Available online at www.sciencedirect.com

SCIENCE DIRECT®

ELSEVIER

Pervasive and Mobile Computing 1 (2005) 157–189

pervasive and mobile computing

www.elsevier.com/locate/pmc

Towards seamless mobility on pervasive hardware

M. Satyanarayanan[a,*], Michael A. Kozuch[b], Casey J. Helfrich[b], David R. O'Hallaron[a]

[a]*School of Computer Science, Carnegie Mellon University, Pittsburgh PA, United States*

[b]*Intel Research Pittsburgh, Pittsburgh PA, United States*

Received 1 March 2005; accepted 23 March 2005
Available online 31 May 2005

Abstract

Preserving one's uniquely customized computing environment as one moves to different locations is an enduring challenge in mobile computing. We examine why this capability is valued so highly, and what makes it so difficult to achieve for personal computing applications. We describe a new mechanism called *Internet Suspend/Resume (ISR)* that overcomes many of the limitations of previous approaches to realizing this capability. ISR enables a hands-free approach to mobile computing that appears well suited to future pervasive computing environments in which commodity hardware may be widely deployed for transient use. We show that ISR can be implemented by layering virtual machine technology on distributed file system technology. We also report on measurements from a prototype that confirm that ISR is already usable today for some common usage scenarios.

Keywords: Mobile computing; Pervasive computing; Personal computing; Internet Suspend/Resume; Virtual machines; VMware; Distributed file systems; Coda; Portable storage; Seamless mobility; Human attention; Thin clients; Process migration

* Corresponding address: Carnegie Mellon University, School of Computer Science, 5000 Forbes Avenue, 15213 Pittsburgh, PA, United States. Tel.: +1 412 268 3743; fax: +1 412 268 4136.

E-mail addresses: satya@cs.cmu.edu (M. Satyanarayanan), michael.a.kozuch@intel.com (M.A. Kozuch), casey.j.helfrich@intel.com (C.J. Helfrich), droh@cs.cmu.edu (D.R. O'Hallaron).

1574-1192/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.pmcj.2005.03.005

1. Introduction

The dawn of the 21st century has seen explosive interest in pervasive computing. Coming roughly a decade after its founding manifesto by Mark Weiser [39], much of this research addresses issues that may seem exotic relative to mainstream computing. Noticeably absent from the discourse is any mention of what will happen to the enormous installed base of personal computing applications, and the substantial investment people have made in learning to use them effectively. Will spreadsheets, word processors, illustration programs, tax preparation software, and such continue to exist in a pervasive computing world? We believe that the need to calculate, write, draw, prepare tax returns and so on is unlikely to vanish. If these applications will not disappear, what can pervasive computing do for them? Can this familiar world be improved in some fundamental way? These are the questions we explore in this paper.

The plummeting cost of hardware hints at disruptive change on the horizon. Some day, pervasive deployment of commodity hardware may liberate users from carrying a laptop or having to use a specific desktop. Imagine a world where coffee shops, airport lounges, dental and medical offices, and other semi-public spaces provide desktop or laptop hardware for their clientele. In such a world, users could travel hands-free yet make productive use of slivers of time anywhere. We envision a world in which the “personal” (that is, user customization) aspect of personal computing is retained, but the “computing” aspect becomes a commodity. The thesis of this paper is that seamless mobility of users in such a world can be achieved without changing today’s well-entrenched base of personal computing applications.

We begin by examining the characteristics of personal computing applications and the importance of seamless mobility. Then, in Section 3, we compare existing design strategies for seamless mobility and identify their strengths and weaknesses. From these roots, we derive a new strategy. We describe the design, implementation and evaluation of this new strategy in Sections 4–7. We examine the assumptions and limitations of our solution in Section 8, and conclude with a summary in Section 9.

2. Background

2.1. Whither personal computing?

Since the birth of personal computing in the early 1980s, a vibrant ecosystem of operating systems (OSes), graphical user interfaces (GUIs), applications, user expectations and computing practices has evolved. The most valuable part of this ecosystem is a rich collection of applications ranging from spreadsheets and word processors to CAD tools and medical imaging aids. We refer to them as *personal productivity applications* because their primary goal is to amplify the cognitive ability of a user. They share certain traits:

- Their workload typically involves long think times, during which the processor is idle. Yet, crisp response is vital whenever a user emerges from the thinking phase and interacts with the application. Sluggish response distracts the user and hurts productivity.

- Use of the application typically involves critical persistent state (files) that is unique to each user.
- Users customize each application to a significant extent. These customizations are rarely frivolous; rather, they help a user tune the application to his specific cognitive preferences and thus improve his productivity.
- Over time, these applications have evolved into tightly integrated application suites that each dominate their specific niche. Microsoft's Office suite is the best-known of these, but many other examples exist. Skill in using the dominant application suite for a niche is essential for professional success today.

This entire edifice rests upon the crisp interactive response made possible by low-latency access to a dedicated local processor. It represents our collective reward for the move from timesharing to personal computing. Unfortunately, in making that move we gave up a valuable capability — seamless mobility across hardware. A user could walk up to any "dumb terminal" attached to a timesharing system and access his entire personalized computing environment there. Since these terminals were stateless, differences between them were only superficial.

In contrast, moving between two random personal computers today is rarely a seamless experience. There are likely to be major differences in OS and application versions and customizations. The persistent states on the two computers are completely disjoint, thus requiring explicit management of files. In contrast to dumb terminals, personal computers are painfully stateful!

Is it possible to preserve the hard-won benefits of personal computing, while regaining the seamless mobility that came for free with timesharing? Can we do so without incurring the problem that led to the death of timesharing, namely its poor interactive response under heavy load? That is the challenge we address.

2.2. Why seamless mobility matters

Seamlessness has been an important attribute of mobile computing since the birth of the field in the early 1990s. What does the term "seamless" mean, and why it is so important? A seamless transition is one that involves a potentially disruptive state change, yet hardly distracts the user. The classic example of this is cell phone handoff between two access points — the user is never aware of the transition. Another example is the transition between connected and disconnected operation in a system such as Coda [16].

Low distraction is the defining characteristic of seamlessness. In other words, very little *user attention* is consumed. User attention is the most precious resource in mobile computing. Moore's Law does not apply to it, as it does to many other resources such as CPU power, network bandwidth, and memory capacity. As a result, human attention does not improve even over many decades.

The need to treat human attention as a consumable resource was first recognized by Herb Simon [31]: "*What information consumes is rather obvious: it consumes the attention of its recipients. Hence a wealth of information creates a poverty of attention and a need to allocate that attention efficiently among the overabundance of information sources that might consume it*". Although this particular quote is from 1971, Simon's focus on human attention dates back to the late 1940s [30]. When mobile, some user attention is consumed by basic needs such as avoiding obstacles or adjusting to an unfamiliar physical environment. A scarce resource is thus made even scarcer by the demands of mobility. Seamlessness is a highly valued quality in this situation because it lowers the demand on user attention. Restoring a familiar environment, consistently meeting user expectations, and avoiding surprises all help achieve seamlessness.

3. Design strategies for seamless mobility

Over time, a variety of strategies for achieving seamless mobility have emerged. We briefly review these in Sections 3.1–3.5 and then compare their strengths and weaknesses using a uniform framework in Section 3.6. We then use this comparison to motivate a new strategy, which is the subject of the rest of the paper.

3.1. Thin client

The earliest form of user mobility, dating back to the early 1960s, was supported by timesharing systems attached to dumb terminals. Thin clients are the modern-day realization of this capability. A thin client consists of a display, keyboard and mouse combined with sufficient processing power and memory for graphical rendering and network communication with a compute server using a specialized protocol. All application and OS code is executed on the server. The client has no long-term user state and needs no disk. Many thin-client protocols exist, and their relative merits have been explored by Lai and Nieh [20].

Thin-client computing is similar to timesharing in that even trivial user–machine interactions incur queuing delay on a resource that is outside user control. In both cases, queuing delay is acutely sensitive to the vagaries of the external computing environment. In timesharing, the shared resource is the processor. In thin-client computing, it is the network. The adequacy of thin-client computing is highly variable, and depends on both the application and the available network quality. If near-ideal network conditions (low latency and high bandwidth) can be guaranteed, thin clients offer a good computing experience. As network quality degrades, interactive performance suffers. It is latency, not bandwidth, that is the greater challenge. Tightly coupled tasks such as graphics editing suffer more than loosely coupled tasks such as web browsing. The combination of the worst anticipated network quality and the most tightly coupled task determines whether a thin client is satisfactory. The extreme case of network disconnection cannot be tolerated by thin clients.

3.2. Distributed file system

Location transparent distributed file systems such as AFS [14] and Coda [28] have long offered a limited form of seamless mobility. 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. To quote a 1990 AFS paper [27]: “*User mobility is supported: 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*.”

There are a number of ways in which this capability falls short of the ideal of seamless mobility. Only persistent state is saved and restored. Volatile state, such as the execution states of interactive applications, is not preserved. Another shortcoming is that the user sees the native operating system and application environment of the client. This part of his computing environment is therefore not seamlessly preserved across machines.

3.3. Process migration

Process migration is an operating system capability that allows a running process to be paused, relocated to another machine, and continued there. It represents seamless mobility at the granularity of individual processes, and has been a research focus of many experimental operating systems built in the past 20 years. Examples include Demos [24], V [37], Mach [41], Sprite [8], Charlotte [1], and Condor [40]. These independent validation efforts have shown beyond reasonable doubt that process migration can indeed be implemented with acceptable efficiency.

Yet, in spite of its research popularity, no operating system in widespread use today (proprietary or open source) supports process migration as a standard facility. The reason for this paradox is that process migration is excruciatingly difficult to get right in the details, even though it is conceptually simple. A typical implementation of process migration involves so many external interfaces that it is easily rendered incompatible by a modest external change. In other words, process migration is a brittle abstraction. Long-term maintenance of machines with support for process migration involves too much effort relative to the benefits it provides.

3.4. Language-based mobility

The audience of a language-based approach to mobility is an application developer rather than a user. However, the approach is relevant to our discussion if all of a user's applications are written in this language. A number of advantages follow from limiting discourse to applications written in a specific language. By careful language definition, the concept of seamless mobility can be built into the programming language and supported by its runtime system. This support can be fine-grained (that is, at the granularity of individual objects) rather than coarse-grained (as in the case of process migration). As a result, the approach can support many configurations where parts of an application execute at one site while others execute at another site. This is a much richer space of possibilities than that offered by other approaches. Many corner cases that would be difficult to handle seamlessly are avoided by careful language specification.

