data in two hops—suspend machine to server, then server to resume machine.

Distributed file systems allow a highly asymmetric separation of concerns, thereby reducing the skill needed to administer ISR machines or to deploy new ones. A small professional staff at an operations center can handle tasks that require expertise, such as backup for disaster recovery, load balancing, and adding new users. We expect that an operations center with file servers and professional staff will often be dedicated to a specific company, university, or Internet service provider. In that case, domain-bridging mechanisms such as AFS cells or Kerberos realms will be valuable when users from different organizations use ISR at semipublic locations such as coffee shops or doctor's offices.

Large VM state

A key obstacle to using ISR is *high resume latency*. Today, a typical VM can range in size from a few Gbytes to many tens of Gbytes. Naive approaches to transferring states this large will result in intolerable resume latencies. Fortunately, numerous real-world considerations can be exploited to mitigate these delays.

Temporal locality. User mobility patterns often exhibit temporal locality. For example, consider a common pattern we envision for ISR: A user begins his day by working at home, suspends work and leaves for his office, resumes work at his office, suspends work at the end of the business day, and resumes work at home after dinner. In another example, a corporate campus worker or a factory supervisor might visit coworkers at various locations many times throughout the day, resuming and suspending work at any of those locations.

In these scenarios, caching VM state at fine granularity will translate temporal locality of ISR machine usage into file reference locality at those machines. Resume latency will be much lower at ISR machines with large persistent file caches because misses will occur only on state that has changed since the last use of that machine by its current user.

Distributed file systems reduce the skill needed to administer ISR machines or to deploy new ones.

Proactivity. With the help of higher-level software, it may sometimes be possible to identify likely resume machines and to proactively transfer VM state to those machines, thereby lowering resume latency. Because proactivity merely requires warming a file cache in our design, the consequences of acting on a bad prediction are mild. A bad prediction could evict useful files from a cache and may waste network bandwidth, but there will be no loss of correctness or need for complex cleanup.

State synthesis. Some VM state rarely changes after initialization. For example, installing Windows XP and the Microsoft Office suite on a small VM configuration can consume one-quarter to one-half its virtual disk capacity. Because this state is identical on other similarly configured VMs, it could be captured on read-only media such as CD-ROMs and distributed to ISR machines with poor Internet connectivity. At resume, ISR software could synthesize large parts of the VM state from the read-only media or the file caches of nearby ISR machines, rather than demand-fetching it over a slow network.

Portable storage

USB and FireWire data storage devices are widely available today as unobtrusive flash-memory key chains or microdrives. If a user is willing to carry such a device, the system could copy part of the user's VM state to it at suspend and obtain the state from it at resume, thereby improving performance at poorly connected ISR sites.

However, using a portable storage device presents a number of problems:

- the entire VM state may not fit on the device;
- the copy operation may take too long at suspend for a user who is in a hurry to leave;
- at resume, the VM state on the device could be stale with respect to current VM state—for example, if the user forgets to update the

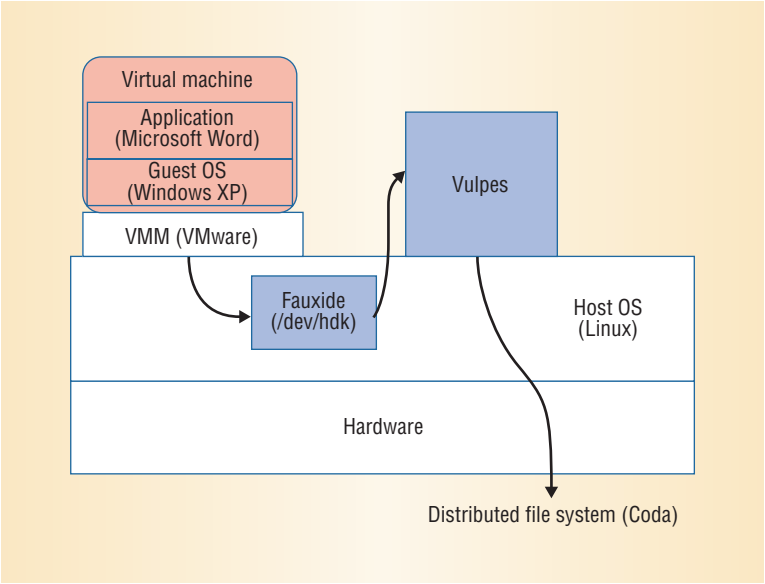

Figure 1. Internet Suspend/Resume host architecture. Fauxide, a loadable kernel module, redirects disk I/O requests on `/dev/hdk` to Vulpes, a user-level process that implements VM state transfer policy and maps VM state to files in Coda.

device at suspend or takes the wrong device when traveling; and
- the device can be lost, broken, or stolen while traveling.

