Safe Transient Use of Local Storage for VM-based Mobility

**Stephen Smaldone†, Jan Harkes, Liviu Iftode†
and Mahadev Satyanarayanan (†Rutgers University)**

March 2010
CMU-CS-10-110

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

This research was supported by the National Science Foundation (NSF) under grant numbers CNS-0509004 and CNS-0831268, and an IBM Open Collaborative Research grant. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily represent the views of the NSF, IBM, Rutgers University, or Carnegie Mellon University.

Keywords: Internet Suspend/Resume, ISR, OpenISR, virtual machines, portable storage, I/O performance, opportunistic resource usage

Abstract

This paper investigates the transient use of free local storage for improving performance in VM-based mobile computing systems. Many such systems boot from a portable storage device to create a "zero-install" environment on a computer that is borrowed for temporary use. Unfortunately, consumer-grade portable storage devices are optimized for capacity and cost rather than performance, which has been shown to be a critical factor in user experience on VM-based systems. We propose a zero-install solution called *TransPart* that uses the higher-performing local storage of borrowed hardware for performance-critical operations, yet preserves the integrity and privacy of data on that local storage. Our solution constructs a virtual storage device on demand by safely borrowing free disk blocks from the host's storage. We present the design, implementation, and evaluation of a TransPart prototype that requires no modifications to the software or hardware of a borrowed computer. Experimental results confirm that TransPart offers low overheads and startup cost, while improving user experience. We also suggest potential uses of the ideas underlying TransPart in other VM-based contexts.

1 Introduction

A growing number of systems exploit virtual machine (VM) technology to encapsulate and dynamically deliver user-specific state to a computer, thus enabling user mobility across hardware. SoulPad [6] is a well-known example of such a system, storing the entire state of a user VM in a bootable USB key. The Collective [8, 23] and the Internet Suspend/Resume system [16, 24] are two other examples of this genre of systems. Moka5 [1] is a commercial product in this space. While these systems differ considerably in their technical details, they share the top-level goal of decoupling a user's personal computing environment from a specific machine.

We use the term *SoulPad-like systems* to refer to this broad class of mobile computing systems. Their usage model is quite different from the ubiquitous email, Web access, and social networking capabilities provided by BlackBerries, iPhones, and other mobile devices. The strength of SoulPad-like systems lies in their ability to precisely, safely and rapidly re-create a user's Windows or Linux desktop environment as a thick client on borrowed hardware at any time and place. The user implicitly confirms the importance of these attributes to him by his choice of a SoulPad-like system over the lightweight alternatives mentioned above. Since user experience in a desktop environment is critically dependent on the I/O performance of its local storage system, we focus on that issue in this paper.

Use of borrowed hardware exposes a trust relationship between owner and borrower. Consider a situation where you borrow someone's hardware, use it, and then return it. Assuming that you have not tampered with the hardware, the owner is confident after the next power cycle that almost every component of his hardware (processor, memory, display, graphics accelerator, network interface, wireless interface, DVD drive, etc.) is back in its pristine state. The sole exception is, of course, the disk. Even if you had no malicious intent, it is possible that you inadvertently ran software that viewed disk contents that the owner considered private or, worse, modified or deleted important files. Encryption of disk blocks by the owner can ensure privacy in this context, but it cannot prevent mutilation or deletion of disk contents.

Can we improve upon this state of affairs? Is it possible for the owner, upon return of his hardware, to be confident that his disk is also in a pristine state? What does "pristine" mean in the context of a device with persistent state, such as a disk? We posit that the unstated owner intent when loaning hardware is *(a) to allow unrestricted use of free disk blocks so long as they remain free at the end of the loan period, but (b) to deny read and write access to allocated disk blocks during the loan period.* This is the intuitive meaning of "pristine" for a disk-like device. In other words, it is to provide the borrower with the illusion of a virtual disk that is composed solely of the free blocks of the real disk. Today, this owner intent is sustained purely through the good will and best efforts of the borrower. Our goal is to create a lightweight mechanism that enforces this implicit owner intent.

The concept of safely borrowing free disk blocks has broad applicability, well beyond the original motivation for this work. For ease of exposition, we defer discussion of these broader uses to Section 5.3, and concentrate on SoulPad-like usage models in the earlier sections. We make the following contributions in this paper:

- We identify a new storage-level concept that is well aligned with an emerging model of computing.
- We describe *TransPart,* a prototype design and implementation of this concept in Linux.
- We present experimental results to confirm that TransPart is a lightweight mechanism. Specifically, we show that TransPart's I/O performance overhead is low and its runtime startup cost is modest.
- We discuss broader uses of this new functionality.

2 Background and Related Work

2.1 SoulPad-like Systems

The SoulPad model of computing encapsulates a user's computing state within the confines of a VM. The entire state of this VM at a particular point in its execution is copied to a bootable USB storage device and physically transported by the user to a target machine. To resume the VM, the target machine is first booted from the USB storage device. The freshly-booted environment provides the VM monitor (VMM) support necessary to resume the suspended VM. Other than a compatible hardware architecture and the ability to boot from a USB storage device, there are no particular requirements on the target machine. Ubiquity is thus enhanced by eliminating the need to pre-install any software on target machines. Only the USB storage device needs to be configured with the correct boot image.