The best early example of work in this genre is Emerald [15]. A more recent example is *one.world* [12]. The growth in popularity of Java and its support for *remote method invocation* [23] have made this approach feasible and relevant to a wide range of computing environments.

3.5. Aura task migration

A radically different approach to mobility is offered by the Aura system [10]. Rather than trying to preserve as much of an environment as possible, Aura abstracts away most of its application-specific components. What it strives to preserve across a move is the highest layer of this abstraction, called the *task layer*. The same task may be realized in different ways on different hardware platforms, possibly using entirely different applications. What is preserved is the notion of progress through the task, much like a checkpoint in workflow software.

On each Aura client, a layer above individual applications and services called *Prism* manages task migration. By explicitly representing user intent, Prism makes available to the rest of the system a powerful basis on which to adapt or anticipate user needs. The implementation runs on Windows XP, and supports task migration for the Microsoft Office suite. Sousa and Garlan [33] provide further details on Prism and task migration in Aura.

3.6. Comparing strategies

A critical comparison of existing strategies for seamless mobility may point the way to a better solution. With this goal in mind, we have identified a set of attributes along which strategies for seamless mobility can be compared:

- *Seamlessness:* How smooth and unobtrusive is the user experience when moving from one site to another? How close is the environment at the new site to that at the old? How much re-familiarization and adjustment does the user have to make? How distracted is the user in adjusting from the old world to the new?
- *Solution Generality:* Does the solution work for all applications or only for a few? Do applications have to be modified, recompiled or relinked? How stringent are the language, programming model and related constraints imposed on applications?
- *Network Resilience:* How critically dependent is the solution on the network? How badly does poor network quality (low bandwidth or high latency) hurt user experience? How long does the connectivity to the old site from the new have to last? Can the user continue work in the face of network disconnections?
- *Ubiquity:* How easy is widespread deployment of the solution? How many critical assumptions does it make about the similarity of the old site and the new? How robust is the solution in the face of system heterogeneity? What is the level of expertise and quality of system administration needed to sustain a deployment in the real world?
- *Network Load:* How large is the volume of data transferred? Is the network workload bursty or uniform?
- *Implementation complexity:* How hard is the solution to build, debug and get right? How difficult is it to maintain in the face of application, operating system and hardware changes?

Table 1 shows how each of the strategies discussed in Sections 3.1–3.5 maps to the above attributes. For simplicity, we have chosen scores of “high”, “medium”, and “low” to characterize a strategy with respect to an attribute. A score of “high” is best for the attributes of seamlessness, solution generality, network resilience, and ubiquity. A score of “low” is best for the attributes of network load and implementation complexity.

The thin-client approach receives top scores on almost all attributes. Seamlessness is excellent because the old and new environments are indistinguishable except possibly for minor differences in the display and keyboard/mouse. The approach works for all

Table 1
Comparison of strategies for seamless mobility

	Seamlessness	Solution generality	Network resilience	Ubiquity	Network load	Implementation complexity
Thin client	high	high	low	high	low	low
Distributed file system	medium	high	high	medium	medium	medium
Process migration	high	high	medium	low	medium	high
Language-based mobility	high	medium	medium	medium	medium	medium
Aura task migration	low	low	high	low	low	medium
ideal solution	high	high	high	high	low	low
plausible near-ideal	high	high	high	high	*likely high*	*likely high*

applications and requires no modifications to the operating system. It is easy to implement and trivial to deploy, thus earning it top scores on implementation complexity and ubiquity. Keystrokes, mouse movement and display updates generate only modest network load. The Achilles heel of thin clients is their poor network resilience. They cannot tolerate disconnection, and user experience degrades when network quality is poor.

Distributed file systems score well on solution generality because they are integrated with the operating system. An application that is written to use local files works unchanged on remote data. The use of caching keeps network load acceptable except when files are very large. Cache misses and update propagation require use of the network, but designs such as Coda keep this dependence to a bare minimum. Hence, this approach receives a high score for network resilience. It does not receive the top score for seamlessness because a user sees the native operating system environment, which may differ at the old and new sites. It does not receive the top score for ubiquity because it requires changes or extensions to the operating system.

Process migration scores well on seamlessness and solution generality. Since it is tightly integrated with the operating system, it re-creates an application's execution environment with great fidelity at the destination. However, for reasons discussed in Section 3.3, it scores poorly on ubiquity and implementation complexity.

Language-based mobility scores well on seamlessness since this property is well supported by the language and its runtime system. However, it does not work for applications that are written in a different language. The scores for solution generality and ubiquity are hence lower.

Aura task migration allows for the possibility of a different application and modality of interaction when a user moves from an old site to a new one. By design, it therefore scores low on the seamlessness attribute. Solution generality and ubiquity are low since this approach is critically dependent on Aura. Network resilience is high, since the old site is not accessed after task migration.

3.7. Deriving a better strategy

From Table 1, it is clear that no existing strategy scores high on all of the first four attributes. In other words, no existing strategy offers a high degree of seamlessness for all applications while remaining easy to deploy and resilient to poor network quality. Can one invent a new strategy with these properties?

A clue to solving this problem comes from the adage that there is no free lunch. In other words, there will have to be a tradeoff. This suggests that any solution that scores well on the first four attributes is likely to score poorly on the last two. A plausible solution is therefore likely to impose substantial network load and involve considerable implementation complexity. The last row of Table 1 illustrates the attributes of such a plausible solution.

In the rest of this paper, we describe a strategy for seamless mobility called *Internet Suspend/Resume (ISR)* that has these attributes. Recognizing that network bandwidth will continue to improve, ISR exploits this ample bandwidth by encapsulating the *entire* state of a personal computer (including its disks) and delivering it anywhere on demand. It thus builds upon the concept of a caching file system, with the significant difference that it is now entire machine state (not just user files) that is delivered through caching. The volume of state transferred is now much larger. This is the tradeoff being made for more precisely and completely recreating a user's entire computing environment.

4. Internet Suspend/Resume

4.1. Background

As its name implies, ISR was inspired by the suspend/resume feature of laptops. That capability was created by laptop designers as a means of extending battery life. To enter the suspended state one just closes the cover of a laptop. In that state very little energy is used. When the cover is opened, the state at suspend is restored with near-perfect fidelity within a few seconds. In the context of this paper, suspend/resume achieves seamless mobility at the cost of having to carry hardware.

Our key insight was to recognize that the suspend/resume metaphor could be extended to situations where a user carries no hardware. In other words, one can logically suspend a machine at one Internet site, travel to some other site and then seamlessly resume work there on another machine. By mimicking the suspend/resume feature of laptops we gain two advantages. First, this is a simple and well-understood metaphor for users. Second, operating systems and applications have already been evolved to gracefully cope with a number of discontinuities across suspend and resume. For example, a dynamically obtained IP address may change when a laptop is resumed at a location far from where it was suspended. As another example, most laptop applications that use the network transparently re-establish TCP connections that are broken on suspend. As a third example, USB devices attached to a docking station are missing when a laptop is resumed by a user on his travels; they reappear upon return. By leveraging existing mechanisms and user expectations, ISR greatly reduces the need to modify operating systems and applications, and the need to re-educate users.

4.2. Hypothetical scenario

ISR inspires many futuristic scenarios, and effectively creates a new computing paradigm. Imagine, for example, this hypothetical ISR scenario from 2020:

The alarm rings to begin a hectic Thursday for Shanta. She is soon in her study, working on the slides for the class that she will be teaching this morning. Soon it is time to leave. She clicks on the suspend icon on the screen, and her work is saved. On the commute to work, she stops at her doctor's office for a simple medical test. Unfortunately, the technician is backed up and Shanta has to wait much longer than expected. She borrows a wireless laptop from the office staff, logs in, resumes her work, and finishes a few more slides for class before the technician is ready for her. Shanta suspends her work, hands back the laptop, goes in for the test, and is soon on the road again. Reaching her office just 10 min before class, Shanta uses her desktop to put the finishing touches on her lecture slides and then leaves for class. Each classroom is equipped with an LCD projector connected to a networked computer. Shanta resumes where she left off in her office and proceeds to give an entertaining and insightful lecture.

When she returns to her office from class, Shanta finds that the computing services staff has replaced her desktop by a newer and much more powerful machine that she had recently ordered. Fortunately, she does not have to waste any time in setting up the new machine, copying files or customizing it in any way. All she does is to log in, and she finds her work just where she left off at the end of class. Her day proceeds as planned.

Late in the afternoon, Shanta leaves for the airport and takes a flight for a business meeting the next day. The fold-out tray at each seat has come a long way from its simple ancestor of the early 21st century. When it isn't being used as a tray, it doubles as a screen/keyboard/mouse that is connected to a rack of blade servers at the back of the aircraft. Shanta resumes work where she left off in her office, completing the slides for her talk and her spreadsheet calculations for the budget discussions the next day. High-bandwidth wireless Internet connectivity is available from the aircraft, but it is very expensive. Shanta therefore chooses to work disconnected during the flight.

When Shanta checks in to her hotel, the clerk at the front desk hands her a laptop for use during her visit. There is a drop-off site near the airport departure gates where she can return the laptop the next day. Shanta works late into the night, completing her slides and spreadsheet calculations. Her meetings the next day are intense, but ultimately successful. The deal is clinched, and Shanta's hosts invite her to an early dinner to celebrate.

When Shanta returns to her rental car after dinner, she is dismayed to find that her luggage (including laptop) has been stolen. Fortunately, nothing on the laptop is irreplaceable. Her precious computing state (including many highly confidential files) were saved on servers on the Internet when she last suspended work. All residual personal state on the laptop was encrypted, so the damage and inconvenience from the loss of the laptop is only its hardware cost. Except for a small deductible, Shanta's homeowner insurance will cover everything. On the flight home, Shanta orders a drink and falls asleep . . . .

4.3. Realization

Although the scenario in the previous section is science fiction, the mechanism on which it is predicated is implementable today. ISR can be realized by combining two off-the-shelf technologies: *virtual machine* (VM) technology and *distributed file system* technology. Each VM encapsulates distinct execution and user customization state. The distributed file system transports that state across space (from suspend site to resume site) and time (from suspend instant to resume instant).

