Smart Phones as Self-Cleaning Portable Caches for Infrastructure-Based Mobile Computing

Stephen Smaldone[†], Benjamin Gilbert, Matt Toups,
Liviu Iftode[†], Mahadev Satyanarayanan

July 2008
CMU-CS-08-140

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

[†]Dept. of Computer Science, Rutgers University, Piscataway, NJ

This research was supported by the National Science Foundation (NSF) under grant numbers CNS-0509004 and CNS-0520123. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF, Carnegie Mellon University, or Rutgers University.

Keywords: Smart phones, 3G phones, iPhone, PDA, Internet Suspend/Resume®, OpenISR®, mobile computing, pervasive computing, virtual machines, VMware, VirtualBox, Xen, KVM, Horatio, operating systems, distributed file systems, content addressable storage

Abstract

An increasingly ubiquitous computing infrastructure promises mobile users an attractive "carry-nothing" computing model, in which powerful computers for personal use can be found anywhere, anytime. Since pure "carry-nothing" computing approaches can make users unacceptably dependent on the availability and performance of Internet connectivity, compromise solutions in which users carry their computing environment have been proposed in the literature. To make these approaches practical, and to outweigh the benefits of "carry-everything" models, a user's personal computing environment must be conveniently, efficiently, and reliably stored and transferred between the local computing infrastructure and centralized servers. Unfortunately, all currently proposed approaches fail to meet one or more of these requirements.

We present *Horatio,* a trusted personal assistant that enhances user experience in infrastructure-based mobile computing systems. Horatio runs on smart phones (or other smart personal devices, such as PDAs, handheld PCs, etc.), leveraging three key properties of these devices: *(i)* users carry them wherever they go, *(ii)* users place high levels of trust in them, and *(iii)* they are almost always connected to the Internet, albeit with varying and sometimes limited bandwidth. Horatio devices act as trusted allies in three ways. First, they cache users' personal computing environments and make them available even when servers are not reachable. Second, they ensure that any cache modifications will be eventually stored at servers. Third, they improve the performance of both saving and retrieving a personal computing environment.

In this paper, we present the initial basic design of Horatio and a number of near-term future directions for this project. We describe how Horatio: *(i)* extends the Internet Suspend/Resume® system by introducing new state transfer protocols based upon the separation of *control* and *data* state, *(ii)* cleans cache state asynchronously without user intervention, and *(iii)* reduces both the suspend and resume latencies of a user's personal computing environment while maintaining high levels of availability and reliability.

1 Introduction

After two decades of the "carry your own laptop" model of mobile computing, an increasingly ubiquitous computing infrastructure is poised to enable a more attractive "carry-nothing" mobility model. In this new world, users will find powerful computers available to them everywhere they go, but this is not sufficient to fully realize the "carry-nothing" model. In order to represent an acceptable alternative to today's "carry-everything" mobility, an infrastructure-based solution must present users with their exact personal computing environment as quickly, reliably, and securely as their laptops do today. The Internet Suspend/Resume® (ISR) approach [16, 15] addresses this requirement by encapsulating a user's personal computing environment within a virtual machine (VM) and delivering that VM state over the Internet. In the "resume" step, a user checks out her encapsulated personal computing environment (called a *parcel*) from a remote ISR server and executes it on an infrastructure computer that she trusts (an ISR client). When she is done, the user checks in ("suspends") her parcel. The ISR system then transfers modified portions of the parcel back to its server, where it resides passively until resumed again at the same or different ISR client.

The promise of the ISR model is critically dependent on the delay experienced during the suspend and resume operations. Users perceive these delays as the price that they pay for adopting carry-nothing mobile computing. Unfortunately, the amount of VM state to be transferred can be substantial. On suspend, a cautious user (that is, one who wishes to be certain that her recent work has been saved) must wait until all modified VM state has been propagated back to her ISR server before she can depart from the ISR client. On resume, a user must wait until a bare minimum of VM state has been retrieved before she can start working (the rest of the VM state can be fetched on demand, or proactively during an ISR session). For both suspend and resume, the user-perceived delay depends directly on the performance and availability of the network between ISR client and server. In extreme cases of poor network performance, a user may in effect be denied access to her parcel, thus rendering the ISR solution unusable.

In this paper, we describe how personal smart devices (e.g., smart phones, PDAs, etc.) can be used to improve performance and availability in ISR-like models of mobile computing. We propose to use such devices as temporary trusted caches for parcels, exploiting both their storage and Internet connectivity. Since most users already carry such a device with them, we do not really violate the "carry-nothing" philosophy of ISR. By transferring parcels to/from personal smart devices rather than ISR servers, suspend and resume operations can be fast even with poor Internet connectivity. At the same time, unlike previous solutions such as SoulPad [5] that rely only on the storage of a portable device, our approach uses the Internet connectivity of a personal smart device to "clean" a cached parcel by transferring modified state to the ISR server in the background. In this way, a parcel suspended to a smart device is eventually stored safely at its ISR server, without further user actions. We thus offer a multi-hop completion path for the suspend operation, with only the first hop determining user experience.

Horatio is the term we use for the smart device in this model, the name being inspired by Hamlet's trusted ally in Shakespeare's play. In addition to its primary purpose such as making phone calls, performing PDA functions, and so on, a Horatio-enabled device also serves as a faithful assistant to a user in both the resume and suspend steps of an ISR session.

On the resume path, Horatio can serve as a *lookaside cache,* as described by Tolia et al [20]. The ISR server is contacted, but most data transfer is from the device to the ISR client. When the ISR server is inaccessible, Horatio can act as a fallback mechanism to deliver a user's parcel to

the ISR client without any Internet communication. Horatio can play a central role in establishing trust in an ISR client prior to resume, through one of several approaches based on portable storage that have been described in the recent past [18, 8]. Horatio can also be used as the boot device for an ISR client in order to establish the host operating system and virtual machine monitor (VMM) context for ISR suspend and resume operations.

On the suspend path, Horatio plays a more complex role. We have augmented the ISR protocol to include Horatio as a second (multi-hop) path between the ISR client and the ISR server. When a user suspends her current ISR session, state transfers are initiated by the ISR client both with the ISR server and with Horatio. From the user's viewpoint, the suspend is declared complete when all of the modified state is safely transferred to Horatio: the user is now free to leave the ISR client. Although modified ISR state may continue to be transferred from that ISR client to the ISR server, that state will not be trusted by the ISR server. However, the ISR state stored on Horatio is trusted, and will be eventually transferred to the ISR server or will be used to validate any untrusted state received by the server from the client. These actions use Horatio's Internet connectivity (such as 3G cellular communication) or opportunistically use WiFi, WiMax, Ultra-WideBand(UWB) or other communication technologies as the mobile device receives coverage from them. When all modified state has been propagated to the ISR server (regardless of the path through which it got there), the cache copy on Horatio is declared clean.

Suspend and resume times can be further reduced by applying a number of optimizations, some of which are also relevant to reducing energy consumption on a Horatio device. Essentially, these optimizations perform the state transfers relevant to suspend and resume speculatively and opportunistically, before the user declares her intention. In the suspend case, the speculation is that the transferred state will not be further modified during the rest of the current ISR session. In the resume case, Horatio can speculate on the identity of the next ISR client location through a variety of predictive approaches. It can then use its predictions to warm that ISR client by proactively transferring ISR state there. Some ambiguity in the precise identity of a future ISR client can be tolerated if a collection of such clients use networked storage for their ISR caches.