A robust solution must treat portable storage only as a performance assist, not as a substitute, for the underlying distributed file system. Our design makes data on portable devices self-validating: A stale device may not improve performance, but it will never hurt correctness or availability.

ARCHITECTURE AND IMPLEMENTATION

ISR uses the Coda distributed file system[5] for data storage and transport. This choice was based on four key factors:

- complete VM state can fit into a cache because Coda clients cache files on their local disks;
- Coda's support for *hoarding*—anticipatory cache warming—provides a clean interface to exploit advance knowledge of resume machines;
- Coda's support for disconnected and weakly connected operation provides resilience against a wide range of failures and abnormal network conditions; and
- Coda's user-space implementation simplifies experimentation.

ISR represents very large VMware files as a directory tree in Coda rather than as a single file. Virtual disk state is divided into 256-Kbyte chunks, and each chunk is mapped to a separate Coda file. These files are organized as a two-level directory tree to allow efficient lookup and access of each chunk. Our choice of 256 Kbytes is based on a trace-driven analysis of chunk size on performance. The large memory state file is stored in compressed form and uncompressed into `/tmp` just prior to resume.

Figure 1 shows the client architecture that interfaces VMware to Coda. A loadable kernel module, Fauxide, serves as the device driver for a pseudodevice named `/dev/hdk` in Linux. A VM is configured to use this pseudodevice as its sole virtual disk in "raw" mode. Fauxide redirects disk I/O requests to a user-level process, Vulpes, which implements VM state transfer policy. Vulpes also maps VM state to files in Coda and controls the hoarding of those files and their encryption. Because Vulpes is outside the kernel and fully under our control, it is easy to experiment with a wide range of state transfer policies.

Lookaside caching[6] enables a Coda client to use portable devices in a way that reduces vulnerability to human error and device failure. On a cache miss, the client first fetches metadata for the missing object from the server. The Coda metadata definition has been broadened to include the SHA-1 hash of file content. With a valid hash, the client can obtain file content from any matching source. For example, if a mounted portable storage device has a file with matching length and hash, the client can copy it locally rather than fetching the file over a slow network from the file server. Lookaside caching is flexible since misses can be directed to a local file tree, a mounted portable storage device, a nearby NFS or Samba server, a neighboring ISR machine's cache, or even to distributed hash table storage.

Our implementation uses a hash index as a hint to make lookaside caching more efficient. To detect version skew between index and content, Coda recomputes a file's hash after a successful lookaside. In case of a mismatch, Coda redirects the cache miss to the file server. In that case, some small amount of work is wasted on the lookaside path, but consistency is still preserved. Coda's callback mechanism ensures that cached metadata, including the hash, tracks server updates.

EVALUATION

Although our prototype is not yet of production quality, it is robust enough for experiments to evaluate ISR's performance tradeoffs.

From a user's viewpoint, the two dominant questions about ISR performance are "How soon can I begin working?" and "How sluggish is work after I resume?" These questions correspond to the ISR performance metrics *resume latency* and *slowdown*. Ideally, both metrics would be zero. Unfortunately, VM state transfer policies that shrink resume latency may increase slowdown and vice versa. We have therefore conducted a series of controlled experiments to quantify these tradeoffs for typical ISR scenarios.

Experimental methodology

Because ISR is intended for the interactive workloads typical of a laptop environment, we have developed a benchmark that models an interactive Windows user. The Common Desktop Application (CDA) uses Visual Basic scripting to drive Microsoft Office applications such as Word, Excel, PowerPoint, Access, and Internet Explorer. CDA pauses between operations to emulate think time.

Our experimental setup consists of 2.0-GHz Pentium 4 clients connected to a 1.2-GHz Pentium III Xeon server through a 100-Mbps Ethernet. All machines have 1 Gbyte of RAM and run Linux RedHat 7.3. Clients use VMware Workstation 3.1 and have an 8-Gbyte Coda file cache. The VM is configured with 256 Mbytes of RAM and 4 Gbytes of disk storage and runs Windows XP as the guest OS. A NISTNet network emulator controls available bandwidth to servers.