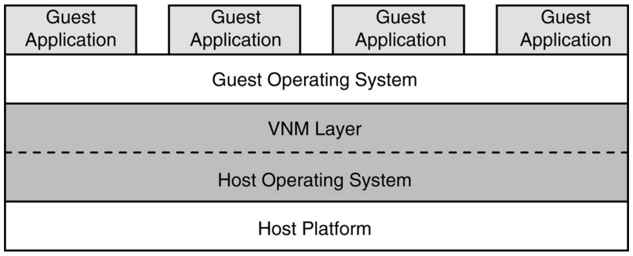


Fig. 1. Virtual machine layered on host OS.

A VM is an emulated hardware abstraction whose fidelity is so good that neither system nor application software executing within the VM can tell that it is not directly executing on bare hardware [11]. Emulation is performed with the assistance of a *virtual machine monitor* (VMM). All our work on ISR until now has used VMware Workstation (abbreviated to just "VMware"), a commercial VMM for the Intel IA-32 architecture [13]. VMware operates in conjunction with a *host* operating system, relying on it for services such as device management. Fig. 1 illustrates this situation. VMware runs on several host OSes and supports a wide range of guest OSes including Windows 95/98, Windows 2000/XP, and Linux. VMware maps the state of a VM to host files. When a VM is suspended, its volatile state is also saved in a file. After suspension, the VM's files can be copied to another machine with a similar hardware architecture; there, VMware can continue execution of the machine.

Distributed file systems are a mature technology, with designs such as AFS that aggressively cache data at clients for performance and scalability. The use of such a distributed file system, with all ISR sites configured as clients, is the key to mobility and simplified management of ISR sites. Demand caching at a resume site ensures that relevant parts of VM state follow a user from suspend to resume. Since an ISR site holds no user-specific state except during active use, it can be treated like an appliance. An idle site can be turned off, moved, or discarded at will without centralized coordination or notification.

The highly asymmetric separation of concerns made possible by a distributed file system reduces the skill level needed to manage ISR sites. Little skill is needed to maintain machines or to deploy new ones. System administration tasks that require expertise (such as backup, restoration, load balancing, and addition of new users) are concentrated on a few remotely located servers administered by a small professional staff. We expect that server hardware and the professional staff to administer them will often be dedicated to a specific organization such as a company, university or ISP. Since locations such as coffee shops and doctors' offices are likely to be visited by ISR users belonging to many different organizations, domain-bridging mechanisms such as AFS *cells* or Kerberos *realms* will be valuable. Fig. 2 illustrates how a deployment of ISR might be organized.

4.4. Evolution

Since the fall of 2001 we have evolved the ISR concept through three prototype implementations: ISR-1, ISR-2 and ISR-3. All three prototypes have used VMware as the virtual machine monitor and Linux as the host operating system. ISR-1 was a proof-of-concept implementation that used NFS for file storage. By the end of 2001, this prototype

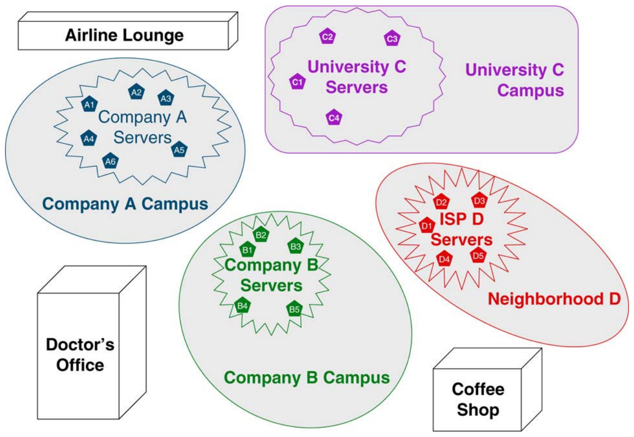


Fig. 2. Hypothetical ISR deployment.

had confirmed that layering a VM on a distributed file system could indeed yield seamless suspend and resume functionality [18]. Our next step was to improve the performance of ISR and to expand its functionality to support disconnected operation and use of portable storage devices. This led to a new prototype, ISR-2, that used Coda as its distributed file system. From early 2002 until late 2004, we used ISR-2 to explore many aspects of the ISR concept and to develop techniques for improving its performance and functionality [19]. In late 2004, we turned our attention to real-life deployment of ISR and to gaining hands-on usage experience. This required us to create a new prototype. ISR-3 subsumes much of the code and functionality of ISR-2, but offers simpler installation and usage as well as greater flexibility in system configuration [17]. In ISR-3, Coda is only one of many possible mechanisms that can be used for distributed storage of VM state. The structure of ISR-3 makes it easy to replace Coda with alternatives such as AFS or Lustre [29], or to use a built-in storage layer based on HTTP and SSH.

We focus on ISR-2 in this paper. The system descriptions in Sections 5 and 7 also apply to ISR-3 if Coda is used as the distributed storage layer. However, the performance measurements reported in Sections 6 and 7 apply specifically to ISR-2 — we have not yet conducted a performance evaluation of ISR-3. With rare exception, we do not distinguish between specific prototypes in the rest of this paper. It will usually be clear from the context whether the term "ISR" means the broad concept or "ISR-2".

5. ISR design and implementation

5.1. Distributed file system

We had a choice of three distributed file systems for ISR: NFS, AFS and Coda. Although NFS is the most widely supported of these, we did not use it for two reasons. First, NFS only does caching of blocks in memory; it does not cache data persistently in the local file system. Hence, the cache size at an ISR site can be no larger than its memory size, which is typically much smaller than total VM state size. This limits ISR's ability to take advantage of temporal locality of access to VM state. Second, our goal is to support ISR anywhere on the Internet, including locations with less than optimal network connectivity. NFS is designed for LAN access, and tends to perform poorly in WAN environments.

Both AFS and Coda clients use the local disk as a file cache. AFS is a much more robust and mature implementation, with an extensive deployment base and better performance. In spite of this, we decided to use Coda for the following reasons.

First, Coda's support for hoarding (anticipatory cache warming) provides a clean interface to take advantage of advance knowledge of resume site. Although originally developed to cope with disconnection, this mechanism can also be used to improve performance by warming a cache in advance of use. All that is needed is a list of file names and their relative importance.

Second, Coda supports trickle reintegration which is valuable for propagating dirty client state to file servers in the background from poorly connected ISR sites. This reduces the amount of dirty state waiting to be propagated at suspend. Although a user can walk away immediately after suspend, the owner of the ISR site cannot turn off or unplug it until its cache state is clean. Trickle reintegration improves site autonomy by shortening this window of vulnerability.

Third, the AFS client implementation is entirely in the kernel. In contrast, the Coda client implementation is almost entirely in user space; only a small module for redirecting file references resides in the kernel. The user space implementation simplified our extensions for use of portable storage, as explained in Section 7.2.

5.2. Security model

VM state is encrypted by ISR client software before being stored in Coda. Neither servers nor persistent client caches contain VM state in the clear. Compromise of Coda servers can, at worst, result in denial of service. Compromise of a client after a user suspends can at worst prevent updated VM state from being propagated to servers, also resulting in denial of service. Even in these situations, the privacy and integrity of VM state are preserved.

When a user walks up to an ISR client machine, he must explicitly authenticate himself via a mechanism such as Kerberos [34] before he can resume. We have not addressed the much harder problem of establishing that an ISR client is safe to use. It is up to the user to make this judgement. This is an acceptable solution in restricted locations such as home or work, but does not scale to unrestricted locations. Scenarios like Section 4.2 will require mechanisms that enable a user to be confident that the hardware and software on a random client have not been compromised. Many researchers are investigating this difficult problem [32,36,38], and we plan to leverage workable solutions that emerge.

5.3. Data layout

Caching VM state at fine granularity is important for taking advantage of temporal locality in user movements. Large, monolithic VM state files are therefore not a good match for Coda's policy of caching entire files. The mismatch is especially acute for files corresponding to virtual disks, which can be many tens of GB in size. Our solution is to

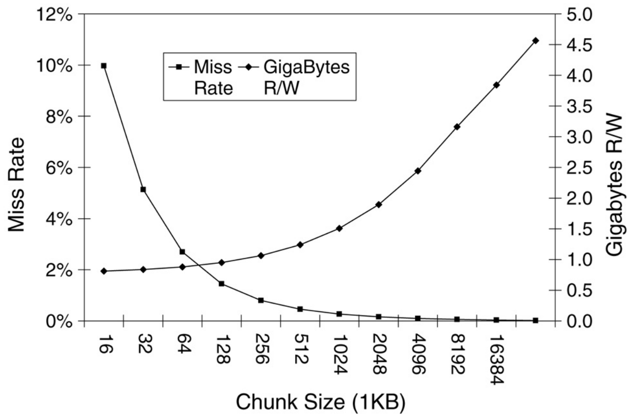


Fig. 3. Trace-driven analysis of chunk size.

represent such large files as a directory tree rather than as a single file. Virtual disk state is divided into 128 KB 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 128 KB for chunk size is based upon a trace-driven analysis of two opposing concerns. If the chunk size is too large, internal fragmentation becomes significant. Whole file caching will waste bandwidth when partially written files are transferred to servers. Further, each demand miss will waste bandwidth as the client fetches data that it never uses. If the chunk size is too small, we will generate too many cache misses because we fail to adequately exploit spatial locality. Since each cache miss slows performance, the overall impact can be significant.

To determine appropriate values for the chunk size, we captured a trace of the disk blocks fetched by VMware during the execution of an industry-standard PC benchmark called *Sysmark* [5]. We adapted a cache simulation package, *Dinero IV* [9], to calculate the miss ratio and bandwidth consumed during workload execution for various chunk sizes. Fig. 3 presents the results of these simulations. As the figure shows, a chunk size of 128 KB strikes a reasonable balance between bandwidth wastage and miss ratio.

5.4. Client architecture

Fig. 4 shows the client architecture that we have developed to interface VMware to Coda. A loadable kernel module called *Fauxide* serves as the device driver for a pseudo-device named `/dev/hdk` in Linux. A VM is configured to use this pseudo-device as its sole virtual disk in “raw” mode. Disk I/O requests to `/dev/hdk` are redirected by Fauxide to a user-level process called *Vulpes*. It is Vulpes that implements VM state transfer policy, as well as the mapping of VM state to files in Coda. Vulpes also controls the hoarding of those files. Since Vulpes is outside the kernel and fully under our control, it is easy to experiment with a wide range of state transfer policies. Fig. 5 illustrates the logical layering of an ISR client.