The remainder of this paper is organized as follows. Sections 2 and 3 further motivate this work and place it in context. Section 4 describes the Horatio design, while Section 6 discusses various key points along with future work. Finally, Section 7 reviews the work related to Horatio, and Section 8 concludes the paper.

2 Motivating Examples

In this section, we describe two scenarios that illustrate difficulties in Internet delivery of user parcels on demand anywhere and at any time. In each case, we also describe how Horatio enables users to overcome the relevant issues. These are, of course, only representative scenarios — there are many other scenarios in which Horatio can be helpful.

2.1 Example 1: Oasis of Connectivity

The first example is one in which there is an *oasis of connectivity*. Figure 1 illustrates the scenario for this example. In the figure, there are two large geographic areas represented: the United States and Romania. Within the U.S., an ISR user can travel from client site to client site, resuming her

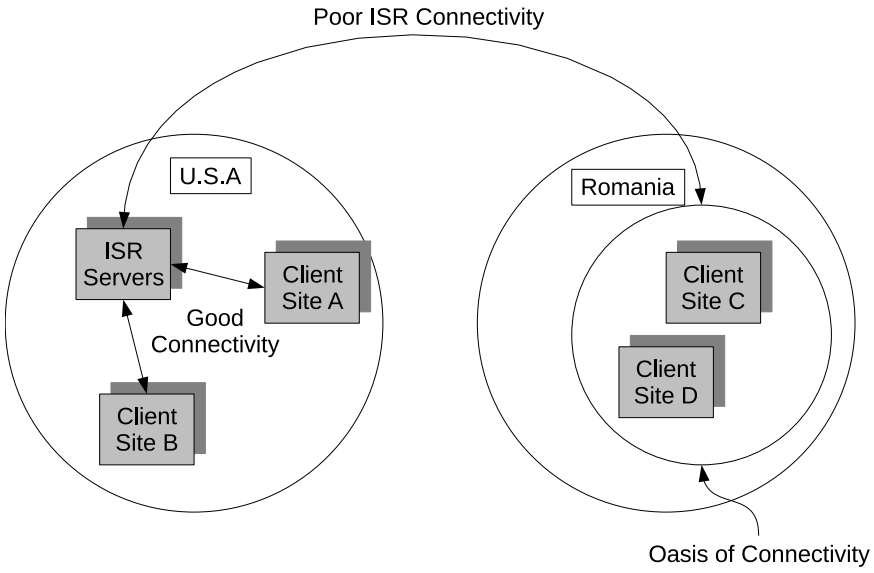


Figure 1: Oasis of connectivity example.

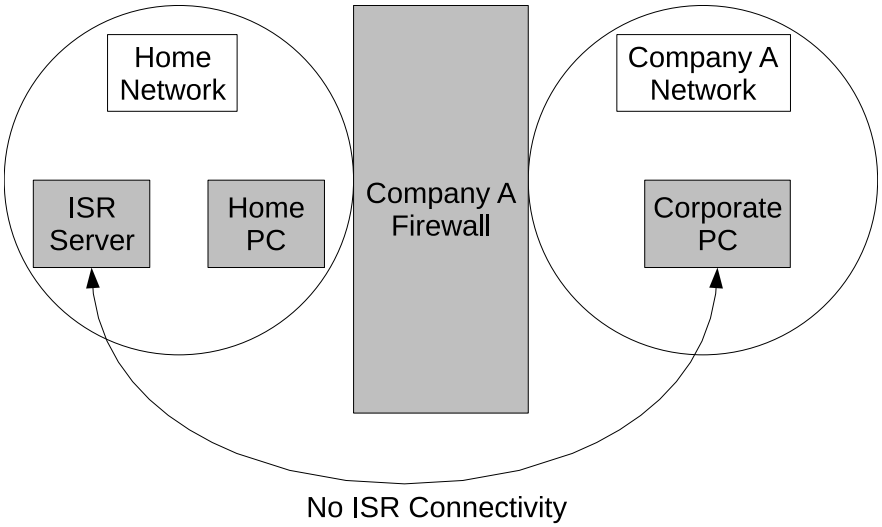


Figure 2: Corporate firewall example.

sessions directly from centrally located ISR servers and suspending her sessions back to the servers when she is ready to leave a site.

Should the user travel to Romania, she may find good connectivity between sites within the country (C and D in the example), but poor connectivity to ISR servers in the U.S. If this connectivity were intermittent or very low bandwidth, it could substantially inflate the user's suspend and resume times, to the point of rendering ISR unusable.

Horatio offers a powerful solution to this problem. With Horatio the user can suspend her session to her trusted device while she is still in the U.S., prior to traveling abroad. When she reaches Romania, she can resume right from Horatio and continue working where she left off. As she moves from Site C to Site D, for example, she can quickly suspend and resume to and from

Horatio with performance similar to what she experiences while in the U.S. Additionally, as she travels from site to site in Romania, Horatio can opportunistically take advantage of transient good connectivity to the U.S. to incrementally propagate modified state to ISR servers. Finally, when the user returns to the U.S., Horatio can complete any remaining state transfer and synchronization steps.

2.2 Example 2: Corporate Firewall

The second example illustrates a different scenario involving a user who finds herself migrating between her home network and the corporate network of a customer whom she supports. Figure 2 illustrates the scenario for this example. While on her home network, she can resume and suspend her ISR session as normal. Unfortunately, when she travels to the customer site, she finds that their strict set of firewall policies prevents her from resuming or suspending her sessions from any local PCs that she might be able to access.

Again, Horatio solves this problem. Since she can carry her suspended session into the client site on her Horatio device, she can freely resume her computing environment when she arrives. Later, when she is ready to leave and go back to her home network, she can suspend her ISR session to Horatio and carry her modified state back out with her. When she arrives back to a PC on her home network, she can resume her session, or synchronize it directly back to an ISR server. Alternatively, if her Horatio device is equipped with cellular or WiMAX communication capabilities, Horatio is able to bypass her customer's firewall and to start synchronizing modified ISR state back to its server before she returns to her home network. In effect, the use of Horatio enables "sneakernet" to augment use of the Internet.

3 Taxonomy of Infrastructure-Based Mobile Computing

A number of solutions have been proposed to provide "carry-nothing" mobile computing environments to users. The solution space can be described along two dimensions. One dimension corresponds to the location of the user's computing environment when it is not in use (i.e., the "parcel home," in ISR terminology). The other dimension corresponds to where the user's computing environment is currently instantiated and in use (i.e., the "parcel execution site," in ISR terminology). Figure 3 illustrates the design space defined by these two dimensions. In the figure, each of the two dimensions ranges from local to remote. We divide this space into four quadrants and describe the characteristics of each quadrant in the rest of this section. For ease of exposition, we use the term "parcel" to mean "user's computing environment" even in non-ISR models.

3.1 Local Home/Local Execution

The local-local approach is exemplified by SoulPad [5], and corresponds to the scenario where the user carries a personal state storage device (e.g., USB drive, iPod, etc.) that contains the parcel. A user resumes operation by connecting her personal storage device to a trusted computer, booting the VM, and resuming her PC session directly from the personal storage device. At suspend, the modified parcel is stored back onto her personal storage device.