In other VM-based systems such as ISR, the Collective and Moka5, the transport of VM state occurs over the network from a server. The necessary boot image is typically pre-installed on target machines. However, variants of these systems also support the booting up of a target machine from a USB storage device to establish the correct VMM environment and then fetching VM state over the network. For ease of exposition, we refer to these variants as well as to the original SoulPad system as *SoulPad-like systems*. They all share three important attributes. First,

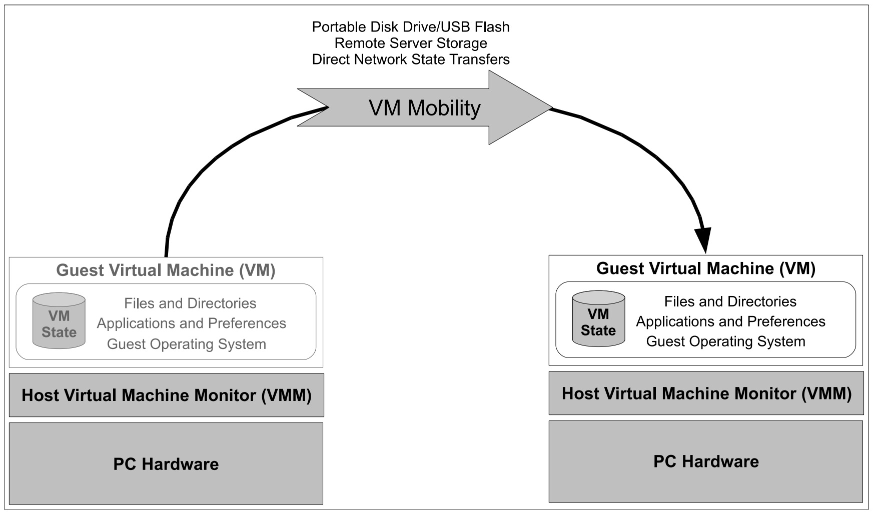

Portable Disk Drive/USB Flash
Remote Server Storage
Direct Network State Transfers
VM Mobility
Guest Virtual Machine (VM)
VM State
Files and Directories
Applications and Preferences
Guest Operating System
Host Virtual Machine Monitor (VMM)
PC Hardware
Guest Virtual Machine (VM)
VM State
Files and Directories
Applications and Preferences
Guest Operating System
Host Virtual Machine Monitor (VMM)
PC Hardware

Figure 1: Components of a SoulPad-like System

no pre-installed software is needed on a target machine. Second, since they are booted from a USB storage device, the I/O performance of that device typically limits overall system performance. And third, the VM-encapsulated user computing environment executes as a *guest* on top of the USB-booted *host.* The guest VM includes a user's files and directories, her applications and preferences, and even her operating system. Figure 1 illustrates the components of a SoulPad-like system.

2.2 I/O Limits on SoulPad-like Systems

As described in Section 2.1, SoulPad-like systems make no use of the local disk on a target machine — hence, the question of safely borrowing free disk blocks does not arise at all. Unfortunately, the realities of system performance intrude upon this simple picture. The difficulty arises from the fact that portable storage devices typically sacrifice I/O performance in order to optimize for size, weight, capacity, robustness and low cost. Their read and write performance are typically much worse than that of the local disk on a desktop. In addition, USB connectivity offers lower bandwidth than SATA connectivity.

			Size	Speed	Transfer Rate (MB/sec)	
Storage Device	Label	Type	(GB)	(RPM)	Read	Write
PNY Attache USB Flash Drive	SP-USB	USB	16	Flash	30.51 (1.01)	6.65 (0.29)
SanDisk MicroSD Card	SP-MSD	USB	8	Flash	16.03 (0.23)	11.9 (0.2)
Apple iPod	SP-IPOD	USB	20	4200	12.63 (0.18)	12.38 (0.17)
Internal SATA Drive	TP-SATA	SATA	250	7200	41.78 (2.48)	32.05 (1.14)

Transfer rate results are the mean of 5 measurements. Standard deviations are reported in parentheses.

Figure 2: Portable Storage Device Characteristics.

Figure 2 compares the measured I/O performance of three typical USB-attached storage devices (a SanDisk MicroSD card, a flash drive, and an Apple iPod) with that of that of a typical SATA desktop internal disk. We observe that the internal disk clearly outperforms all the portable devices in both read and write transfer rates.

Unfortunately, it is not just I/O-intensive applications that are affected by slow USB storage devices. Poor I/O performance of the boot device severely limits operating system performance, including basic functionality such as swapping and application launch. Our experiments (not shown here) indicate that both write and read performance of the boot device are important for satisfactory overall system performance. Enabling a SoulPad-like system to safely borrow free blocks from the local disk of a target machine can overcome this problem while preserving ubiquity and VM-based user state mobility.

2.3 Opportunistic Use of Free Storage

TransPart focuses on transient use of free local storage for improving performance in VM-based mobile computing systems. The opportunistic use of free blocks on a storage device was investigated by Cipar et al [9] in the *Trans-*

parent File System (TFS). Beyond this high-level similarity, however, there is considerable divergence of goals and mechanisms between TFS and TransPart.

TFS focuses on a class of Internet-scale peer-to-peer applications such as SETI@Home and Freenet, where individual users contribute their personal desktops to a free pool when the desktop is not being used. These applications have to be tolerant of weak persistence guarantees: TFS can unilaterally release space allocated to any file that is not currently open. In contrast, until the next reboot, TransPart offers the classic persistence guarantee of a file system: once created, a file remains in existence until it is explicitly deleted. This strict compatibility with existing file system guarantees allows unmodified SoulPad-like systems (and, by extension, guest OSes and guest applications within a locally-executing VM) to use TransPart. TFS and TransPart also differ in their interfaces, with TFS providing a file system interface while TransPart offers a block device interface that can support any desired file system. Files allocated via TFS need to be explicitly deleted. In contrast, no explicit cleanup is needed after use of TransPart: the virtual device simply disappears, and its storage reappears as free blocks of a mounted file system on the host OS.

Since TFS is intended to be used while the host OS is active, it is implemented through in-kernel modifications to the ext2 file system. In contrast, TransPart is used only when the host OS is inactive and its implementation completely avoid changes to the host file system. It is implemented in userspace in Python, using the same library as `fsck` to navigate ext2 and ext3 file system data structures. It would be relatively straightforward to extend TransPart to use free blocks in other file systems, such as ext4, NTFS, FAT-16, and FAT-32.