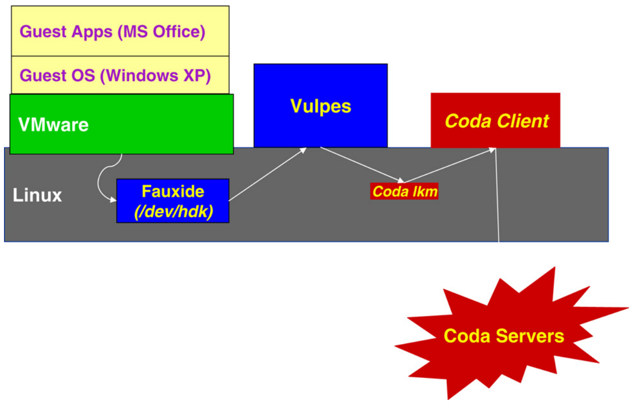


Fig. 4. ISR client architecture.

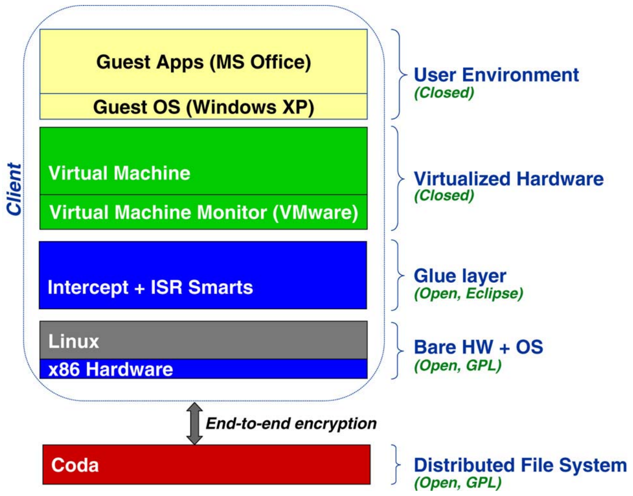


Fig. 5. ISR client layering.

6. ISR performance

6.1. Metrics

From a user's perspective, the key performance metrics of ISR can be characterized by two questions:

- *How soon after resume can I begin useful work?*
- *After I resume, how much is my work slowed down?*

We refer to the first metric as *resume latency* and the second as *slowdown*. Ideally one would like zero resume latency and zero slowdown. In practice, there are trade-offs between the two. Shrinking resume latency may increase average slowdown and vice versa. An important goal of our performance evaluation was to quantify these trade-offs for typical ISR scenarios. A related goal was to determine whether our prototype and today's networking infrastructure are adequate for ISR deployment. As discussed below, our results confirm that ISR is already usable today for some common usage scenarios. At the same time, the results also reveal certain limitations of our prototype.

6.2. Benchmark

ISR is intended for interactive workloads typical of laptop environments. Applications from the Microsoft Office suite dominate such workloads. We initially considered an industry-standard PC benchmark called *Sysmark* [5]. Unfortunately, this proprietary benchmark has very restrictive limitations on publication of results. We have therefore developed our own benchmark called the *Common Desktop Application (CDA)* that models an interactive Windows user.

CDA uses Visual Basic scripting to drive Microsoft Office applications such as Word, Excel, PowerPoint, Access, and Internet Explorer. CDA operates on each of these applications independently. The operations mimic typical actions that might be performed by an office worker. In totality, CDA consists of a total of 113 independently timed operations such as `find-and-replace`, `open-document`, and `worksheet-sort`. Actions such as keystrokes, object selection, or mouse-clicks are not timed. CDA pauses between operations to emulate think time. The pause is typically 10 s, but is 1 sec for a few quick-response operations such as `find-and-replace`.

The input data sets used by CDA are of moderate size. For example, Excel operates on two spreadsheets. One is 4783 rows by 6 columns, and occupies 570 KB of disk space; the other is 4095 rows by 100 columns, and occupies 1.7 MB. Word is used on a short novel of 760 KB (initially without images); PowerPoint operates on a 20-slide, 116 KB presentation; the Access database is approximately 2 MB; and the Internet Explorer data set is 446 KB of html pages.

Approximately 50% of the benchmark operations generate disk traffic at the Vulpes interface, with a roughly exponential distribution of data volume. The most disk-intensive operation is launching PowerPoint, which transfers 5.6 MB. Fig. 6 shows the data access characteristics of this benchmark, as seen by Vulpes. The curves labelled "Reads" and "Writes" show the cumulative volume of read and write traffic seen by Vulpes over the life of the benchmark. The curves labelled "Unique clean" and "Unique dirty" show the cumulative amount of distinct data read or written during the benchmark. The difference between "Read" and "Unique clean" indicates the extent of temporal read locality as seen by Vulpes. Similarly, the difference between "Write" and "Unique dirty" shows the temporal write locality seen by Vulpes. Note that I/O buffer caches inside the guest and host OSes absorb a significant amount of read and write locality, thus lowering the locality seen by Vulpes.

6.3. Experimental setup

Our experimental infrastructure consisted of 2.0 GHz Pentium 4 clients connected to a 1.2 GHz Pentium III Xeon server through 100 Mb/s Ethernet. All machines had 1 GB of

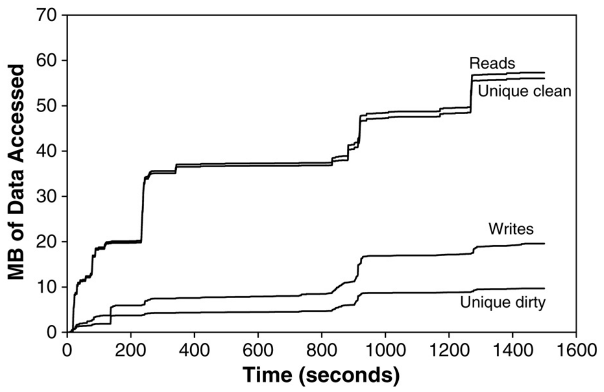


Fig. 6. Data accesses seen by Vulpes. This figure shows the cumulative volume of data read and written at the Vulpes interface by the CDA benchmark. The benchmark time in this figure is longer than the 1071 s shown in Table 2 because of slowdown caused by the logging in Vulpes that provided the data for this figure.

Table 2
Benchmark time: no ISR support

With think time (s)	No think time (s)
1071 (10)	93 (6)

The first column shows the total running time of the benchmark. Each data point is the mean of three trials, with standard deviation in parentheses. The second column is obtained by summing the execution times of the individual operations that make up the benchmark.

RAM, and ran RedHat 7.3 Linux. Clients used VMware Workstation 3.1 and had an 8 GB Coda file cache. The VM was configured to have 256 MB of RAM and 4 GB of disk, and ran Windows XP as the guest OS.

We used the NISTNet network emulator [7] to control available bandwidth. We measured ISR performance at four different bandwidths: 100 Mb/s, 10 Mb/s, 1 Mb/s and 100 Kb/s. The first two correspond to LAN speeds, and NISTNet was not configured to add any latency at these speeds. At 1 Mb/s, we configured NISTNet to add 10 ms latency, and at 100 Kb/s it added 100 ms.

Table 2 shows the benchmark time on our experimental setup without ISR support. In other words, the files used by VMware are on the local file system rather than on `/dev/hdk`. The effects of Fauxide, Vulpes and Coda are thus completely eliminated, but the effect of VMware is included. The total running time of 1071 s is a lower bound on the benchmark time achievable by any state transfer policy in our experiments.

6.4. VM state transfer policies

The copyout/copyin mechanism of ISR-1 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. Three steps can be taken to shorten resume latency:

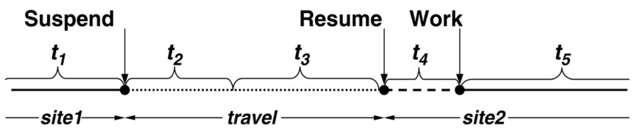


Fig. 7. Conceptual ISR timeline.

- Propagate dirty state to servers before suspend.
- Warm the file cache at the resume site.
- Allow resume to occur before full state has arrived.

These steps are not mutually exclusive, and can be combined in many different ways to generate a wide range of policies. We explore some of these policies below.

The conceptual timeline shown in Fig. 7 provides a uniform framework for our discussion of policies. The figure depicts a user initially working for duration *t1* at Internet location *site1*. He then suspends, and travels to Internet location *site2*. In some situations, the identity of *site2* is known (or can be guessed) a priori. In other situations, it becomes apparent only when the user unexpectedly shows up and initiates resume. The transfer of dirty state from *site1* to file servers continues after suspend for duration *t2*. There is then a period *t3* available for proactive file cache warming at *site2*, if known. By the end of *t3*, the user has arrived at *site2* and initiates resume. He experiences resume latency *t4* before he is able to begin work again. He continues working at *site2* for duration *t5* until he suspends again, and the above cycle repeats itself. With some state transfer policies, the user may experience slowdown during the early part of *t5* because some operations block while waiting for missing state to be transferred.

Note that Fig. 7 is only a canonical representation of the ISR timeline. Many special or degenerate cases are possible. For example, *t2* may not end before resume if travel duration is very short. In that case, the residue of *t2* may add to *t4* in contributing to resume latency. On the other hand, a clever state transfer policy may allow this residue to overlap *t4*. In other words, propagation of dirty state from the suspend site to file servers could overlap state propagation from those servers to the resume site. Another special case is when *t5* is very brief. With such a short dwell time, full VM state may never accumulate at *site2* — only enough to allow the user a few moments of work past the suspend point at the end of *t1*. While many such special cases are conceivable, the timeline in Fig. 7 is likely to cover a wide range of common real-world scenarios.

6.5. Baseline policy

6.5.1. Description

The baseline policy is a worst-case strawman that we do not expect to be used in practice. After suspend, all dirty state is transferred to the server during *t2*. The period *t3* is empty. Following resume, the entire VM state is transferred to the resume site during *t4* and pinned in the client cache. Note here that no state transfer occurs during either execution period *t1* or *t5*. This optimizes for execution speed at the cost of suspend and resume latency.

Table 3
Resume latency: baseline

Bandwidth	Resume latency
100 Mb/s	2504 (18) s
10 Mb/s	5158 (34) s
1 Mb/s	>9 h
100 Kb/s	>90 h

This table shows resume latency for the baseline policy at different bandwidths. The results for 100 Mb/s and 10 Mb/s are actual experimental measurements: in each case, the mean of three trials is reported along with the standard deviation in parentheses. The results for 1 Mb/s and 100 Kb/s are estimated, and were obtained by dividing total VM state size by nominal bandwidth.