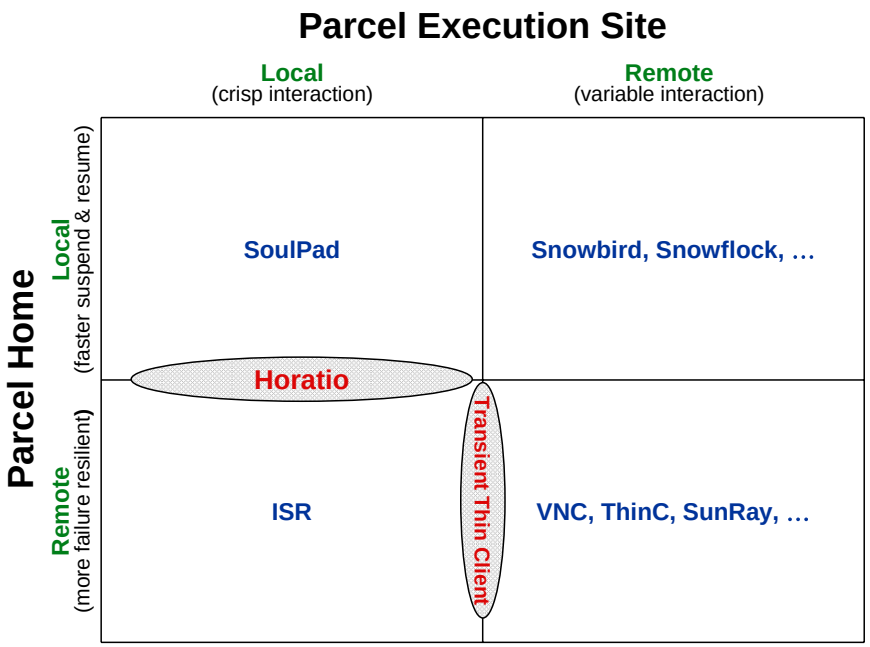


Figure 3: Taxonomy of Approaches to Infrastructure-based Mobile Computing

SoulPad provides good performance in terms of suspend and resume times, since a parcel is stored locally and can be accessed with low latency. The time required to suspend or resume is only dependent on the speed of the local interconnect (e.g., USB). This approach also provides the user with a crisp interactive computing experience [19], since execution occurs locally. The local-local approach provides good availability, since the user has direct control over her personal state storage device, and always carries this device with her wherever she goes. No Internet connectivity is required. Unfortunately, this approach has poor robustness. Since the parcel home is a personal storage device, it is vulnerable to damage, catastrophic failure, loss or theft. These events lead directly to loss of the user's parcel. A careful regimen of backups can help, but few users are sufficiently self-disciplined for this to be a satisfactory solution.

The classic unvirtualized PC model can also be viewed as occupying the local-local quadrant of Figure 3. In this case, the local disk on the PC serves the role of personal state storage device. By removing the local disk, carrying it to a different machine, and booting it there one can achieve a SoulPad-like user experience. The "parcel" in this case is the entire local disk.

3.2 Local Home/Remote Execution

The local-remote approach corresponds to the scenario in which remote computational resources are used to execute a locally-resident parcel. Some scenarios of the emerging "cloud computing" model correspond to this quadrant of Figure 3. A specific example of this approach is the Snowbird system [10], which is designed for a class of applications called *bimodal applications* that have both interaction-intensive and resource-intensive phases of execution. Such applications occur in many domains including scientific computing, digital animation, CAD and computational biology.

During execution, Snowbird dynamically migrates a parcel to the optimal execution site based on its current characteristics. When the application is executing in an interaction-intensive phase,

it will execute local to the user's site for crisp interactivity. During a computationally-intensive phase, Snowbird migrates the application to execute at a more appropriate site depending upon the resource needs of the application (e.g., to a compute server farm, to the site of a large data warehouse, etc.) To improve the efficiency of VM migration, an execution site may prefetch and/or retain cached parcel state even when it is not the current execution site of that parcel.

Another example of this model of execution is the Snowflock system [11], which enables a single local parcel to be efficiently cloned on a remote compute cluster for parallel execution.

3.3 Remote Home/Remote Execution

The remote-remote approach, typically referred to as "thin client computing" [1, 2, 3, 4], corresponds to the scenario where the user parcel is stored and executed on a centrally-located server and accessed remotely though the use of stateless clients. A user logs on to the server via any client; that client provides the display, keyboard, mouse and the bare minimum of local computing necessary for rendering and user interaction. The client is thus the modern counterpart of a "dumb terminal" from the timesharing era. All user interactions are sent to the central server and display results are returned to the stateless client. In this way, all execution and state changes occur directly at the central server. As a result, when the user suspends the session, there are no state updates to be sent to the server.

Thin client performance critically depends on the performance characteristics of the interconnecting network. In some cases, it could be dependent on network components not directly under the control of the user or user's organization (e.g., wide-area interconnection over the Internet). Without direct end-to-end control of network latency, it is very difficult to ensure crisp interactive response for a good user experience [19]. System availability is likewise dependent on the availability of the network between the thin client and server. On the other hand, this approach is robust against loss, damage or theft of hardware at the edges of the network because the user's parcel never leaves the central server environment.

3.4 Remote Home/Local Execution

The fourth approach, remote-local, combines the crisp user experience of local execution with good robustness. In the remote-local approach, a user client contacts a central server to fetch a copy of their parcel. The user session is then resumed. After the user's session is suspended, updates to the parcel are propagated back to the server. The parcel can then be deleted at the client, if desired. An Internet Suspend/Resume system exemplifies this approach.

Since a parcel is transferred just before a resume and just after a suspend, both delays are directly dependent on the speed of the network interconnect between ISR clients and servers. A sufficiently slow network may, in effect, deny a user timely access to her parcel. One can hoard ISR state in advance of use at an ISR client to greatly reduce resume delay, but this is only possible if the identity of that client is known in advance.

Both suspend and resume performance of ISR can be improved through the use of additional hardware elements. Mobile lookaside caches can be used to improve the resume performance by using portable storage devices [20] to keep a cache of a user's computing environment with the user. Staging servers [7] can also help reduce resume performance overhead by introducing an

active element that prefetches a copy of the required resume state in order to bring it "closer" to the resuming host.

Suspend performance can be improved through the use of waystations [9]. This approach utilizes additional active hardware elements to stage state updates for network file systems. These waystations provide a higher-speed, lower latency location for clients to stage state updates, which are then propagated to a server at a later time. Adding these additional hardware elements in the suspend (or write) path allows for a suspend to occur in two stages. The first stage moves updates from the client to a waystation. At this point, the user's state is cleaned from the client and the user is free to move to a new location. During the second stage, the waystation propagates the user updates to the central server. Since this stage overlaps with the user's movement between locations, the user does not have to wait for the state updates to be written to the server.

A security issue is however raised by introducing an additional element on the suspend path. Users must trust the waystations to correctly propagate their state updates back to the server. Although it is possible for a user to verify that what has been propagated is correct, there is no way for a user to enforce that waystations propagate the updates, in the first place. Therefore, it is possible for a malicious (or possibly failure-prone) waystation to accept updates from users and to never forward them. Furthermore, there is no way for a user to guarantee a priori that any particular waystation or set of waystations will behave correctly. Replicating state updates to multiple waystations in a Byzantine fault tolerant fashion [12, 6] may help, but also increases the cost of deployment in terms of the additional hardware elements required and increased protocol complexity. The critical difference between a waystation and a Horatio device is that the latter is under the direct control of the user and can therefore be trusted by her.