More broadly, opportunistic use of free storage has been extensively investigated by earlier work. As early as 2002, Beck et al introduced the term *logistical storage* to describe opportunistic use of free storage at Internet sites to reduce network transmissions [5]. Other work of this genre include IBMs Storage Tank [20] and Kang et al's work [12] on virtual storage provisioning. These efforts address storage opportunism at network scale, while TransPart targets individual storage devices. Also relevant is the scheduling opportunism of Lumb et al's work on using free blocks to improve disk scheduling by overlapping high-priority and low-priority disk workloads [19].

3 Design and Implementation

In this section, we present the design and implementation of TransPart. Throughout this paper we use a number of specific terms as we describe the model, design, and implementation. For clarity, we define these terms here.

When referring to the borrowing of PC hardware from one user by another, we refer to both users unambiguously as the *owner* and the *borrower*. We refer to the hardware and software components of the owner's PC as the *local* components. We also refer to the owner's PC as the *borrowed* PC.

The *host* is the execution environment within which a *guest virtual machine* or *VM* may execute. We use the terms *host*, *host virtual machine monitor*, or *VMM* interchangeably. Collectively, the host VMM and guest VMs define the borrower's software that executes on the owner's PC hardware. The guest VM executes a user's personal computing environment, and as shown in Figure 1 the guest VM encapsulates the user's PC state.

Finally, a *free* disk block is a disk block that satisfies one of two conditions: *(i)* it has not been *allocated* to a local file system or *(ii)* it has been allocated to a local file system, but is not currently *in-use* by the file system.

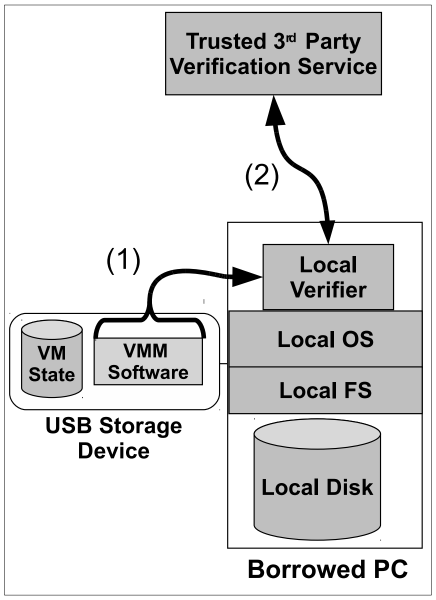
The design of TransPart is directed by a set of design goals. We list these below.

- *Borrow the free disk blocks of a local hard disk to create a virtual storage device for a guest execution environment.*
- *Protect the in-use local disk blocks from guest VM read and write operations.*
- *Achieve the first two goals without any modifications to local hardware, OS, or application software.*

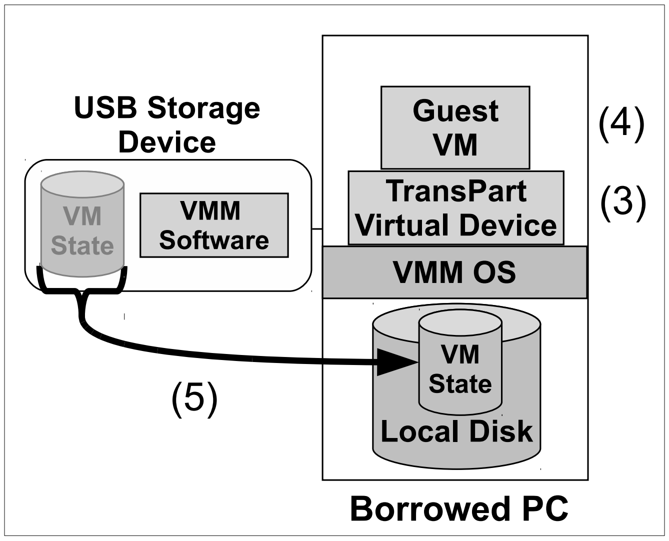
3.1 Design Assumptions

We have made two assumptions while designing TransPart. First, we assume that *no one physically tampers with any hardware involved*, including the portable storage device and the borrowed local PC hardware. We believe this to be a realistic assumption given the usage models envisaged. Most reasonable people would not loan out hardware storing precious data if they expected physical tampering to occur. Additionally, we believe it is reasonable to assume that in other scenarios where a user may borrow a PC, such as in an Internet Cafe, while kiosk computing, or when using a compute cluster, a level of surveillance (either human or video) would exist to deter such physical tampering from occurring.

Second, we assume that *the local OS does not execute concurrently with the borrower's software*. This enforces a strict isolation between the local and the borrower's software access to the local storage device.



(a) VMM Software Verification Phase.



(b) VMM Execution Phase.

A borrower attaches her USB disk to the borrowed PC *(1)*. The owner uses a local software verifier to contact a trusted 3rd party verification service, sends hashes of the VMM software to the service, verifying the VMM software *(2)*. During boot-up, TransPart discovers the free blocks on the local disk and creates a TransPart device *(3)*. The guest VM is launched *(4)*, using the TransPart device as its disk after VM State is transfered *(5)*.

Figure 3: TransPart Model.

3.2 TransPart Scenario

Figure 3 presents the TransPart model. In the remainder of this section, we describe the model through the use of the example shown in the figure. The numbers in parenthesis correspond to those in the figure. For the purposes of clarity, we refer to both a host VMM and any related software as *VMM software*.

Alice (the borrower) wishes to borrow her friend Bob's (the owner) PC to temporarily execute her SoulPad-like personal computing environment.[1] Since Bob is willing to loan his PC to Alice, she connects her portable SoulPad-like USB disk to Bob's PC *(1)*. This is the first step in the VMM Software Verification Phase of the TransPart model, as shown in Figure 3a.