Without ISR support, our setup runs the CDA benchmark in 1,071 seconds. This represents a lower bound on the execution time of any ISR state transfer policy because it eliminates the effects of Fauxide, Vulpes, and Coda. Benchmark time serves as the figure of merit for slowdown.

State transfer policies

Copyout/copyin is the most conservative endpoint in a spectrum of VM state transfer policies. All state is copied out at suspend; resume is blocked until the entire state has arrived. Resume latency can be shortened by three steps, which can be combined to generate a wide range of state transfer policies:

- propagating dirty state to servers before suspend,
- warming the file cache in advance of arrival at the resume machine, and
- letting the user resume before full state has arrived.

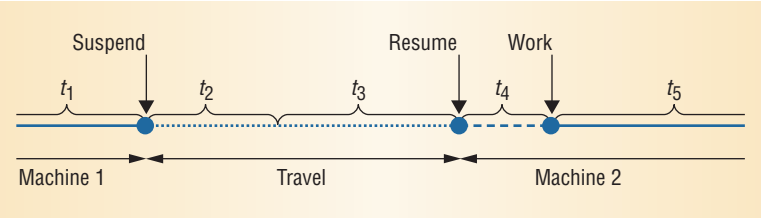

Figure 2. Conceptual ISR timeline. With some state transfer policies, the user can experience significant slowdown during the early part of period t_5.

Figure 2 shows a conceptual timeline for comparing such policies. The figure depicts a user initially working for duration t_1 at machine 1; the user then suspends work and travels to machine 2. In some situations, the system knows (or can guess) machine 2's identity a priori. In other cases, the machine may become apparent only when the user shows up unexpectedly and initiates resume.

The transfer of dirty state from machine 1 to file servers continues after suspend for duration t_2. A period t_3 is then available for proactive file cache warming at machine 2, if known. By the end of t_3, the user has arrived at machine 2 and initiates resume. He experiences resume latency t_4 before he can resume work. He continues working at machine 2 for duration t_5 until he suspends again, and the cycle repeats. With some state transfer policies, the user can experience significant slowdown during the early part of t_5 because some operations block while waiting for the system to transfer missing state.

Baseline policy

The baseline policy is a worst-case straw man. After suspend, all dirty state is transferred to the server during t_2; the period t_3 is empty. At resume, the entire VM state is transferred during t_4 and pinned in the resume machine's cache.

As the "Baseline" column of Table 1 shows, resume latency for this policy is large and highly bandwidth-sensitive. At 100Mbps, resume latency is about 40 minutes. At 10 Mbps, resume latency roughly doubles; it does not increase by a factor of 10 relative to 100 Mbps because Coda, rather than the network, is the bottleneck at the higher bandwidth. Below 10 Mbps, resume latency is intolerable.

The "Baseline" column of Table 2 confirms that slowdown is negligible for the baseline policy. This is because no cache misses occur after resume. Below 100 Mbps, slowdown increases slightly due to Coda background activity such as trickle reintegration.

Table 1. Resume latency, in seconds, for different state transfer policies and bandwidths. Each result is the mean of three trials, with the standard deviation in parentheses.

Bandwidth	Baseline	Fully proactive	Pure demand-fetch	Portable storage lookaside
100 Mbps	2,504 (18)	10.3 (0.1)	14 (0.5)	13 (2.2)
10 Mbps	5,158 (34)	10.2 (0.0)	39 (0.4)	12 (0.5)
1 Mbps	> 9 hours	10.2 (0.0)	317 (0.3)	12 (0.3)
100 Kbps	> 90 hours	11.4 (0.0)	4,301 (0.6)	12 (0.1)

Table 2. Running time, in seconds, of CDA benchmark for different state transfer policies and bandwidths. Each result is the mean of three trials, with the standard deviation in parentheses.

Bandwidth	Baseline	Fully proactive (same as baseline)	Pure demand-fetch	DVD lookaside
100 Mbps	1,105 (9)	←	1,160 (6)	1,141 (36)
10 Mbps	1,170 (47)	←	1,393 (20)	1,186 (17)
1 Mbps	1,272 (65)	←	4,722 (69)	2,128 (34)
100 Kbps	1,409 (38)	←	42,600 (918)	13,967 (131)

Fully proactive policy

If we can predict machine 2, we can define a more aggressive state transfer policy. At machine 2, this policy shifts the entire state transfer time from t_4 to earlier periods in the ISR timeline. During t_3—or earlier, for any state already available at the servers—machine 2 transfers all updated state to its local cache. At resume, all that remains is to launch the VM. This policy is likely to be most effective when a user is working among a small set of sites, such as when alternating between home and work.