3.5 Hybrid Approaches

Figure 3 also presents two hybrid approaches. These approaches straddle the middle-ground between quadrants, adapting one of the two dimensions to more closely track user requirements.

3.5.1 Transient Thin Client

One hybrid approach is the Transient Thin Client approach [15]. This approach has been proposed to reduce the resume latency for an ISR session by leveraging the key idea of Snowbird: to allow a user's ISR session to start execution in thin client mode while the relevant parcel state transfer occurs. When the transfer completes, execution switches to thick client mode and is indistinguishable from a normal ISR session.

3.5.2 Horatio

The second hybrid approach is Horatio, which is the primary focus of this paper. Horatio can be seen as a hybrid between ISR and SoulPad (i.e., a lazy ISR or an active SoulPad). As shown in Figure 3, Horatio combines the advantages of both approaches while eliminating some of their disadvantages. First, Horatio exhibits better robustness than SoulPad by being a self-cleaning cache, which only temporarily stores the modified state on the mobile device until it is fully transferred to the server. In fact, should Horatio be destroyed or lost before propagating all of the modified state to the server, a user can roll back to the last consistent copy of her parcel from the

State Name	State Type	Description
Memory Image	data	Contains the encrypted memory image and execution state of the suspended VM that executes the user's session.
Disk Image	data	Contains the encrypted chunks of the VM disk accessed during the most recent resume/suspend cycle for the virtual machine.
Keyring	control	Stores the encryption keys and cryptographic hashes of the virtual disk chunks.
Configuration File	control	Stores various operational parameters of a parcel for the ISR client to use during resume and suspend. Also stores the encryption key used to encrypt the keyring and virtual memory image.
Nonce	control	A unique identifier generated when a parcel is checked out from an ISR server. Existence of this nonce for a parcel implies that a parcel is checked out. If the nonce does not exist, the server is the owner of the parcel.

Figure 4: Description of data and control state within ISR.

server. Horatio thus achieves robustness as a natural part of its workflow, rather than demanding voluntary adherence to a careful backup policy. Second, Horatio improves the suspend/resume performance of ISR by incorporating the advantages from the SoulPad model. It minimizes the suspend latency of ISR by allowing modified state to be transferred to a local device, hiding its propagation latency to the server. It also reduces the resume latency by providing a lookaside cache at the user's next resume location. Third, by acting as a mobile trusted waystation, it removes the trust issues in multi-hop state propagation to ISR servers. Finally, predictive software on a Horatio device can trigger cache warming on future ISR clients and thus reduce resume latency there.

4 Design

In what follows, we will describe the design of Horatio starting from the basic ISR model. In ISR, a user's computing environment is completely contained within a VM. As a result, migrating a user's session requires suspending the VM and migrating the associated on-disk state to another computer. As mentioned earlier, this state is called a *parcel* in ISR terminology. A parcel includes a nonce, an encrypted memory image, an encrypted disk image, an encrypted keyring, and a configuration file. Figure 4 describes each element in more detail.

Horatio separates parcel state into two parts: *data state* and *control state*. *Data state* is the parcel state required to resume an ISR session, and may be stored in more than one place at the same time (e.g., the Horatio device and an ISR server). *Control state* is the parcel state that must be possessed in order to modify the data state, and effectively acts as a "lock" on the data state. Control state is much smaller than data state: merely 5.5 MB for a typical ISR parcel whose data state is 8 GB.

The rest of this section describes how state is transferred between ISR clients and Horatio, and between Horatio and ISR servers. This includes a description of the basic state transfer protocol, as well as parcel ownership and transfer.

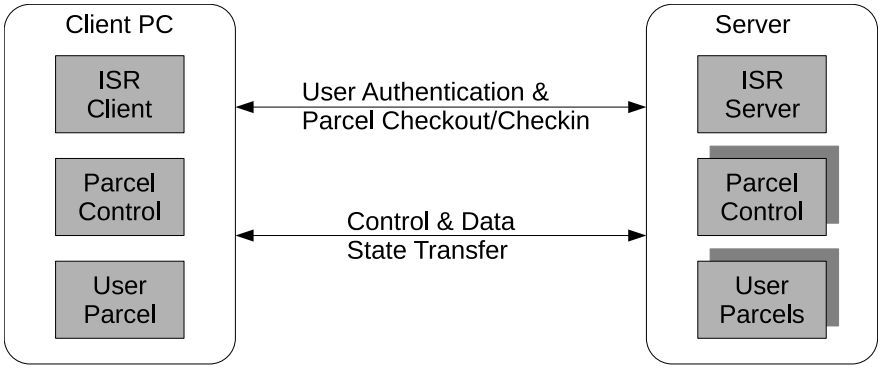


Figure 5: Standard ISR state transfer.

4.1 State Transfer

The standard ISR model involves state transfers only between clients and servers. In this model, shown in Figure 5, a user logs onto a trusted ISR client machine and initiates the state transfer process between the local client and a remote ISR server. This process authenticates the user, verifies that the requested parcel is not currently in use (e.g., checked out on a different client by the same user), and transfers parcel control and data to the client. Once this is complete, the client resumes the user's session. When the user's session is suspended, the client transfers the modified portions of the parcel back to the server. Finally, control is transferred once the modified data state has been received by the server. The server then validates the data and control state transferred from the client, and commits these changes to its local copy of the parcel.

With Horatio, the bidirectional state transfer is transformed into a triangular model, as shown in Figure 6. In this model, users have the option to defer the state transfer from client to server by transferring data and control to a mobile device that they carry with them (Horatio device). Eventually, the data state reaches the server from the client, from Horatio, or from both. Once the modified portions of the parcel have arrived at the server, Horatio can validate them and transfer control to the server.

There are numerous paths over which state transfer can occur and Horatio is designed to take advantage of any available connectivity to the client and to the server. For example, Horatio might communicate with the client over USB, Ethernet, WiFi or Bluetooth, and with the server over WiFi, WiMax, UWB, or a cellular data network.

4.2 Parcel Ownership

Exactly one site (client, server, or Horatio) *owns* a parcel at any given time. Ownership of a parcel is explicitly transferred through the protocol described in the next section. However, all or part of that parcel's data state can be freely prefetched from its server or copied between clients without involving the owner. These copies will need to be validated by the parcel owner before use.

Only the owner of a parcel can modify its data and control state. This typically occurs after a user resumes an ISR session. Since a client may choose to fetch data state from a server only on demand, ownership of a parcel does not necessarily imply possession of its entire data state.