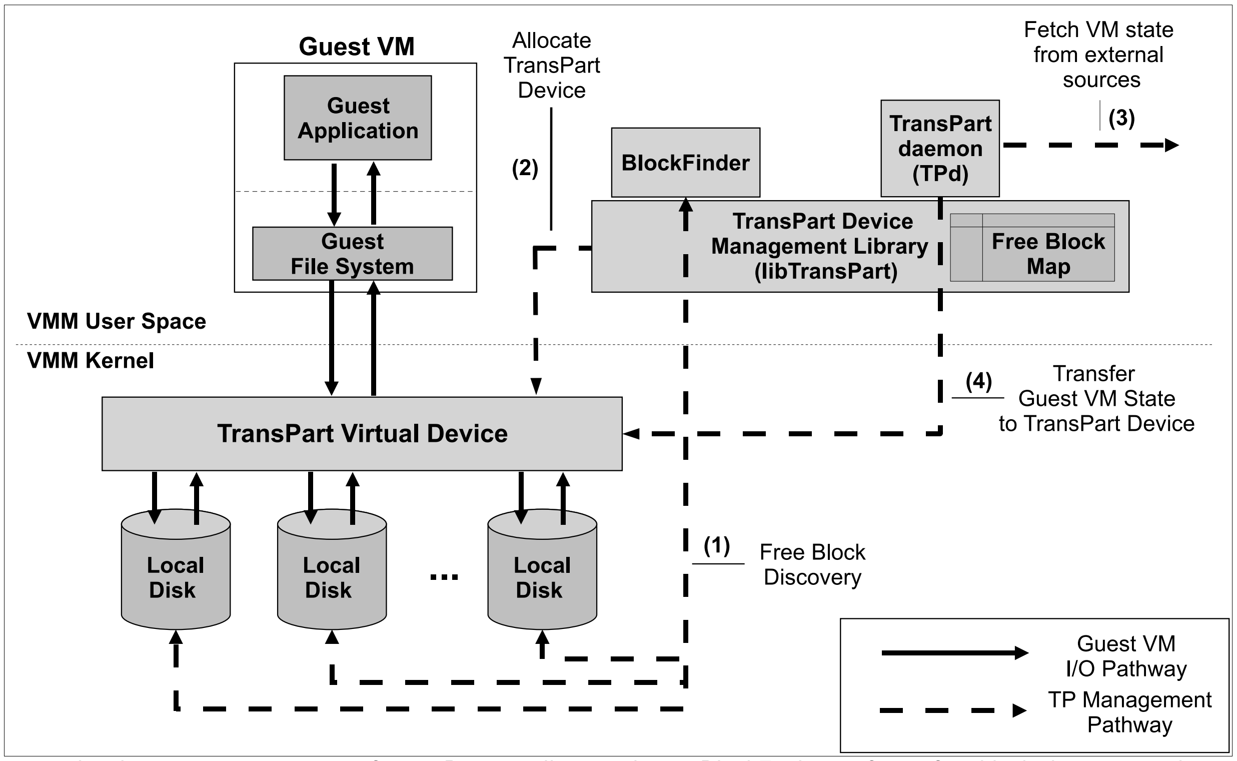
Next, Bob wants to verify that Alice will not access any of his private data (or any of his personal computing state). So, he opens his local web browser and downloads a software verifier, which performs a remote software attestation process [25, 22] by connecting to a "well-known" trusted 3rd party verification site over his Internet connection. The verifier calculates a secure hash of the borrower's VMM software stored on the connected USB and sends it to the 3rd party verification site. The verification site responds, verifying the authenticity of the hash by comparing it against "known-good" hashes. Since all user personalization occurs in a guest, the VMM software is easy to sanitize and we expect there to be a small number of such "known-good" hashes.

Validation of the VMM software by the 3rd party establishes a root of trust starting with the verification site down to the VMM software stored on Alice's USB disk (we refer the reader to [25, 17, 4, 22, 12, 29, 21] for the relevant details of these techniques.) The guest VM is not verified, since this is not required. Only the VMM needs to be trusted to ensure that Alice's guest VM may safely use the free blocks on his local disk; she will not have access to read or write any of Bob's in-use disk blocks.

To continue the process, Bob then hibernates the currently running execution of his PC environment and boots into the VMM software stored on Alice's USB key. By hibernating instead of rebooting, Bob saves his current execution/application state and can quickly recover when his machine is returned. At this point, we transition from the VMM Software Verification Phase (Figure 3a) to the VMM Execution Phase, as shown in Figure 3b.

In order to execute Alice's guest VM using the free blocks of the local disk as a transient storage location, the VMM must export the available free blocks on the local disk as a virtual device, and provide a device interface. While

[1]In an Internet kiosk scenario, the attendant on duty could play the role of Bob.



The three core components of TransPart are illustrated. *(1)* BlockFinder performs free block discovery and creates block allocation mappings. *(2)* TransPart device allocation occurs by the TransPart daemon through the TransPart Library API. *(3)* & *(4)* the TransPart daemon migrates guest VM state from external sources to newly allocated TransPart devices.

Figure 4: TransPart System Overview.

Alice's VMM boots up, TransPart discovers all available free blocks on the local disk and allocates an appropriately sized TransPart Virtual Disk Device, for Alice's guest VM, by aggregating the necessary amount of free blocks (3). The TransPart device provides Alice with access to the free blocks of the local disk, while protecting both the integrity and privacy of Bob's data.

Now that the VMM has finished booting and TransPart has allocated a virtual device for her guest VM, Alice wishes to start working within her personal computing environment. The remaining step in the process is for Alice's guest VM to be resumed *(4)*. As the guest VM executes, Alice's USB drive acts as a lookaside cache [30] providing access to the VM state *(5)*. Alice can now work, unhindered by the performance restrictions of her portable storage device. Note that the boot-up performance of the guest VM is not limited by the USB performance, since the TransPart device is allocated prior to guest execution, and the guest only executes from the higher-performing TransPart device.