This policy is applicable when *site2* cannot be predicted and when the site may become disconnected after a successful resume. For this policy, we expect the resume latency to be the longest, as it transfers the entire state in *t4*, but expect slowdown to be the shortest, because all VM state is available locally before resume.

Our implementation of the baseline policy has the Coda client at each ISR site operating in write-disconnected mode. In this mode, dirty cache state is trickled back to servers in the background. This can cause slight performance degradation of foreground activity and thus contributes to slowdown. However, it also improves suspend latency by reducing the volume of dirty state to be propagated at suspend.

6.5.2. Results

We expect the baseline policy to exhibit poor resume latency because all state transfer takes place during the resume step. We also expect network bandwidth to be a dominant factor. Table 3 confirms this intuition.

At 100 Mb/s, the resume latency is about 40 min. When bandwidth drops to 10 Mb/s, resume latency roughly doubles. The reason it does not increase by a factor of 10 (to match the drop in bandwidth) is that the data transfer rate at 100 Mb/s is limited by Coda rather than by the network. Only below 10 Mb/s does the network become the limiting factor. The results in Table 3 show that the baseline policy is only viable at LAN speeds, and even then only for a limited number of usage scenarios.

In contrast to resume latency, we expect slowdown to be negligible with the baseline policy because no ISR network accesses should be necessary once execution resumes. Tables 2 and 4 confirm that slowdown is negligible at 100 Mb/s. The total running time of the benchmark increases from 1071 to 1105 s. This translates to a slowdown of about 3.2%, where slowdown is defined as $(T_{\text{bw}} - T_{\text{noisr}})/T_{\text{noisr}}$, with T_{bw} being the benchmark running time at the given bandwidth and T_{noisr} its running time in VMware without ISR. This slowdown can be viewed as the intrinsic overhead of ISR. It is composed of two parts: the overhead due to indirection through Fauxide and Vulpes, and the greater cost of file operations such as `open` and `close` in Coda relative to the local file system. To focus on operation latency one can exclude think time from total benchmark running time. The second column of Table 2 and the third column of Table 4 show that total operation latency grows from 93 to 113 s, an increase of about 22.5%.

Table 4
Benchmark time: baseline

Bandwidth	With think time (s)	No think time (s)
100 Mb/s	1105 (9)	113 (2)
10 Mb/s	1170 (47)	168 (47)
1 Mb/s	1272 (65)	204 (56)
100 Kb/s	1409 (38)	251 (24)

The second column shows the total running time of the benchmark at various bandwidths for the baseline policy. Each data point is the mean of three trials, with standard deviation in parentheses. The third column is obtained by summing the execution times of the individual operations that make up the benchmark.

As bandwidth drops below 100 Mb/s, Tables 2 and 4 show that slowdown grows slightly. It is about 9.2% at 10 Mb/s, 18.8% at 1 Mb/s, and 31.6% at 100 Kb/s. This slight dependence on bandwidth is due to Coda background activity such as trickle reintegration.

6.6. *Fully proactive policy*

6.6.1. *Description*

If we can predict *site2*, we can define a much more aggressive state transfer policy. At *site2*, this policy shifts the entire state transfer time from *t4* to earlier periods in the ISR timeline. During *t3* (or earlier, for any state already available at the servers) *site2* transfers all updated state to its local cache. Note that this includes both VM disk and memory state. At resume, all that remains is to launch the VM.

We expect this policy to be most effective when a user is working among a small set of sites, such as when alternating between home and work. If two sites start in a synchronized virtual state, then the state to be transferred during travel is limited to the unique state modified during *t1*. Like the baseline policy, after a successful resume the fully proactive policy permits operation at *site2* while disconnected.

For this policy, we expect resume latency to be shortest, because all state transfer has been moved to time *t3*. Slowdown will also be minimal because all state is available before resume. As in the baseline case, trickle reintegration contributes slightly to this slowdown.

6.6.2. *Results*

With a fully proactive policy one expects resume latency to be bandwidth independent and very small because all necessary files are already cached. The only delays incurred are those of Vulpes uncompressing the file containing the suspended VM memory image, and of VMware launching a VM with this image. Table 5 confirms this intuition. Resume latency is only 10–11 s at all bandwidths.

Post-resume ISR execution under a fully proactive policy is indistinguishable from the baseline policy. The user experience, including slowdown, is identical. The results of Table 4 therefore apply to both policies. Clearly, the fully proactive policy is very attractive from the viewpoint of resume latency and slowdown.

What is the minimum travel time for a fully proactive policy to be feasible? This duration corresponds to $t2 + t3$ in Fig. 7. There are two extreme cases to consider. In the best case, the resume site is known well in advance and its cache has been closely

Table 5
Resume latency: fully proactive

Bandwidth	Resume latency (s)
100 Mb/s	10.3 (0.1)
10 Mb/s	10.2 (0.0)
1 Mb/s	10.2 (0.0)
100 Kb/s	11.4 (0.0)

This table shows the resume latency for the fully proactive policy at different bandwidths. Each data point is the mean of three trials; standard deviations are in parentheses.

tracking the cache state at the suspend site. All that needs to be transferred is the residual dirty state at suspend — the same state that is transferred to servers during *t2*. For our experimental configuration, we estimate this state to be about 47 MB at the mid-point of benchmark execution. Using observed throughput values in our prototype, this translates to minimum best case travel time of 45 s with a 100 Mb/s network, and about 90 s with a 10 Mb/s network. Both of these are credible bandwidths and walking distances today between collaborating workers in a university campus, corporate campus or factory.

At lower bandwidths, we estimate the best case travel time to be at least 800 s (roughly 14 min) at 1 Mb/s, and 8000 s (roughly 2 h 15 min) at 100 Kb/s. The 14 min travel time is shorter than many commutes between home and work, and bandwidths close to 1 Mb/s are available to many homes today.

In the worst case, the resume site has a completely cold cache and is only identified at the moment of suspend. In that case, $t3$ must be long enough to transfer the entire state of the VM. From the baseline resume latencies in Table 3 and the value of $t2$ above, we estimate minimum travel time to be 2550 s (roughly 43 min) for a 100 Mb/s network, and 5250 s (88 min) for a 10 Mb/s network. These are plausible travel times from office or home to aircraft seat.

To summarize, there are some common usage scenarios today where a fully proactive strategy makes ISR viable. Over time, network infrastructure will improve, but travel times are unlikely to decrease. Hence, we expect ISR with a fully proactive policy to become viable for a broader range of scenarios in the future.

6.7. Pure demand-fetch policy

6.7.1. Description

Suppose a user arrives unexpectedly at an ISR site. If we wish to keep *t4* as short as possible, we can use a pure demand-fetch policy to amortize the cost of retrieving the VM disk state over *t5*. In this policy, only virtual memory state is retrieved during *t4*; the transfer of disk state in 128 KB chunks is deferred. As soon as the virtual memory state has arrived, the VM is launched. Then, during *t5*, disk accesses by the VM may result in cache misses that are serviced from the file server.

We expect the resume latency for this policy to be short, as only critical state is transferred during *t4*. We also expect substantial slowdown because of cache misses during *t5*.

Table 6
Resume latency: pure demand-fetch

Bandwidth	Resume latency (s)
100 Mb/s	14 (0.5)
10 Mb/s	39 (0.4)
1 Mb/s	317 (0.3)
100 Kb/s	4301 (0.6)

This table shows the resume latency for the pure demand-fetch policy at different bandwidths. Each data point is the mean of three trials; standard deviations are in parentheses.

6.7.2. Results

In our prototype, the state transferred at resume for pure demand-fetch is the compressed memory image of the VM at suspend (roughly 41 MB). The transfer time for this file is a lower bound on resume latency for this policy at any bandwidth. As Table 6 shows, resume latency rises from well under a minute at LAN speeds of 100 Mb/s and 10 Mb/s to well over an hour at 100 Kb/s.

We expect the slowdown for a pure demand-fetch policy to be very sensitive to workload. If the workload has high temporal locality of virtual memory and file accesses, Vulpes will access relatively few chunks. The first access to each file results in a cache miss, and will therefore contribute to slowdown. High spatial locality in the workload will also result in relatively few chunks being accessed.

Table 7 shows the observed running time of the benchmark under a pure demand-fetch policy. To estimate slowdown, these numbers should be compared to the non-ISR results of Table 2. The total benchmark time rises from 1071 s without ISR to 1160 s at 100 Mb/s. This represents a slowdown of about 8.3%. As bandwidth drops, the slowdown rises to 30.1% at 10 Mb/s, 340.9% at 1 Mb/s, and well over an order of magnitude at 100 Kb/s. The slowdowns below 100 Mb/s will undoubtedly be noticeable to a user. But this must be balanced against the potential improvement in user productivity from being able to resume work anywhere, even from unexpected locations.

Table 8 shows the distribution of slowdown across benchmark operations. At 100 Mb/s, 43% of the operations are slowed down less than 10%; 27% are slowed down between 10% and 50%; and so on. As bandwidth drops, a greater fraction of the operations are slowed by higher amounts.

6.8. Impact of storage efficiency

Since Coda is an experimental user-level system, it is less efficient than a well-tuned, in-kernel NFS implementation. As mentioned in Section 6.5.2, the effect is noticeable only above 10 Mb/s. To get an idea of the potential for improvement, we measured the pure demand-fetch policy at 100 Mb/s using NFS rather than Coda. Table 9 presents the results. Comparing this with Tables 6 and 8, we see that there is significant improvement in both resume latency and slowdown. However, at the more challenging bandwidths for ISR (below 10 Mb/s), merely improving file system efficiency does not help much. Other mechanisms, such as proactivity, are needed in those situations.

Table 7
Benchmark time: pure demand-fetch

Bandwidth	With think time (s)	No think time (s)
100 Mb/s	1160 (5.6)	173 (9.5)
10 Mb/s	1393 (20.1)	370 (14)
1 Mb/s	4722 (69)	2688 (39)
100 Kb/s	42 600 (918)	30 531 (1490)

The second column shows the total running time of the benchmark at various bandwidths for the pure demand-fetch policy. The entries in the third column are obtained by summing the execution times of the individual operations that make up the benchmark. Each data point is the mean of three trials, with standard deviation in parentheses.