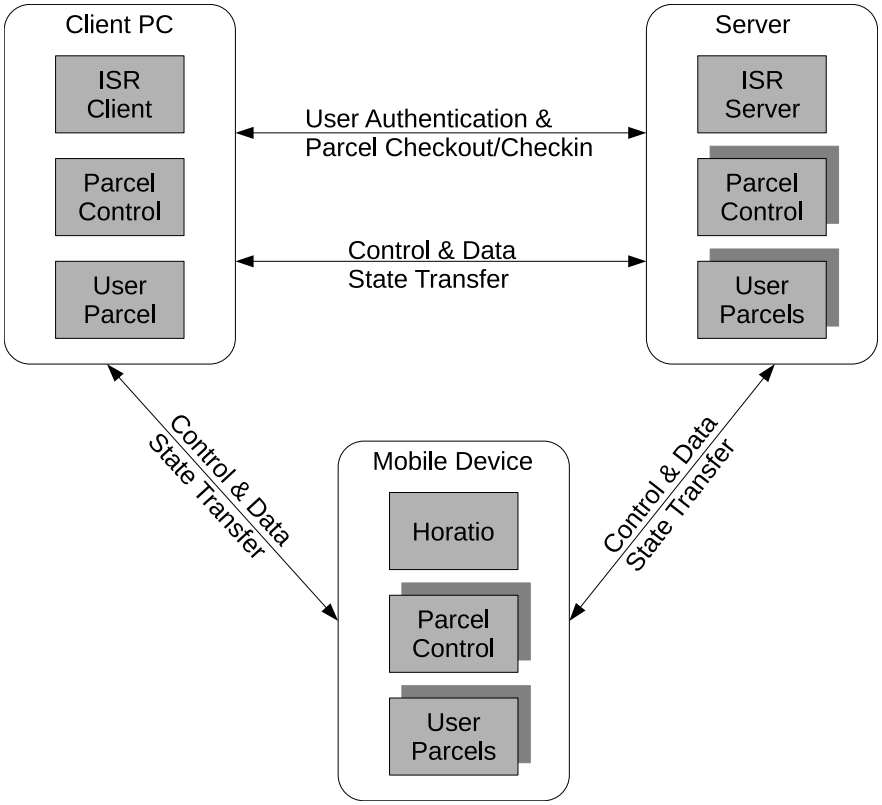
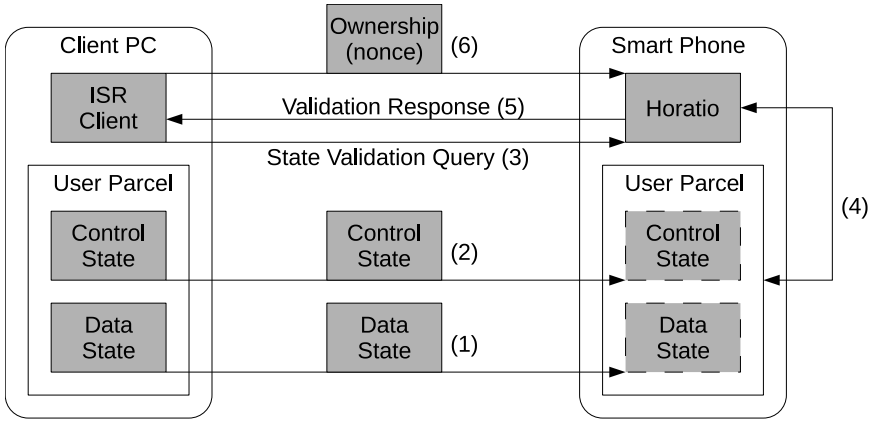


Figure 6: ISR state transfer with Horatio.

Only the owner of a parcel can validate parts of its data state. There are many circumstances in which such validation is necessary. For example, residual cache state at a client from a previous session of a user will need to be validated before use. The client, being the parcel owner after resume, can perform the validation to decide whether cache state can be reused or has to be fetched afresh. As another example, only the specific Horatio device on which a user performed a suspend of a parcel from a client can validate data state that has been transferred by that client to the server after suspend.

There are three possible parcel owners. The first case is when the latest version of a parcel's data and control state has been checked in. In this case, the server is the owner and no client can be executing a user session for that parcel. The second case is when a parcel has been checked out by a client. In that case, the parcel is owned by the client, regardless of whether a user is active at the client. The third case is when a user session has been suspended to a Horatio device. In that case, the device is the parcel owner.

By default, Horatio never retains ownership longer than necessary: it transfers parcel ownership back to its server as soon as the server has received all modified data state for that parcel. However, a user may choose to have Horatio retain ownership even after the complete data state for a parcel is safely on its server. This may be valuble, for example, when the user knows ahead of time that she may wish to resume a parcel from Horatio because of poor Internet connectivity at a site that she plans to visit.



(1-2) ISR client propagates data and control state to Horatio. (3) ISR client issues state validation query to Horatio. (4) Horatio validates transferred state using cryptographic hashes and (5) responds to ISR client's query. (6) ISR client transfers ownership to Horatio by sending the unique nonce associated with the transferred parcel to Horatio, using a two-phase commit protocol. At this point, Horatio is the owner of the parcel. The ISR client is now free to delete parcel state.

Figure 7: Horatio Ownership Transfer Protocol.

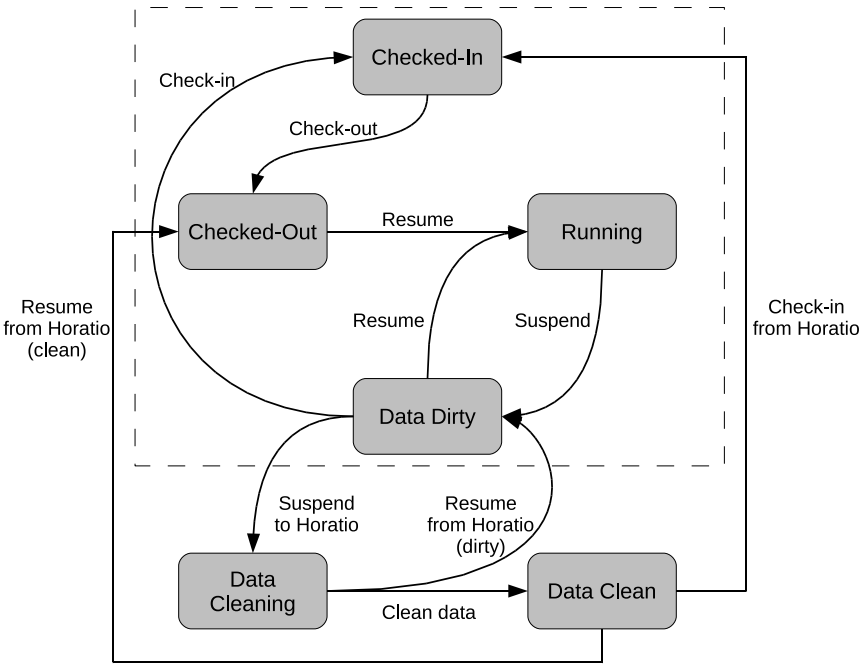
4.3 State Validation and Ownership Transfer

Ownership transfer must follow certain rules in order to preserve correctness. A parcel must be owned by exactly one site at a time. Its ownership can only be transferred between trusted sites. With respect to a specific parcel, data state on a client or Horatio device is said to be *dirty* when it has been modified more recently than that parcel's data state on the server. Ownership can only be transferred to a site after all dirty state has been transferred to the same destination. This rule ensures that the parcel owner is always able to reconstruct a complete copy of the parcel.

Enforcing these rules is the job of the ownership transfer protocol, which has to be resilient to network, server, client and Horatio device failures. Figure 7 shows the protocol. In the figure, the user has suspended an ISR session and has just initiated the state transfer process to migrate his suspended session from the client to her Horatio device (in this example, a smart phone). First, the client propagates all of the dirty data state for the parcel to Horatio. Once this has been completed, the client transfers the control state to Horatio.

After control state transfer is complete, the client queries Horatio to verify that all modified state has been properly transferred. Horatio uses the control state to generate a list of the modified disk chunks and verifies that *(i)* all of the chunks are locally present and *(ii)* the cryptographic hash of the chunks match those stored in the control state. Then, Horatio generates cryptographic hashes for the memory image and execution state, the parcel keyring, and the parcel configuration file. These hashes are sent back to the client as a response to the validation query.