There are a couple of additional features of the model to describe. We have assumed a SoulPad-like device in describing our model, where all guest VM state is stored on the USB disk. The TransPart model does not preclude guest VM state from being fetched from some other source, for example, an ISR server over the Internet, some other portable storage device carried in tandem with the USB device and locally attached, or even from a user's smart phone [26]. Finally, we describe demand-fetching of guest VM state in the description. It is also possible to copy the VM State in its entirety before resuming a guest VM, assuming the VM State is not prohibitively large.

3.3 Design Overview

In this section, we describe our design for the TransPart system. Recall that our design goal is to borrow the free blocks from a local disk, while protecting both the integrity and privacy of the existing local data. We accomplish this with a system design consisting of three core components: *(i)* the TransPart daemon, *(ii)* BlockFinder, and *(iii)* the TransPart Library. Figure 4 graphically depicts the TransPart system.

The TransPart daemon (TPd) is first activated during VMM boot-up, and remains active throughout the entire VMM session. Its role is to allocate TransPart devices as required by the guest VM, and to transfer guest VM state either on-demand or prior to guest VM execution (Section 3.4 presents additional details). The second core component, BlockFinder, performs free block discovery and mapping. This process is further described in Section 3.5. The third core component, the TransPart Library (libTransPart) provides all of the primitives and the API used by the other two components to discover free blocks on local disks, to allocate new TransPart devices, and to manage existing TransPart devices. The API interface is described in Section 3.6 below. Finally, the library provides access to the Free Block Map. This map stores the discovered free blocks and maintains the mapping of allocated free blocks to existing TransPart devices.

Figure 4 also illustrates the path I/O accesses take as they flow through the system. A file system access from an application, for example, is issued to the guest VM's virtual file system. This access is intercepted by the VMM and passed to the TransPart device, ultimately being serviced by one (or more) local disks. Of course, there are also multiple levels of caching involved, to improve I/O performance, but we ignore this for the sake of clarity.

Since TransPart devices only include free local disk blocks, there is no possible way a guest VM can either read or write to a block that is in-use by a local file system. Normally, a user would not need to log directly into the VMM and so such logins are disallowed by the TransPart security model, by default.

3.4 Device Allocation

Device allocation is handled automatically for the guest VM by the TransPart daemon during boot, prior to guest VM execution. The size for any given TransPart device is determined by the size of the guest VM state. In most cases, the device size can be substantially smaller than the actual full guest VM state size, since only portions of the state are fetched on-demand as needed for guest VM execution. This on-demand model does incur additional I/O performance overhead, but is mitigated by the fact that it is only incurred on the first access and can take advantage of spatial locality to prefetch additional state, ahead of the time it is needed. In the cases where on-demand guest VM state is disabled and all of the state copied prior to guest VM execution, TPd must allocate a TransPart device large enough to store the entire guest VM state.

To allocate a TransPart device, TPd utilizes the libTransPart calls, described later (in Section 3.6). Free block allocation occurs by choosing unallocated free blocks from the extents stored in the Free Block Map, and updating the map to change the status of the newly allocated free blocks.

3.5 Free Block Discovery and Mapping

Free block discovery occurs in two phases. In the first phase, BlockFinder enumerates the local devices searching for local storage devices, then it discovers the on-disk partitions. In the second phase, BlockFinder searches through each available partition on the local disk to discover and map the free blocks.

Most, if not all, modern file systems maintain a set of block allocation tables as meta-data on disk, for each formatted disk partition. For instance, the Linux ext2 [7] file system stores a block allocation bitmap in each block group that maintains the allocation status for each block in the associated block group. BlockFinder utilizes file system on-disk semantics to properly parse the file system metadata and discover free disk blocks, mapping them in the Free Block Map.

Normally, swap partitions are not used by TransPart, since host PC state resides there when hibernated. Should a user choose to discard that state, she would be free to also use any swap partitions with TransPart. Of course, this would force the host PC to perform a full reboot (rather than a simple wake-up from hibernation), after the TransPart user's session has completed.

The Free Block Map is not simply a one-to-one mapping of free blocks. Instead, it is built as a set of free block *extents*, or runs of consecutive free blocks. This minimizes the size of the mapping, and allows for more intelligent block allocation policies. We defer discussion of block allocation policies and the related topic of fragmentation until Section 5 of the paper.

3.6 TransPart Interface

The TransPart API consists of three library calls for TransPart device management:

- `tpalloc(size)`: this call attempts to allocate a TransPart device of the requested size.
- `tpfree(tp_device)`: this call attempts to free an existing TransPart device.
- `tprealloc(tp_device, size)`: this call attempts to resize an existing TransPart device to the requested size. This is used to grow or shrink TransPart devices.

The library also exports a set of calls for free block discovery and free block mapping operations:

- `blkfind(device, count)`: search on a device and find the requested number of free blocks.

- `blkmap_insert(device, block_number, tp_device, tpblock_number)`: insert a free block mapping into the Free Block Map managed by libTransPart.
- `blkmap_remove(device, block_number, tp_device, tpblock_number)`: remove a free block mapping from the Free Block Map managed by libTransPart.
- `blkmap_update(device, block_number, tp_device, tpblock_number)`: modify a free block mapping currently stored in the Free Block Map managed by libTransPart.
- `blkmap_query(tp_device, tpblock_number)`: query the Free Block Map to determine the status of an existing free block. The status may be either allocated or not.

3.7 Implementation

In this section, we describe the TransPart prototype. For this, we have implemented the TransPart daemon, BlockFinder, and the TransPart Library components. Together, the components are installed on the portable storage device and are executed during various stages of boot-up of the host VMM from the portable storage device. No modifications were made to the host, guest or local software.