As the "Fully proactive" column of Table 1 shows, resume latency is bandwidth-independent and very small (10-11 seconds) because all necessary files are already cached. Post-resume ISR execution is indistinguishable from the baseline. Clearly, this policy is very attractive when feasible.

The minimum travel time needed for a fully proactive policy is $t_2 + t_3$. In the best case, the resume machine is known well in advance, and its cache has been closely tracking the suspend machine. Only the residual dirty state (about 47 Mbytes at the midpoint of the CDA benchmark) must be transferred. This results in a minimum travel time of 45 seconds on a 100-Mbps network and 90 seconds on a 10-Mbps network. These are credible bandwidths and walking distances between collaborating workers at a typical university campus, corporate site, or factory. At 1 Mbps, which is available via DSL or cable modem to many homes today, the best-case travel time is roughly 14 minutes. The attractiveness of ISR in such scenarios is likely to increase over time because networks will improve, but commutes are unlikely to shorten.

Pure demand-fetch policy

Suppose a user arrives unexpectedly at an ISR machine. To keep t_4 short, a pure demand-fetch policy only retrieves memory state during this period. The transfer of VM disk state occurs on demand over t_5, resulting in Coda cache misses that cause slowdown.

For our prototype and benchmark, the state transferred during t_4 is about 41 Mbytes. The time to transfer this state is a lower bound on resume latency for this policy. As the "Pure demand-fetch" column of Table 1 shows, resume latency increases from under one minute at LAN speeds to more than one hour at 100 Kbps.

The slowdown for a pure demand-fetch policy is highly sensitive to bandwidth, as Table 2 shows. The total benchmark time increases from 1,105 seconds without ISR to 1,160 seconds at 100 Mbps; this represents a slowdown of about 8.3 percent. As bandwidth drops, the slowdown increases to 30.1 percent at 10 Mbps, 340.9 percent at 1 Mbps, and well over an order of magnitude at 100 Kbps. Although slowdowns below 100 Mbps will undoubtedly be noticeable, their impact must be balanced against the improvement in user productivity resulting from the ability to work anywhere, even at unexpected locations.

Demand-fetch with lookaside policy

Lookaside caching can improve the pure demand-fetch policy in many ways. If a user is willing to wait briefly at suspend, the VM memory state file can be written to a portable storage device. At the resume machine, lookaside caching from the device can reduce t_4. If read-only or read-write media with partial VM state are available at the resume machine, lookaside caching can use them to reduce the cost of cache misses during t_5.

Table 1 presents results for the first of these scenarios. A comparison of the "Pure demand-fetch" and "Portable storage lookaside" columns of this table show a noticeable improvement below 100

Mbps and a dramatic improvement at 100 Kbps. A resume time of just 12 seconds rather than 317 seconds (at 1 Mbps) or 4,301 seconds (at 100 Kbps) can make a world of difference to a user who only has a few minutes of time available to work while in a coffee shop or a waiting room.

To explore the impact of lookaside caching on slowdown, we used a DVD as a lookaside device. The DVD contained VM state captured after installation of Windows XP and the Microsoft Office suite but before any user-specific or benchmark-specific customizations. Roughly half of the file cache misses are satisfied through lookaside. As Table 2 shows, using lookaside consistently reduces benchmark time relative to a pure demand-fetch policy, particularly at lower bandwidths.

Seamless ubiquitous access to a user's uniquely customized personal environment is the holy grail of mobile computing. ISR represents an important step toward this goal. By exploiting advance knowledge of travel, transferring VM state incrementally, and using portable storage, we have shown that the performance cost of seamless mobility can be made acceptable. Using these techniques, resume latency in ISR is little more than the typical delay a user experiences when opening a laptop. By leveraging the consistency of a distributed file system, ISR is robust in the face of many human errors and is tolerant of poorly managed environments.