The client verifies the hash returned by Horatio against hashes generated locally. If they match, then ownership is transferred from the client to Horatio using a two-phase commit protocol. In the prepare phase, the nonce is sent to Horatio by the client. Horatio writes the nonce to a temporary file and returns an acknowledgment to the client. When the client receives the acknowledgment, it moves its copy of the nonce to a temporary file and issues a commit request. Horatio then commits the transaction by moving the nonce from a temporary to a permanent file. At this point, parcel



The states surrounded by the dashed line represent the most common ISR states and associated state transitions. The states outside the dashed line represent the newly added Horatio states and state transitions.

Figure 8: Horatio Data and Control Transfer Finite State Machine.

ownership has been successfully transferred. Horatio now informs the client of this fact. The client is then free to delete data and control state.

In this example, data and control were transferred from a client to Horatio. This transfer can also happen in the opposite direction, from Horatio to a client, should a user arrive at their next location prior to control being transferred from Horatio to a server.

The separation of data and control transfer provides three direct benefits. First, propagation of data state back to a server can be performed at any time prior to control transfer. As a user travels between computing locations, her Horatio device can transfer her parcel's modified data state wirelessly to the server. Once a valid copy of the modified state reaches the server, the control state can be transferred and the parcel committed. Second, since data is propagated ahead of control transfer and validated by the ownership transfer protocol, it is possible for the data state to be transferred to a server (or any other target site) from multiple, possibly untrusted, sites. By only requiring trust establishment between sites during control transfer, not data transfer, Horatio provides additional opportunities for concurrency and network opportunism in data state propagation. Finally, since all data state is stored in encrypted form, it is possible for ownership to be transferred to Horatio and for the user to leave the site while a copy of the data state remains on the previously trusted client. This enables the client (now, untrusted) to continue to propagate data state to the server, possibly in parallel with Horatio, while the user travels.

Figure 8 describes the data and ownership transfer protocols as a finite state machine. The figure illustrates the typical ISR states and state transitions (those that are inside the dashed box) and the new states and state transitions introduced by Horatio. A typical session starts when a user

checks out her ISR parcel on a trusted client, represented by the transition from the CHECKED-IN state to the CHECKED-OUT state. This also represents ownership transfer from the ISR server to the client. Afterward, the user will resume her session, represented by the transition from CHECKED-OUT to RUNNING, and will interact with her parcel. Once she is finished, she will suspend the session, transitioning from the RUNNING state to the DATA DIRTY state. At this point, she may choose to: *(i)* resume again (transitioning back to the RUNNING state), *(ii)* check the parcel back in to the server (transitioning to the CHECKED-IN state), or *(iii)* suspend to Horatio on her smart phone (transitioning to the DATA CLEANING state). If she decided to check the parcel back into the server, she is done and may leave the site once the check-in process completes (parcel ownership has been transferred back to the ISR server). Alternatively, if she decides to suspend the parcel to her smart phone, then she may leave once the Horatio data and ownership transfer protocol completes and this completion is represented by the DATA CLEANING state in the figure (ownership resides with Horatio now). As she transits between sites, Horatio executes the data state cleaning protocol. Completion of this protocol is represented by the transition from the DATA CLEANING state to the DATA CLEAN state. At this point, the user can decide to either resume from Horatio (transitioning back to the CHECKED-OUT state) or she may check her parcel in directly from Horatio (transitioning to the CHECKED-IN state). The former case also represents ownership transfer from Horatio to the ISR client, while the latter case also represents ownership transfer from Horatio to the ISR server. Alternately, the user may choose to resume her session from Horatio at an ISR client prior to completion of the data state cleaning protocol. This is represented by the transition from the DATA CLEANING state back to the DATA DIRTY state.

4.4 State Cleaning and Power Optimization

As mentioned above, the separation of control and data transfer allows for the concurrent propagation of data state to a server from multiple sources. This is especially beneficial to our model, since Horatio is a power-constrained mobile device. To reduce the energy demands on Horatio, we take advantage of the fact that any client or waystation can continue to transfer data state to a server even after ownership has been transferred to Horatio. This modified state may also be passed to other clients, but it cannot be committed at a server or used at other clients until it is validated by the parcel owner.

When Horatio transfers parcel ownership to a server, any data state previously propagated to the server will be validated. Opportunistic data transfers can lead to multiple uncommitted versions of partial data state at the server, which cannot be discarded until the server regains ownership of the parcel.

4.5 Lookaside Caching

After Horatio has finished state transfer and relinquished ownership, it may retain a copy of the most recently modified data state for use as a portable lookaside cache similar to [20]. To use Horatio in this way, a user connects the Horatio device to the client computer upon arrival at the next computing location. Whenever the client finds that it needs to fetch a data chunk from the server, it first sends a request to Horatio for the missing chunk, identified by its cryptographic hash. If Horatio possesses the data for a chunk matching that hash, it returns the chunk to the

client. Otherwise, it returns a negative response and the client fetches the data chunk from the server.

4.6 Client Boot Image

Horatio may optionally include an operating system image, including a VMM and ISR client software, which can be used to boot a computer via a USB or similar connection. If the user encounters a potential resume site which does not have an ISR client installed, or if she does not trust the software installed on the target computer to be free from malware, she can boot the computer directly from Horatio and immediately have access to a working and trustworthy ISR client. The user may choose to resume her parcel directly from Horatio, bypassing the client's disk entirely. Or, for improved performance, the user may boot from Horatio but use the client's faster disk to store parcel data. In the latter case, when the user suspends her parcel, ISR will transfer modified parcel data to Horatio and remove the parcel from the client's disk.

Suspending and resuming parcels while booted from Horatio is a seamless extension of the model previously described; ISR detects Horatio's current role and ensures that parcel data is properly located and updated. At resume, if Horatio already owns the parcel, the ISR client automatically locates and uses the parcel state; otherwise, it is fetched from the server as usual. On suspend, Horatio retains ownership of the parcel, without the need for an explicit transfer from the client computer. Because parcel state created when booting from Horatio is compatible with state created on an infrastructure-based client and then transferred to Horatio, the user can alternate between these two models as appropriate for her circumstances.

4.7 Robustness

In our proposed model, state reliability is a concern because the Horatio device might be lost, damaged, or run out of power. If this occurs while Horatio is the parcel owner, there are two detrimental effects. First, any modified control and data state stored on the device could be lost. Second, since the device was the last parcel owner, any other site would be prevented from checking out a copy of the parcel from the server.

To handle these issues, we provide a server operation that allows a user to override the lock on a checked-out parcel and roll back to a previously committed version of the parcel. In this way, the server becomes the parcel owner and a user can check out a new copy of her parcel. In this case, the nonce associated with the old control state on Horatio no longer matches the newly-generated nonce stored in the new control state. Hence any state held by Horatio is considered invalid.

5 Implementation Roadmap

We are currently working to complete a prototype implementation of Horatio, which includes all of the features currently described in the system design. We are targeting both latest generation smartphones and PDAs for the Horatio prototype. Finally, we plan to evaluate Horatio utilizing various mobile device hardware platforms over different available network interconnects (e.g., cellular, WiFi,USB, etc.).