The TransPart Library is implemented in C. The current version supports both ext2 and ext3 file systems. For low-level semantic access to ext2/3 file systems, we used the Linux ext2fs libraries utilized by the standard mke2fs, e2fsck, and other utilities. BlockFinder and the TransPart daemon are both implemented in Python version 2.6. We store the Free Block Map as a table in a SQLite 3.6.10 database. To create in-kernel TransPart devices, we leverage the existing Linux Device-mapper [11]. In so doing, we take advantage of all the performance benefits of the mature Linux kernel code supporting logical volume management.

In order to optimize performance of the various operations carried out on the Free Block Map, TransPart first allocates a small TransPart device to hold the system files related to the Free Block Map. By doing this, TransPart is also able to take advantage of the performance of the local internal disk, rather than being constrained to the portable storage device.

4 Evaluation

Our experimental evaluation of the TransPart prototype addresses two questions:

- How much does guest VM application and OS performance improve due to TransPart?
- How much additional time is added during start-up due to TransPart?

Together, these questions allow us to explore both the benefits and costs of the TransPart system. Our goal in this study is to demonstrate the benefits of the TransPart system towards improving the user experience within the framework of VM-encapsulated mobile computing. We use OpenISR version 0.9.6 as the SoulPad-like system for all experiments in the evaluation. To closely approximate SoulPad with OpenISR and to eliminate the effects of network bandwidth all experiments are carried out with fully hoarded guest VM state on a disconnected machine.

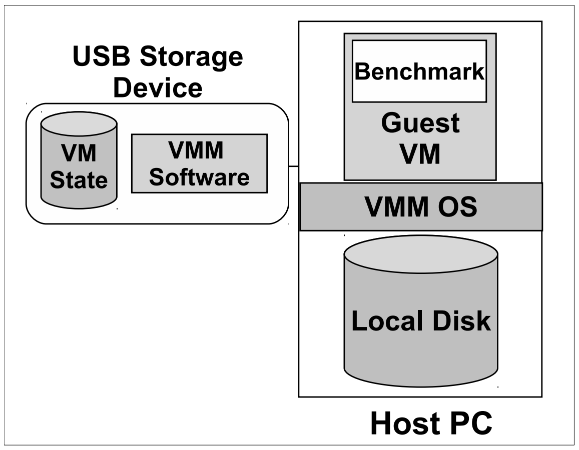
In this section, we describe our experimental methodology (Section 4.1), then present two sets of experimental results. First, we report the results pertaining to TransPart benefits (Section 4.2) and then we present the results pertaining to TransPart costs (Section 4.3).

4.1 Experimental Methodology

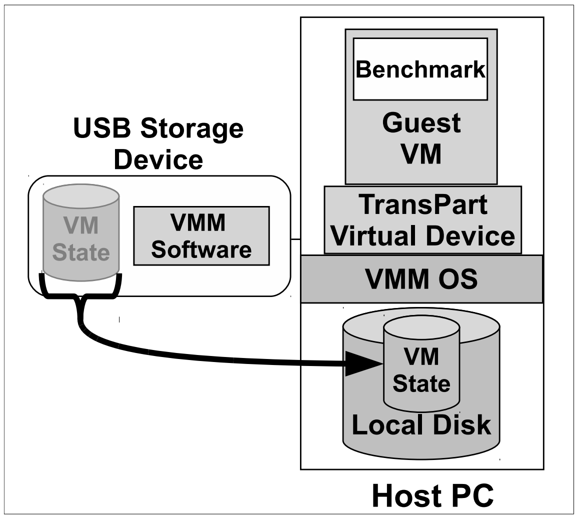
For all experiments we use a Dell Optiplex 755 PC with a 2.33 GHz Core 2 Duo CPU, 3 GB of RAM, a 250 GB Serial ATA disk at 7200 RPM, and support for Hi-Speed USB. This PC acts as the host PC, which boots the portable USB-based VMM and executes OpenISR and the TransPart software. We run a guest VM (ISR parcel) configured to use 512 MB of memory and a 4 GB virtual disk. Inside the guest VM we run the Fedora Linux 10 OS (Linux kernel version 2.6.27). For all VM storage devices, we use the Linux ext2 file system.

Figure 5 shows a diagram of the individual components of our experimental configuration. For the SoulPad-like case (Figure 5a), all guest VM state is accessed directly from the attached portable USB device during guest VM execution. For the TransPart case (Figure 5b), guest VM state is first transferred to the host PC's local SATA disk and accessed from there during guest VM execution.

We select a range of portable storage devices to hold the guest virtual machine state. Figure 2 list the devices and their relevant performance characteristics. We also include a row for the internal hard disk used in this study (Internal SATA Drive). Transfer rates listed are the sequential read and write performance for each device. The *Label* column specifies the short-hand notation used to refer to a specific device throughout this evaluation. Those device labels that



(a) SoulPad-like Configuration.



(b) TransPart Configuration.

SoulPad-like configuration is shown on top and TransPart configuration is shown on bottom. Benchmarks are executed within guest VM on host PC. For TransPart, guest VM state is first transferred to host PC disk, while guest VM state is accessed directly from USB for SoulPad-like cases.

Figure 5: Experimental Configuration.

include the *SP-* prefix refer to configurations that conform to the SoulPad-like model, while *TP-SATA* signifies the configuration that utilizes TransPart.

To understand the performance improvements provided by TransPart, we execute six different benchmarks that are representative of the range of tasks for a typical PC user. Each benchmark is executed inside the guest VM and represents a different workload focus. The Postmark [15] benchmark measures the performance of a small I/O and metadata intensive workload across a set of files (Section 4.2.1). We execute a custom benchmark to measure the launch latency of a variety of common desktop applications (Section 4.2.2). The next benchmark we run is a modified Andrew benchmark [13], which emulates a software development workload (Section 4.2.3). To determine user perceived desktop application performance, we execute a custom office productivity benchmark consisting of common operations within the OpenOffice.org [28] Writer (word processor) application (Section 4.2.4). We execute a software installation benchmark that emulates the task of installing a set of software packages within the guest VM (Section 4.2.5). Finally, the Bonnie++ [10] benchmark tests the performance of a set of simple file system accesses to a single large file (Section 4.2.6).