Table 8
Slowdown summary for CDA operations

Slowdown (%)	100 Mb/s (%)	10 Mb/s (%)	1 Mb/s (%)	100 Kb/s (%)
<10	43	37	27	28
10–50	27	19	19	13
50–100	11	8	5	5
100–400	12	22	8	3
400–1000	6	5	11	2
>1000	1	9	30	49

This table summarizes the distribution of slowdown for benchmark operations at different bandwidths. For each operation, slowdown is defined as $(T_{\text{bw}} - T_{\text{noisr}})/T_{\text{noisr}}$, where T_{bw} is the operation latency at the given bandwidth and T_{noisr} is its latency when run in VMware without ISR.

Table 9
Impact of file system efficiency

(a) Resume Latency: Demand-Fetch with NFS

Bandwidth	Resume latency (s)
100 Mb/s	6 (0.03)

(b) Slowdown Summary: Demand-Fetch with NFS

Slowdown (%)	100Mb/s (%)
<10	50
10–50	33
50–100	12
100–400	5
400–1000	0
>1000	0

Part (a) depicts the resume time for the benchmark under the demand-fetch policy. Part (b) depicts the distribution of operation slowdown values, also under the demand-fetch policy.

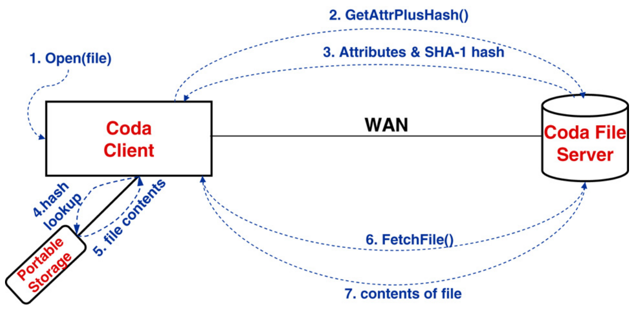


Fig. 8. Lookaside servicing of a cache miss.

7. Augmenting ISR with portable storage

7.1. Background

Our discussion of ISR until this point has emphasised its “hands-free” or “carry-nothing” aspect. In practice, users may be willing to carry something small and unobtrusive if that would enhance their ISR usage experience. Today, USB and Firewire storage devices in the form of storage keychains or microdrives are widely available. By serving as a local source of critical data, such a device could improve ISR performance at sites with poor network connectivity.

There are at least two risks with using a portable storage device to hold VM state. First, it could be out of date with respect to current VM state in the distributed file system. This might happen, for example, if the user forgets to update the device at suspend or if he absent-mindedly picks up the wrong device for travel. Second, the device may be lost, broken or stolen while travelling. These considerations suggest that ISR should treat a portable storage device only as a performance enhancement, not a substitute, for the underlying distributed file system. They also suggest that data on portable devices be self-validating: a stale device may not improve performance, but should do no harm.

7.2. Lookaside cache miss handling

We have extended Coda with a simple yet versatile mechanism called *lookaside caching* to take advantage of portable storage as a performance enhancement. Lookaside caching consists of three parts: a small change to the client-server protocol; a quick index check (the “lookaside”) in the code path for handling a cache miss; and a tool for generating lookaside indexes. Dynamic inclusion or exclusion of indexes is done through user commands.

Fig. 8 illustrates the steps involved in handling a cache miss through lookaside. In the modified client-server protocol, access to a non-cached file begins with an RPC to obtain file attributes such as file size and modification time. This RPC now returns the SHA-1 hash value [21] of the file. The change adds only 20 bytes to the size of the original RPC reply, which is acceptable even on slow networks. Coda’s callback mechanism ensures that cached hash information tracks server updates.

The lookaside occurs between the RPCs to fetch file status and file contents. Since the client possesses the hash of the file at this point, it can cheaply consult one or more local lookaside indexes to see if a local file with identical SHA-1 value exists. Trusting in the collision resistance of SHA-1, a copy of the local file can then be a substitute for the RPC to fetch file contents. To detect version skew between the local file and its index, the SHA-1 hash of the local file is re-computed. In case of a mismatch, the local file substitution is suppressed and the cache miss is serviced by contacting the file server. Coda's consistency model is not compromised, although some small amount of work is wasted on the lookaside path.

The index tool walks the file tree rooted at a specified pathname. It computes the SHA-1 hash of each file and enters the filename–hash pair into the index file. The only requirement on the file tree is that it be part of the filename space on which the tool is run: it can be local, on a mounted storage device, or even on a nearby NFS or Samba server. For a removable medium, the index is located on the medium itself. This yields a self-describing portable storage device that can be used at any ISR site.

7.3. Demand-fetch with lookaside policy

7.3.1. Description

The performance of the pure demand-fetch policy, discussed in Section 6.7, can be improved by using lookaside caching. If a user is willing to wait briefly at suspend, the virtual memory image and a lookaside index for it can be written to his portable storage device. He can then remove the device and carry it with him. At the resume site, lookaside caching can use the device to reduce *t4*.

Another use of lookaside caching exploits the fact that many parts of VM state rarely change. For example, the parts of VM disk state corresponding to executable code for applications and dynamically linked libraries do not change after installation. An organization that deploys ISR could make this VM state widely available on read-only media such as CD-ROMs or DVDs. Lookaside caching from such media can reduce the cost of cache misses during *t5*, and hence improve the slowdown metric. Note that management complexity is not increased because misplaced or missing media do not hurt correctness. Since multiple lookaside devices can be in use simultaneously, these distinct uses of lookaside caching can be easily combined.

7.3.2. Results

Our experiments show that demand fetch performance can be substantially improved through lookaside caching. Table 10 presents our results for the case where a portable storage device is updated with the compressed virtual memory image at suspend, and used as a lookaside device at resume. Comparing Tables 6 and 10 we see that the improvement is noticeable below 100 Mb/s, and is dramatic at 100 Kb/s. A resume time of just 12 s rather than 317 s (at 1 Mb/s) or 4301 s (at 100 Kb/s) can make a world of a difference to a user with a few minutes of time in a coffee shop or a waiting room. Even at 10 Mb/s, resume latency is a factor of 3 faster (12 s rather than 39 s).

Notice that lookaside caching helps exactly when a fully proactive policy is infeasible because the resume site is not predictable. Further, the use of lookaside caching ensures

Table 10
Resume latency: USB lookaside

Bandwidth	Resume latency (s)
100 Mb/s	13 (2.2)
10 Mb/s	12 (0.5)
1 Mb/s	12 (0.3)
100 Kb/s	12 (0.1)

This table shows the resume latency for a demand-fetch policy where a portable storage device containing the virtual memory image at suspend is available to lookaside caching at the resume site. Each data point is the mean of three trials; standard deviations are in parentheses. The device used for these experiments was a USB portable disk.

Table 11
Benchmark time: DVD lookaside

Bandwidth	With think time (s)	No think time (s)
100 Mb/s	1141 (35.7)	161 (27.8)
10 Mb/s	1186 (17.3)	212 (12.3)
1 Mb/s	2128 (33.6)	1032 (31.0)
100 Kb/s	13967 (131.4)	9530 (140.9)

The second column shows the total running time of the benchmark for a demand-fetch policy where lookaside caching has a DVD available for lookaside. As explained in Section 7.3.2, the DVD contains VM state prior to user customization. The entries in the third column are obtained by summing the execution times of the individual operations that make up the benchmark. Each data point is the mean of three trials, with standard deviation in parentheses.

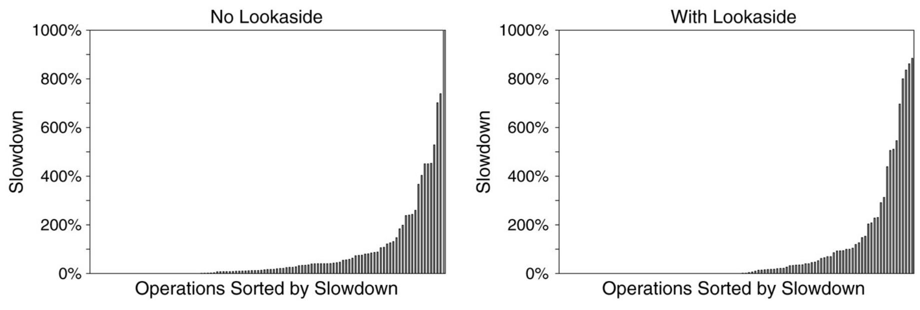
that human errors such as using the wrong device are detected and gracefully handled by the system. Resume latency will then match Table 6, but the user will resume in the correct state.

To explore the impact of lookaside caching on slowdown, we constructed a DVD with the VM state captured after installation of Windows XP and the Microsoft Office suite, but before any user-specific or benchmark-specific customizations. We used this DVD as a lookaside device during the benchmark. Table 11 presents our results.

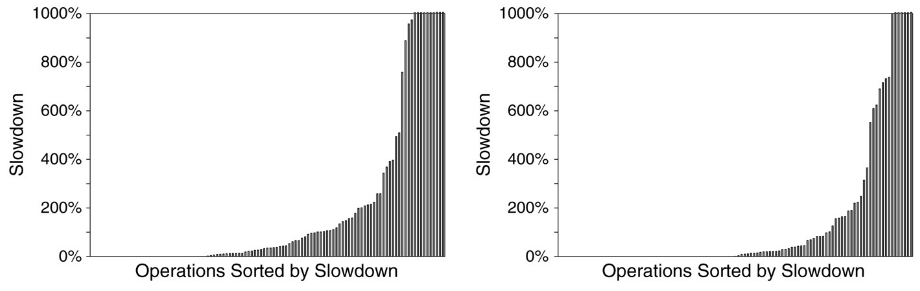
Comparing Tables 7 and 11, we see that benchmark time is reduced at all bandwidths. The reduction is most noticeable at lower bandwidths: roughly a factor of 2 at 1 Mb/s (2128 s rather than 4722 s), and a factor of 3 at 100 Kb/s (13967 s rather than 42600 s). The user impact of lookaside caching can be visualized by examining the distribution of slowdown for individual benchmark operations. Comparing the two columns of Fig. 9, one sees that fewer operations suffer large slowdown with lookaside caching. This is especially noticeable at low bandwidths.