ISR is a versatile mechanism with value beyond mobile computing. By severing the tight binding between personal computing state and computer hardware, ISR frees PC state for use in many innovative ways. For example, replicas of PC state could be saved at widely separated Internet sites to allow disaster-recovery after catastrophic failures. As another example, a time-stamped snapshot of PC state could be created, encrypted, digitally signed, and asynchronously transmitted to a verification authority to demonstrate compliance with security advisories. In both cases, a user's foreground activity can continue with minimal disruption.

In the future, we plan to explore these and other exciting opportunities that ISR makes possible. We also expect to address security concerns: Users must be confident that anonymous computers are safe for them to use. We currently assume that deployed machines are secure, but we are planning research to enforce this assumption. To address this difficult challenge, we expect to leverage previous work on secure coprocessors[7,8] and ongoing research by the Trusted Computing Group (www.trustedcomputinggroup.org) and Microsoft's Next-Generation Secure Computing Base (www.microsoft.com/resources/ngscb/default.mspx). ■

References

1. P.M. Chen and B.D. Noble, "When Virtual Is Better Than Real," *Proc. 8th Workshop Hot Topics in Operating Systems,* IEEE CS Press, May 2001, pp. 133-138.
2. M. Kozuch and M. Satyanarayanan, "Internet Suspend/Resume," *Proc. 4th IEEE Workshop Mobile Computing Systems and Applications,* IEEE CS Press, June 2002, pp. 40-46.
3. S. Osman et al., "The Design and Implementation of Zap: A System for Migrating Computing Environments," *Proc. 5th Symp. Operating Systems Design and Implementation,* ACM Press, Dec. 2002, pp. 361-376.
4. C.P. Sapuntzakis et al., "Optimizing the Migration of Virtual Computers," *Proc. 5th Symp. Operating Systems Design and Implementation,* ACM Press, Dec. 2002, pp. 377-390.
5. M. Satyanarayanan, "The Evolution of Coda," *ACM Trans. Computer Systems,* May 2002, pp. 85-124.
6. N. Tolia et al., "Integrating Portable and Distributed Storage," *Proc. 3rd Usenix Conf. File and Storage Technologies,* Usenix Assoc., Mar. 2004, pp. 227-238.
7. S.W. Smith and V. Austel, "Trusting Trusted Hardware: Towards a Formal Model for Programmable Secure Coprocessors," *Proc. 3rd Usenix Workshop Electronic Commerce,* Usenix Assoc., 1998, pp. 83-98.
8. J.D. Tygar and B. Yee, "Dyad: A System for Using Physically Secure Coprocessors," *Proc. Joint Harvard-MIT Workshop Technological Strategies for Protecting Intellectual Property in the Networked Multimedia Environment,* 1993; www.cni.org/docs/ima.ip-workshop/Tygar.Yee.html.

Michael Kozuch *is a senior researcher at Intel Research Pittsburgh, where his focus is on novel uses of virtual machine technology. Kozuch received a PhD in electrical engineering from Princeton University. Contact him at michael.a.kozuch@intel.com.*

Mahadev Satyanarayanan *is the Carnegie Group Professor of Computer Science at Carnegie Mellon University and the founding director of Intel Research Pittsburgh. His research interests span mobile computing, pervasive computing, and dis-*

tributed systems. Satyanarayanan received a PhD in computer science from Carnegie Mellon University. He is a member of the IEEE Computer Society, a Fellow of the ACM and the IEEE, and the founding editor in chief of IEEE Pervasive Computing. *Contact him at satya@cs.cmu.edu.*

Thomas Bressoud *is an assistant professor of computer science in the Department of Mathematics and Computer Science at Denison University and an affiliate researcher at Intel Research Pittsburgh. His research interests include fault tolerance, content-addressable storage, and distributed systems. Bressoud received a PhD in computer science from Cornell University. Contact him at bressoud@denison.edu.*

Casey Helfrich *is a research engineer at Intel Research Pittsburgh. His research interests include distributed systems, virtualization of hardware, and building research systems. Helfrich received a BS in computer science from Carnegie Mellon University. Contact him at casey.j.helfrich@intel.com.*

Shafeeq Sinnamohideen *is a PhD student in the Computer Science Department at Carnegie Mellon University working on the Internet Suspend/Resume project at Intel Research Pittsburgh. His main research interest is in wide-area distributed file systems. Sinnamohideen received an MS in electrical and computer engineering from Carnegie Mellon University. Contact him at shafeeq+www@cyrus.watson.org.*