To understand the cost of TransPart, we measure the time required to discover and map the free blocks available on the internal hard disk, and we measure the latency of allocating a TransPart device to a guest VM (Section 4.3.1). Combined, these operations determine the overhead of using the TransPart system. Finally, we evaluate the real impact of these costs by measuring the individual start-up times for each of the different portable USB devices with and without TransPart (Section 4.3.2).

All benchmarks in the evaluation are run a minimum of 5 times, and we report the average results over the runs along with standard deviations. Before each new run of a benchmark, we restart the guest VM. For the TransPart cases, we boot off the SP-USB portable device, unless otherwise noted.

4.2 TransPart Performance Benefits

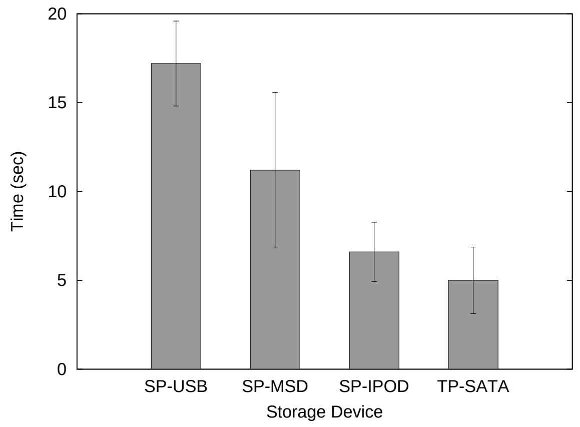
4.2.1 Postmark

In this experiment, we execute the Postmark [15] benchmark (version 1.51). Postmark was created in 1997 and its workload is characterized by many operations on short-lived, small files, and consists of many data and meta-data operations. Since it does not attempt to approximate application processing, it performs very little CPU activity, and is I/O-intensive by design. FIgure 6 lists the Postmark configuration that was used in this experiment, based on the Traeger et al [31] and Soules et al [27] studies.

Figure 7 presents the results of this experiment. Each bar in the figure represents one of the four devices included in the evaluation and reports the completion time for the Postmark benchmark. Since Postmark includes a mixture of operations (read, write, create, and remove), it executes a more diverse set of both data and meta-data operations. As

Postmark Parameter	
Number of Files	5000
Number of Subdirectories	50
File Sizes	512 bytes - 10 KB
Number of Transactions	20000
Operation Ratios	equal
Read Size	4 KB
Write Size	4 KB
Buffered I/O	no

Figure 6: Postmark Configuration.



The Postmark v1.51 benchmark is executed with the configuration described in Figure 6 for each device case. Results reported represent the benchmark completion time in minutes and are the averages of 5 runs.

Figure 7: Postmark Results.

such, it attempts to generate a realistic workload under test. The SP-IPOD and TransPart (TP-SATA) cases outperform both flash drive cases by up to a factor of 3, while the TransPart (TP-SATA) case exhibits a 30% performance improvement over the SP-IPOD case.

4.2.2 Application Launch Latency

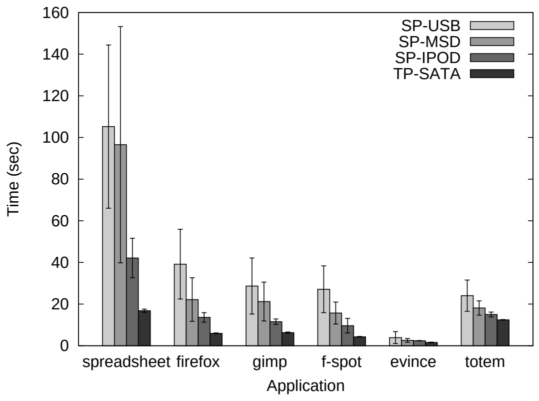
An important indicator of user experience is application launch latency [18]. In this experiment, we are specifically interested in quantifying the time it takes for a typical desktop application to be available for use by a user, once the user has chosen to launch it. This is quite important, as poor performance in this area tends to evoke a visceral reaction by a user, tainting her perception of overall desktop performance from that point onward. To evaluate this, we launch a set of six commonly used applications, measuring the application launch latency within a guest VM for the three portable USB drives and TransPart. We use a custom script to launch and measure the launch latency for *(i)* the OpenOffice.org spreadsheet, *(ii)* the Firefox web browser, *(iii)* the Gimp image manipulation tool, *(iv)* the F-Spot photo editing tool, *(v)* the Evince document viewer, and *(vi)* the Totem multimedia (audio/video) player. Figure 8 presents the results of this experiment.

We make three observations, from the figure. First and most important, TransPart improves performance and reduces variability (shown by the size of the errorbars) for all applications tested. These improvements are most evident when focussing on the spreadsheet and Firefox cases, but TransPart also exhibits the best performance for all cases. When compared to the SP-USB, SP-MSD, and SP-IPOD cases, TransPart exhibits up to a factor of 10, 7, and 4 performance improvements, respectively. Second, application launch latencies for the SP-USB and SP-MSD cases are both high and variable, likely beyond the range of tolerance for a typical user. Finally, the latencies for the SP-IPOD case are still noticeably large to a user, but in a more tolerable range. The latencies also vary less for SP-IPOD than the SP-USB and SP-MSD cases.

4.2.3 Andrew Benchmark

We execute a modified Andrew benchmark [13] against the Linux kernel sources (version 2.6.31.6) such that it will exceed the memory capacity of the VM under test. The benchmark consists of five phases: *(1)* recursively create subdirectories, *(2)* copy the source tree, *(3)* query the status of each source file in the tree without accessing data, *(4)* read the data of each source file, and *(5)* compile and link the sources. The results are presented in Figure 9.