7.4. *Off-machine lookaside*

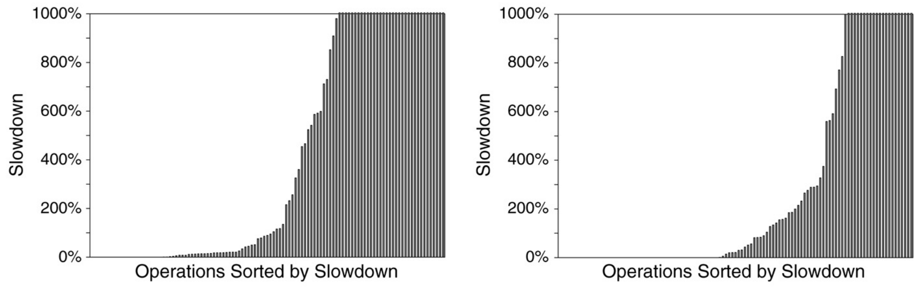
We have recently extended lookaside caching to use off-machine *content-addressable storage (CAS)*. The growing popularity of planetary-scale services such as PlanetLab [22], distributed hash-tables such as Pastry [26], Chord [35], Tapestry [42] and CAN [25], and logistical storage such as the Internet Backplane Protocol [3], all suggest that CAS will



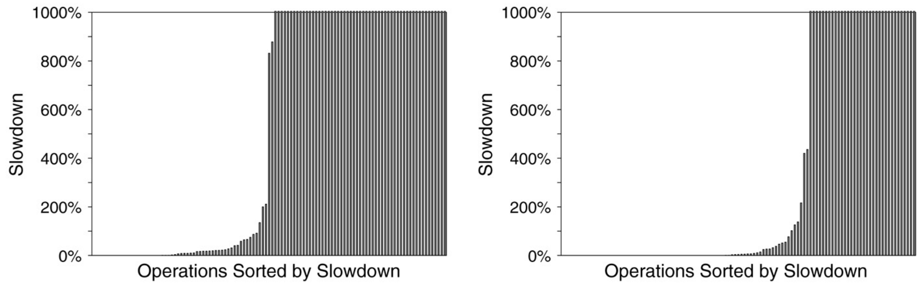
(a) 100 Mb/s.



(b) 10 Mb/s.



(c) 1 Mb/s.



(d) 100 Kb/s.

Fig. 9. Impact of lookaside caching on slowdown of CDA benchmark operations. These figures compare the distribution of slowdown for the operations of the CDA benchmark without lookaside caching to their slowdowns with lookaside caching to a DVD.

Table 12
Benchmark time: jukebox lookaside

Bandwidth	With think time (s)	No think time (s)
100 Mb/s	1068 (5.7)	103 (3.9)
10 Mb/s	1131 (6.8)	163 (2.9)
1 Mb/s	2256 (24.7)	899 (26.4)
100 Kb/s	13514 (30.7)	8567 (463.9)

The second column shows the total running time of the benchmark for a demand-fetch policy where lookaside caching uses a jukebox for lookaside. The jukebox contains the same state as the DVD used for the results of Table 11. The entries in the third column are obtained by summing the execution times of the individual operations in the benchmark. Each data point is the mean of three trials, with standard deviation in parentheses.

become a widely supported service in the future [4]. For brevity, we refer to any network service that exports a CAS interface as a *jukebox*. When presented with a hash, the jukebox returns content matching that hash, or an error code indicating that it does not possess requested content.

Lookaside caching enables ISR to make opportunistic use of jukeboxes. The term "opportunistic" is important here: we treat jukeboxes purely as a performance enhancement; we do not depend on them for consistency. A collection of ISR sites with mutual trust (typically at one location) can export each other's Coda file caches as jukeboxes. No protocol is needed to maintain mutual cache consistency. Divergent caches may, at worst, reduce the performance improvement from lookaside caching.

For each type of jukebox, we implement a *lookaside proxy* that encapsulates the protocol used by that jukebox. Jukeboxes may be added or removed at runtime, and more than one jukebox can be in use at any given time. Table 12 shows the performance benefit of using a LAN-attached jukebox with same contents as the DVD of Section 7.3.2. We see similar improvement in the DVD and jukebox cases.

8. Assumptions and limitations of ISR

Although ISR is viable on current technology, its full potential lies in the future. It is useful to understand which aspects of ISR are limitations of current technology or implementation status, and which are fundamental. Toward this end, we examine its underlying assumptions and limitations in this section. We use the hypothetical scenario of Section 4.2 as a working example throughout this section. We also refer back to the alternative approaches to seamless mobility discussed in Section 3.

8.1. Network bandwidth

Perhaps the most fundamental assumption of ISR is the existence of high bandwidth for VM state transfer. Logically, the entire state of a VM (typically many tens of GB, possibly hundreds of GB) has to be transferred on suspend and resume. This is the price paid for the simplification of state management provided by ISR — one's *entire* personal computer is delivered on demand. This deliberate profligacy represents an important tradeoff. We believe that it is easier and simpler to blindly ship a lot of bits than to sustain the management attention and system administration discipline needed to widely deploy a more frugal abstraction such as process migration.

Logical state transfer does not, however, have to always mean physical state transfer. Clever policies can give the illusion of full VM state transfer while actually transferring much less data. These policies do rely on some assumptions. The fully proactive approach assumes that it is possible to predict a user's resume site with confidence, and that it is possible to rely on a mostly warm persistent cache at that location. Simple predictive ability (based on static locations such as work and home) exists today. In future, we will need to extend this using more sophisticated prediction schemes that rely, for example, on mobility history or integration with calendaring and meeting scheduling software. The pure demand-fetch policy is a "pay as you go" approach to bandwidth use, but it has the weakness that it is depends on network connectivity to service cache misses. Network resilience, which is one of the strong points of ISR relative to thin clients, suffers with a demand fetch policy.

The use of lookaside caching on portable storage represents a different approach to reducing bandwidth usage. Here, "sneakernet" is being combined with the real network to give a composite that has the best of both worlds: device bandwidth and consistency of a real network. If one is willing to compromise consistency, it is possible to imagine an approach where VM state on a portable device is used without contacting servers to verify that it is up to date. Such an approach relies entirely on "sneakernet" and therefore requires zero network bandwidth. Caceres et al. [6] describe a system based on this approach.

It is possible to shrink the size of a VM by placing all user data directly in a distributed file system. The VM only encapsulates the OS and application state; the user data is delivered separately, using the same underlying caching mechanism. This is a usage model that combines traditional PC practice in the Windows world, with the practice found in many Unix environments that rely on distributed file systems. Explicit hoarding of user data will then be necessary to ensure disconnected operation. This approach can be advantageous when the amount of data in user files is much greater than OS and application state. It gives the user direct control at fine granularity over a large part of his total state, while treating OS and application state as an opaque entity.

8.2. Network dependence

Distinct from the volume of data transferred is the issue of network availability. Once data is fully hoarded, ISR does not require the network to be available. The disconnected operation capability of the underlying storage system (Coda in ISR-2) provides the illusion of connectivity for ISR. Optimistic replica control in Coda allows cached state to be used even when disconnected. Updates are buffered by the client, and eventually reintegrated when network connectivity is restored. There is no danger of conflicting updates on reintegration because ISR uses pessimistic concurrency control at coarse granularity — resume occurs only after a lock on the entire VM state is acquired from ISR servers.

Thus, ISR is *asynchronous* in its network dependence. Connectivity is needed while hoarding data, and during eventual reintegration. For extended periods between these two events (possibly lasting many hours), total disconnection is acceptable and has absolutely no performance impact. If the ISR client machine is a laptop, a user can be as mobile and productive with it during the period of disconnection as he is with a laptop today. Of course, direct use of the network by the user (such as Web browsing) cannot be supported while disconnected. But all other work such as editing, authoring, and so on, can be performed just as if the network were up. It is this capability that Shanta makes use of on the plane in the scenario of Section 4.2.

The asynchronous network dependence of ISR distinguishes it from the *synchronous* network dependence of thin clients. By definition, disconnected operation is impossible with thin clients. The quality of the network has to be sufficient at all times for crisp interactive response. Note that it is the worst case, not the average case, that determines whether a thin-client approach will be satisfactory. An organization that adopts thin-client computing must also invest in system management resources to ensure adequate network quality at all times for its most demanding interactive tasks. Adding bandwidth is relatively easy, but reducing latency is much harder. In addition to physical layer transmission delays and end-to-end software path lengths, technologies such as firewalls, overlay networks, and lossy wireless networks add latency and other hurdles. Even when using a pure demand-fetch policy, ISR performance is not sensitive to network latency even though it is sensitive to network bandwidth,

Interest in thin clients is very high today because of frustration with the high total cost of ownership of PCs. Unfortunately, dependence on thin clients may hurt the important goal of crisp interactive response. There is extensive evidence from the HCI community that interactive response times over 150 ms are noticeable and begin to annoy a user as they approach 1 s. To attain the goal of seamless mobility with thin clients, one needs very tight control on end-to-end network latency. This is hard to achieve at large scale. Like a thin client, an ISR client is stateless from the viewpoint of long-term user state. ISR can be viewed as a solution that trades off startup delay for crisp interaction: once execution begins, all interaction is local.

8.3. Ubiquitous virtualized hardware

A key requirement for ISR is that every client must have the same hardware architecture. While cross-architecture emulation is possible, the performance degradation is usually too high to be acceptable. We therefore see architectural uniformity of hardware, at least at the instruction set level, as a long-term assumption of ISR.

Another key requirement is the availability of ISR support on all potential clients. At present, this consists of two parts: the host OS with supporting software such as Coda, and the VMM. In the near term, we see these as components that have to be maintained by system administrators. In the long term, stripped-down versions of these components may be integrated with the BIOS on a client. In that case, there would be no software to maintain on clients. ISR clients would then be as stateless as thin clients are today.

Virtualization can mask many hardware differences such as CPU speed and memory size, and can even emulate missing features such as MMX instructions on the Intel IA-32 architecture. However, an ISR user may sometimes wish to be aware of such differences. For example, a PowerPoint animation created on a machine with a fast CPU and a high-resolution display may not work well on poorer hardware. For ISR to succeed in these situations, it is necessary to verify adequate hardware compatibility before the resume step is attempted. One way to achieve this would be for the VM state to be tagged with an encoding of hardware requirements, and for these requirements to be checked in the lock acquisition phase of resume. The tag information could also be used when selecting hardware to assign to a user, as at hotel check-in in the scenario of Section 4.2.