6 Discussion and Future Work

In this section, we describe a number of outstanding issues and possible research topics that we plan to explore in the near future.

6.1 Power Management

Like all battery-powered mobile devices, Horatio is subject to power constraints. Furthermore, Horatio is meant to coexist on a device with an unrelated primary function (e.g., cell phone, PDA, media player). Therefore, in this design, we must carefully consider the trade-offs between battery life for primary device functionality and Horatio functionality. With respect to the latter, we must balance mobile device battery life against state availability, and also against suspend/resume time. In both cases, we must decide how much state Horatio will transfer while running on battery power.

In the first case, the trade-off is between power and availability. Horatio can improve availability by more aggressively transferring modified state back to an ISR server. This assumes that there is adequate power left for the device to complete the transfer. Alternatively, in a low battery situation, it might be better to conserve power by waiting for the user to arrive at the next compute location, where the device can run on wall power. In this way, Horatio will be better able to preserve both data availability and power. It might be possible to leverage existing work [14] in user behavior as it relates to battery lifetime and charging opportunities, to explore the trade-off between power and availability in Horatio.

In the second case, the trade-off is between power and suspend/resume time. The minimum suspend/resume times are likely to be those in which a large portion of a user's session state is located locally on a Horatio device. Assuming a high-speed, low-latency connection between Horatio and the ISR client PC, the best overall performance a user can experience is when most or all of his computing state is on Horatio. There are cases when this does not hold, such as when an ISR client and server are located on the same LAN. For the sake of this discussion, we ignore these cases as Horatio would not be needed.

Since the best performance is determined by how much of the user's state resides on Horatio, there is a direct trade-off between the amount of state transferred to Horatio and the suspend/resume performance. At suspend time, the minimum requirement is for the modified state to be transferred to Horatio. It is possible, though, for unmodified state to be transferred as well, to allow Horatio to function more effectively as a lookaside cache. There is related tension between suspend and resume times, as well as resume time and power. Specifically, the more unmodified state that is transferred during suspend, the longer the suspend time, but the shorter the resume time. Additionally, the longer the combined suspend/resume time, the more battery power is used.

As future work, we plan to explore all these trade-offs. The goal of this exploration is to determine the correct set of policies to balance the primary functionality of users' mobile devices against the secondary needs of Horatio. We also plan to explore adaptive policies.

6.2 State Availability and Recovery

Section 4.7 noted that dirty parcel state can become unavailable if a Horatio device fails or runs out of power shortly after a user has suspended an ISR session to it. As discussed earlier, the user

may recover access to her parcel by discarding the changes stored in Horatio and rolling back her parcel to its state at the last server commit.

However, this raises an interesting issue: the state stored on the Horatio device would still be valid, were it available (which could happen after the battery is recharged). By choosing to roll back to a previous commit point and resume her ISR session from there, the user is forced to discard any progress made. Although this can be considered an exceptional case, it is also one that is likely to occur in practice.

A more flexible approach would allow for multiple versions of a user session to exist, such that both availability and reliability concerns are met. To support this, though, requires a rethinking of the standard ISR model to allow for multiple versions, while avoiding conflicts in parcel ownership and state versions. As near-term future work, we intend to incorporate a simple versioning scheme that allows for multiple session branches to exist for a user's parcel. As a longer-term goal, we intend to explore possible ways to merge divergent sessions, such that a user's data state can be recombined from differing parcel versions. Ideas from Coda [17] and other systems that use optimistic replication are relevant here.

6.3 Location Prediction and Resume Optimization

Today, most mobile devices support some form of location awareness through the use of various localization technologies (e.g., GPS, cellular localization, WiFi localization, etc.) Therefore, it might be possible for Horatio to track a user and learn her movement patterns over time. With this information, Horatio might be able to predict a user's movement between computing sites, and use these predictions to prepopulate a user's likely next resume site. The effect of this optimization is to potentially reduce the resume time that a user experiences when she finally arrives at her next destination.

There are three possible models to explore. One is *machine prediction*. In this model, Horatio is able to predict the specific PC where the ISR session will be resumed. For example, a user might be on her way to her home or office. The second model is *site prediction*. In this model, Horatio cannot predict the specific machine, but can narrow it down to some small set of machines at a specific site. For example, a user's office might contain two machines she uses on a regular basis. The third and final model is *oasis prediction*. In this model, the site can be predicted only to coarse granularity, and there is uncertainty in exactly which one of of a large set of possible machines in the "oasis" will be the next resume site of the user.

As an extension to location prediction, it might be possible to incorporate non-traditional sources of location into the model. For example, most mobile devices support a calendar. For users that maintain fairly accurate calendars on their mobile devices, this information can also be leveraged by Horatio as an alternate source of location information to predict the next site at which a user will resume her session.

To illustrate the value of using a calendar to aid in location prediction, consider a user who suspends her session at one location (Site A), and then travels to a second location (Site B) to participate in a brief meeting. Finally, after her meeting has concluded, she travels to a third site (Site C) and resumes her previously suspended ISR session. Since the calendar can also provide time duration at a site, in addition to location information, Horatio might be able to correctly predict that Site C is the next resume site, rather than Site B.

6.4 User Interface

Once a user suspends her current session to Horatio, she will still have to wait until parcel state and ownership have been transferred to her Horatio device. The time that any such suspend will take depends, primarily, on the amount of modified state that has built up during her most recent session. Once a session has been suspended, a user would like to be able to leave that site as soon as possible. Therefore, as an aid to users, it makes sense to provide feedback to them with respect to the amount of time a suspend will take. In fact, a user may wish to periodically copy modified state to her Horatio device, in order to reduce the suspend time once she is ready to suspend. This may occur by explicit user command, or transparently in the background using a mechanism similar to trickle reintegration in Coda [13].

6.5 User-Specified Suspend Latency Bounding

As discussed in the previous section, we intend to develop GUI mechanisms to inform a user of the current length of her suspend latency. This provides direct feedback to the user regarding the minimum amount of time she must wait before she can leave her current site. A related direction of future work is to incorporate a mechanism that allows a user to specify a suspend latency bound for her current computing session. This mechanism would allow a user to specify the maximum amount of time she is willing to wait for the suspend operation to complete.

We envision three possible ways in which a suspend time bound may be enforced. In all three cases, the system must monitor the amount of dirty state that is pending transfer to Horatio. Given the networks available to transfer state between the client and Horatio, the system must estimate the time for the suspend transfer to occur, if it were to start immediately.

The first enforcement technique adjusts the performance of the VM in order to constrain the amount of dirty state pending transfer to Horatio. This can be accomplished by slowing or pausing the user's VM execution whenever the dirty state pending transfer exceeds a predefined threshold. This effectively would allow the state transfer to "catch up" after it falls behind during some peak amount of state modification. The drawback to this method is a reduction in the crispness of the user experience for interactive workloads. One potential optimization is to only reduce performance during state modification operations, in order to constrain the degradation to those operations that directly impact the suspend time bound (i.e., those operations that generate dirty state).

The second enforcement technique we intend to explore is the possibility of using the VM checkpoint and rollback mechanisms to meet the user-specified suspend time bound. With this technique, we allow the VM to take periodic checkpoints. Whenever the user's workload will potentially cause her specified suspend time bound to be exceeded, the system notifies the user and offers to roll back to the latest consistent checkpoint. At this point, the user can choose to suspend to Horatio or continue to work. Either way, the system state is reverted to a known point when the time bound was being met so that the client can continuously enforce this guarantee for the user. One benefit of this method over the others is that it allows the user to decide whether or not to exceed her own time boundary. Another benefit is that it provides the user a way to "undo" operations that are costly with regards to the amount of dirty state that is generated. For example, a user may decide to abort a large download once she realizes the impact of that download on

suspend latency. The primary drawback of this method is the cost in performance and storage imposed on the system due to the VM checkpointing.

Finally, it is possible to bypass enforcement altogether and to simply notify the user when her current workload behavior is exceeding the specified time bound. Rather than simply monitoring the user's computing environment and notifying her when the time bound has been exceeded, Horatio could potentially predict, based on historical context, when the user is likely to exceed the time bound. In this way, Horatio could provide two metrics to a user. The first is the current performance with respect to the time bound and the second is the predicted performance based on the current workload and historical context. The ultimate goal of this technique is to allow the user to adjust her workload behavior by giving her continuous fine-grained feedback from the system.

7 Related Work

As described in Section 3, the design space for infrastructure-based mobile computing is very broad. In this space, Horatio can be seen as a hybrid between the standard ISR [16] and SoulPad [5] approaches. SoulPad completely encapsulates a user's computing state within a virtual machine on a passive storage device (such as a USB drive), but has no backup of the storage if the device is lost. ISR provides good robustness, but at the price of increased suspend and resume latency. Our work introduces an active mobile component as computing state manager to improve robustness, while also reducing resume and suspend times.

Our model proposes a mobile, Internet-connected device as a temporary, self-cleaning storage location for state updates during client session suspends and as a state cache during resumes. This is inspired by previous work on fluid replication [9], data staging [7], and lookaside caching [20]. By positioning a user's mobile device as the central active state manager, Horatio is the right element in the right place to provide the most flexibility to the user to manage their computing state. Additionally, Horatio can take advantage of other coincident properties, for example user location and mobility, to improve reliability, performance, and usability for infrastructure-based mobile computing.

8 Conclusion

In this paper, we have presented Horatio, a self-cleaning portable cache for infrastructure-based mobile computing that uses personal smart devices with local storage and Internet connectivity to transfer the state of a user's computing environment. In using mobile phones, devices that most people carry with them wherever they go, we take advantage of the trust users already place in these highly personal devices. Additionally, smart phones (or any similar portable computing platform) enable more than just simple cache storage (e.g., USB storage devices). As active elements, smart phones provide a powerful platform upon which we can improve both the performance and availability of current infrastructure-based computing techniques, such as ISR, while maintaining the high level of reliability already present in these systems. Along with the initial basic design, we have also presented a number of enhancements and a discussion of near-term future directions.

Acknowledgements

Internet Suspend/Resume and OpenISR are registered trademarks of Carnegie Mellon University. This research was supported by the National Science Foundation (NSF) under grant numbers CNS-0509004 and CNS-0520123. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF, Carnegie Mellon University, or Rutgers University.

References

[1] Citrix XenDesktop. `http://www.citrix.com/lang/English/home.asp`.

[2] Sun Ray Desktop. `http://www.sun.com/software/index.jsp?cat=Desktop&tab=3&subcat=Sun%20Ray%20Clients`.

[3] Virtual Network Computing. `http://www.realvnc.com/`.

[4] R. A. Baratto, L. N. Kim, and J. Nieh. THINC: A Virtual Display Architecture for Thin-Client Computing. In *SOSP '05: Proceedings of the Twentieth ACM Symposium on Operating Systems Principles*, Brighton, United Kingdom, 2005.

[5] R. Cáceres, C. Carter, C. Narayanaswami, and M. Raghunath. Reincarnating PCs with portable SoulPads. In *MobiSys '05: Proceedings of the 3rd International Conference on Mobile Systems, Applications, and Services*, Seattle, WA, 2005.

[6] M. Castro and B. Liskov. Practical Byzantine Fault Tolerance. In *OSDI '99: Proceedings of the Third Symposium on Operating Systems Design and Implementation*, New Orleans, LA, 1999.

[7] J. Flinn, S. Sinnamohideen, N. Tolia, and M. Satyanarayanan. Data Staging on Untrusted Surrogates. In *FAST '03: Proceedings of the 2nd USENIX Conference on File and Storage Technologies*, San Francisco, CA, March 2003.

[8] S. Garriss, R. Cáceres, S. Berger, R. Sailer, L. van Doorn, and X. Zhang. Trustworthy and Personalized Computing on Public Kiosks. In *MobiSys '08: Proceedings of the 6th International Conference on Mobile Systems, Applications, and Services*, Breckenridge, CO, 2008.

[9] M. Kim, L. Cox, and B. Noble. Safety, Visibility, and Performance in a Wide-Area File System. In *FAST '02: Proceedings of the 1st USENIX Conference on File and Storage Technologies*, Monterey, CA, 2002.

[10] H. A. Lagar-Cavilla, N. Tolia, E. de Lara, M. Satyanarayanan, and D. O'Hallaron. Interactive Resource-Intensive Applications Made Easy. In *Proceedings of the 8th ACM/IFIP/USENIX International Middleware Conference*, Newport Beach, CA, Nov. 2007.

[11] H. A. Lagar-Cavilla, J. Whitney, A. Scannell, S. M. Rumble, E. de Lara, M. Brudno, and M. Satyanarayanan. Impromptu Clusters for Near-Interactive Cloud-Based Services. Technical Report CSRG-TR578, Department of Computer Science, University of Toronto, May 2008.

[12] L. Lamport, R. Shostak, and M. Pease. The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982.

[13] L. Mummert, M. R. Ebling, and M. Satyanarayanan. Exploiting Weak Connectivity for Mobile File Access. In *Proceedings of the 15th ACM Symposium on Operating System Principles*, Copper Mountain, CO, December 1995.

[14] N. Ravi, J. Scott, and L. Iftode. Context-aware Battery Management for Mobile Phones. In *PerCom '08: Proceedings of the 6th Annual IEEE International Conference on Pervasive Computing and Communications*, Dallas, TX, March 2008.

[15] M. Satyanarayanan, B. Gilbert, M. Toups, N. Tolia, A. Surie, D. R. O'Hallaron, A. Wolbach, J. Harkes, A. Perrig, D. J. Farber, M. A. Kozuch, C. J. Helfrich, P. Nath, and H. A. Lagar-Cavilla. Pervasive Personal Computing in an Internet Suspend/Resume System. *IEEE Internet Computing*, 11(2):16–25, 2007.

[16] M. Satyanarayanan, M. Kozuch, C. Helfrich, and D. R. O'Hallaron. Towards Seamless Mobility on Pervasive Hardware. *Pervasive and Mobile Computing*, 1(2):157–189, June 2005.

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

[18] A. Surie, A. Perrig, M. Satyanarayanan, and D. J. Farber. Rapid Trust Establishment for Pervasive Personal Computing. *IEEE Pervasive Computing*, 6(4):24–30, 2007.

[19] N. Tolia, D. G. Andersen, and M. Satyanarayanan. Quantifying Interactive User Experience on Thin Clients. *Computer*, 39(3):46–52, 2006.

[20] N. Tolia, J. Harkes, M. Kozuch, and M. Satyanarayanan. Integrating Portable and Distributed Storage. In *FAST '04: Proceedings of the 3rd USENIX Conference on File and Storage Technologies*, San Francisco, CA, March 2004.