9. Conclusion

In summary, our results confirm that ISR performs well enough today for serious use in some scenarios. For example, a fully proactive strategy is well matched to a situation where a user has a home office and multiple corporate offices. Such a user would greatly value the simplicity of a single personalized computing environment that can be suspended and resumed at will among these locations. Using portable storage with lookaside caching, this capability could be extended to poorly connected work sites. On a corporate or university campus with 100 Mb/s or better connectivity, a pure demand-fetch policy would allow users who are away from their offices to productively use any nearby machine. This may lead to enhanced levels of collaboration and spontaneous deep interactions between users.

Our work so far has treated the VMM and guest OS as black boxes. Many optimizations are conceivable if we can modify these layers. While this is a promising future research direction, it may be a difficult path because it requires the cooperation of software vendors and may compromise the freely distributable open-source character of ISR prototypes. Using an open-source VMM such as Xen [2] may avoid these drawbacks.

Since ISR is a new approach to personal computing, its widespread use may lead to client, server and network workloads that are very different from those studied in the past. There are also important usability questions that are difficult to answer with confidence in the absence of hands-on usage experience. Hence, an essential component of our research plan is the creation and maintenance of an ISR pilot deployment at Carnegie Mellon that is in daily use by a small user community. Empirical data and usage-based insights from this test bed will guide and prioritize our research efforts. We may also extend this pilot deployment to other user communities to obtain broader validation of ISR.

Of course, these are only baby steps toward the kind of futuristic ISR scenario described in Section 4.2. Enabling such scenarios will require major advances in ISR and security technologies, broader deployment of high-bandwidth network infrastructure, new business models, and societal acceptance of this new model of computing. The reward for this effort will be a transformation of information technology that brings it closer to the ideal expressed by Weiser [39]: “*The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it*”. When one’s personal computing environment is as ubiquitous as light at the flip of a switch or water from a faucet, it will indeed have been woven into the fabric of everyday life!

Acknowledgements

We would like to thank the organizers of the PerCom 2004 conference for inviting M. Satyanarayanan to present the keynote talk upon which this paper is based. Tom Bressoud and Shafeeq Sinnamohideen contributed to the performance measurements reported here, while Partho Nath contributed to the implementation of ISR. Discussions with Niraj Tolia and Dave Andersen were valuable in clarifying the limitations of thin clients, as discussed in Sections 3.1 and 8.2. Discussions with Dave Bantz and Ramon Caceres helped in articulating the properties of personal computing environments, as presented in Section 2.1.

This research was partially supported by the National Science Foundation (NSF) under grant numbers ANI-0081396 and CCR-0205266, and by the Intel Corporation. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF, the Intel Corporation or Carnegie Mellon University. All unidentified trademarks mentioned in the paper are properties of their respective owners.

References

[1] Y. Artsy, R. Finkel, Designing a process migration facility: the Charlotte experience, IEEE Computer 22 (9) (1989).
[2] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, A. Warfield, Xen and the art of virtualization, in: Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles, Bolton Landing, NY, USA, 2003.
[3] M. Beck, T. Moore, J.S. Plank, An end-to-end approach to globally scalable network storage, in: Proceedings of the ACM SIGCOMM Conference, Pittsburgh, PA, 2002.
[4] T. Bressoud, M. Kozuch, C. Helfrich, M. Satyanarayanan, OpenCAS: a flexible architecture for building and accessing content addressable storage, in: 2004 International Workshop on Scalable File Systems and Storage Technologies, September 2004, San Francisco, CA, 2004. http://ardra.hpcl.cis.uab.edu/sfast04/.
[5] Business Applications Performance Corporation, SYSmark 2002, March, 2002. http://www.bapco.com.
[6] R. Caceres, C. Carter, C. Narayanaswami, M. Raghunath, Reincarnating PCs with portable SoulPads, in: Proceedings of Mobisys 2005: the Third International Conference on Mobile Systems, Applications and Services, June 2005, Seattle, WA, 2005.
[7] M. Carson, Adaptation and Protocol Testing Through Network Emulation, September 1999. http://snad.ncsl.nist.gov/itg/nistnet/.
[8] F. Douglis, J.K. Ousterhout, Transparent process migration: design alternatives and the Sprite implementation, Software Practice and Experience 21 (8) (1991).
[9] J. Edler, M. Hill, Dinero IV Trace-Driven Uniprocessor Cache Simulator, http://www.cs.wisc.edu/~markhill/DineroIV/.
[10] D. Garlan, D.P. Siewiorek, A. Smailagic, P. Steenkiste, Project Aura: toward distraction-free pervasive computing, IEEE Pervasive Computing 1 (2) (2002).
[11] R.P. Goldberg, Survey of Virtual Machine Research, IEEE Computer 7 (6) (1974).
[12] R. Grimm, J. Davis, E. Lemar, A. Macbeth, S. Swanson, T. Anderson, B. Bershad, G. Borriello, S. Gribble, D. Wetherall, System support for pervasive applications, ACM Transactions on Computer Systems 22 (4) (2004).
[13] L. Grinzo, Getting virtual with VMware 2.0, Linux Magazine (2000).
[14] J. Howard, M. Kazar, S. Menees, D. Nichols, M. Satyanarayanan, R. Sidebotham, M. West, Scale and performance in a distributed file system, ACM Transactions on Computer Systems 6 (1) (1988).
[15] E. Jul, H. Levy, N. Hutchinson, A. Black, Fine-grained mobility in the emerald system, ACM Transactions on Computer Systems 6 (1) (1988).
[16] J.J. Kistler, M. Satyanarayanan, Disconnected operation in the Coda file system, ACM Transactions on Computer Systems 10 (1) (1992).
[17] M. Kozuch, C. Helfrich, D. O'Hallaron, M. Satyanarayanan, Enterprise client management with Internet Suspend/Resume, Intel Technical Journal 8 (4) (2004).
[18] M. Kozuch, M. Satyanarayanan, Internet Suspend/Resume, in: Proceedings of the Fourth IEEE Workshop on Mobile Computing Systems and Applications, June 2002, Callicoon, NY, 2002.
[19] M. Kozuch, M. Satyanarayanan, T. Bressoud, C. Helfrich, S. Sinnamohideen, Seamless mobile computing on fixed infrastructure, IEEE Computer 37 (7) (2004).

[20] A. Lai, J. Nieh, Limits of wide-area thin-client computing, in: Proceedings of the 2002 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, June 2002, Marina Del Rey, CA, 2002.
[21] National Institute of Standards and Technology, Secure Hash Standard (SHS), 1995.
[22] L. Peterson, T. Anderson, D. Culler, T. Roscoe, A blueprint for introducing disruptive technology into the internet, in: Proceedings of the First ACM Workshop on Hot Topics in Networks, Princeton, NJ, 2002.
[23] E. Pitt, K. McNiff, java.rmi: The Remote Method Invocation Guide, Addison-Wesley Professional, 2001.
[24] M.L. Powell, B.P. Miller, Process migration in DEMOS/MP, in: Proceedings of the 9th ACM Symposium on Operating Systems Principles, October 1983, Bretton Woods, NH, 1983.
[25] S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker, A scalable content-addressable network, in: Proceedings of the ACM SIGCOMM Conference, August 2001, San Diego, CA, 2001.
[26] A. Rowstron, P. Druschel, Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems, in: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Heidelberg, Germany, 2001.
[27] M. Satyanarayanan, Scalable, Secure and highly available distributed file access, IEEE Computer 23 (5) (1990).
[28] M. Satyanarayanan, The Evolution of Coda, ACM Transactions on Computer Systems 20 (2) (2002).
[29] P. Schwann, Lustre: building a file system for 1,000-node Clusters, in: Proceedings of the 2003 Linux Symposium, July 2003, Ottawa, Canada, 2003.
[30] H.A. Simon, Administrative Behavior, Macmillan, New York, NY, 1947.
[31] H.A. Simon, Designing organizations for an information-rich world, in: M. Greenberg (Ed.), Computers, Communications and the Public Interest, Johns Hopkins Press, Baltimore, MD, 1971.
[32] S.W. Smith, V. Austel, Trusting trusted hardware: toward a formal model for programmable secure coprocessors, in: Proceedings of the Third USENIX Workshop on Electronic Commerce, August 1998, Boston, MA, 1998.
[33] J.P. Sousa, D. Garlan, Aura: an architectural framework for user mobility in ubiquitous computing environments, in: Software Architecture: System Design, Development, and Maintenance (Proceedings of the 3rd Working IEEE/IFIP Conference on Software Architecture), Kluwer Academic Publishers, 2002.
[34] J.G. Steiner, C. Neuman, J.I. Schiller, Kerberos: an authentication service for open network systems, in: USENIX Conference Proceedings, Dallas, TX, Winter 1988.
[35] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, H. Balakrishnan, Chord: a scalable peer-to-peer lookup service for internet applications, in: Proceedings of the ACM SIGCOMM 2001, San Diego, CA, 2001.
[36] Trusted Computing Group (TCG), https://www.trustedcomputinggroup.org/, 2003.
[37] M. Theimer, K. Lantz, D. Cheriton, Preemptable remote execution facilities for the V-system, in: Proceedings of the 10th Symposium on Operating System Principles, Orcas Island, WA, December 1985.
[38] J.D. Tygar, B. Yee, Dyad: a system for using physically secure coprocessors, in: Proceedings of the Joint Harvard MIT Workshop on Technological Strategies for Protecting Intellectual Property in the Networked Multimedia Environment, April 1993.
[39] M. Weiser, The computer for the 21st century, Scientific American (1991).
[40] V.C. Zandy, B.P. Miller, M. Livny, Process hijacking, in: 8th International Symposium on High Performance Distributed Computing, Redondo Beach, CA, August 1999.
[41] E. Zayas, Attacking the process migration bottleneck, in: Proceedings of the 11th ACM Symposium on Operating System Principles, Austin, TX, November 1987.
[42] B.Y. Zhao, J. Kubatowicz, A. Joseph, Tapestry: an infrastructure for fault-tolerant wide-area location and routing, Tech. Rep. UCB/CSD-01-1141, University of California at Berkeley, April 2001.

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

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

Casey Helfrich is a research engineer at the Intel Research Lab in Pittsburgh. He received a Bachelor's degree in Physics from Carnegie Mellon University in 2001 and an additional B.S. degree in Computer Science from Carnegie Mellon University in 2002. He joined the Pittsburgh lab at its inception and helped design and build the IT infrastructure for Intel Research. Casey has spent the past two years working on the combination of virtualization and data distribution.

David O'Hallaron is an Associate Professor of Computer Science and Electrical and Computer Engineering at Carnegie Mellon University. His research interests include mobile computing, computational database systems, and scientific computing.