For each phase, we report the completion time (in seconds) for each of the four portable USB devices included in the evaluation. From the table, we observe that TransPart substantially improves the performance in all cases. The result is a 109% improvement for the SP-USB case, 75% over SP-MSD, and 5% over SP-IPOD. When examining the results for the individual benchmark phases, we observe that in all but two phases (1 and 4) both rotational disk devices (SP-IPOD and TP-SATA) substantially outperform the flash devices (SP-USB and SP-MSD).



Application launch latency is measured for 6 common desktop applications. All results are times in seconds and represent the averages of 5 runs.

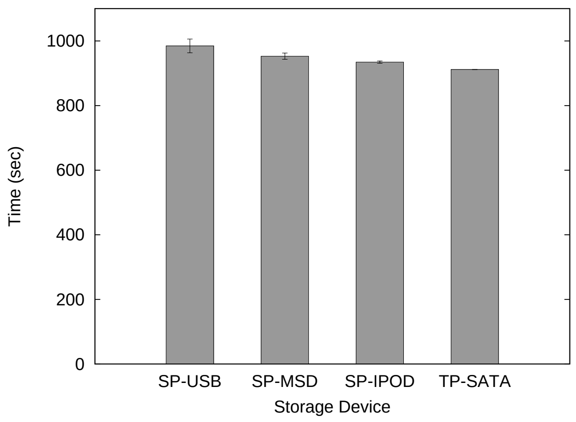
Figure 8: Application Launch Latency.

Benchmark Phase	SP-USB	SP-MSD	SP-IPOD	TP-SATA
Overall	5904 (82)	4970 (179)	2947 (20)	2820 (26)
1	34 (4)	39 (29)	52 (3)	14 (6)
2	782 (22)	616 (146)	189 (14)	164 (28)
3	43 (10)	14 (7)	9 (2)	9 (2)
4	57 (17)	137 (11)	16 (2)	50 (15)
5	4987 (92)	4506 (834)	2682 (15)	2583 (15)

(Storage Device spans SP-USB through TP-SATA.)

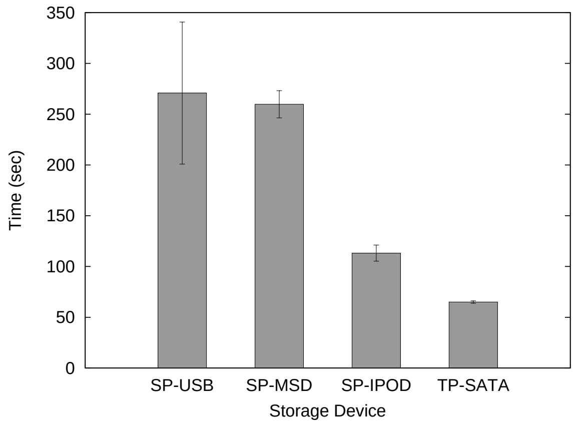
A Modified Andrew benchmark is run and completion results are reported for overall time and for individual phases: (1) subdirectory creation, (2) source tree copy, (3) file status check, (4) file data access, and (5) compilation and linking). Results are the mean of 5 runs and standard deviations are reports in parentheses.

Figure 9: Modified Andrew Benchmark Results.



Custom office productivity benchmark is run and measurements are taken. Results reported represent completion times in seconds and are the averages of 5 runs.

Figure 10: Office Productivity Benchmark Results.



Custom software installation benchmark is run and completion time results are reported in seconds. All results are the averages of 5 runs.

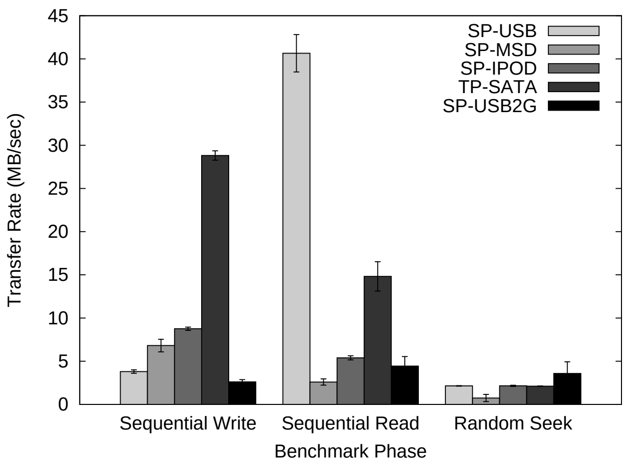
Figure 11: Software Installation Benchmark Results.

4.2.4 Office Productivity

The goal of this experiment is to measure the user experience for a user that is performing a mixture of typical operations in a suite of office productivity tools. To accomplish this, we created a custom benchmark that performs word processing tasks using the OpenOffice.org Writer application. The benchmark consists of a set of Python scripts that interact with the open API provided by the OpenOffice.org suite [28]. The minimum running time of the benchmark is 15 minutes.

Since the rate of document content generation will ultimately influence the disk I/O rate during the benchmark due to periodic file saving, we introduce enough think time to impose a maximum content generation rate of 30 words per minute. We choose 30 words per minute based upon a study [14] that shows the average content creation rate for an average typist.

From the results shown in Figure 10, we observe a modest overall performance improvement between 22 and 73 seconds (from 2% to 8%) when comparing the SP cases to TransPart. These improvements are due to a reduction of application start-up time and reduced delays in application interactive responsiveness during the benchmark. We verified the impact of TransPart on application responsiveness qualitatively, and found the interactive performance of the word processing application with TransPart to closely resemble that of the local PC. For the SP cases, we experienced additional and varied delays in application responsiveness.



Each bar represents the average transfer rate in MB/sec for a particular specific phase of the Bonnie++ v1.03 benchmark. Results are averages of 5 runs.

Figure 12: Bonnie++ Results.

Request Size (GB)	Time (sec)
0	12.1
1	12.9
10	**20.1**
25	31.7
50	54.5
100	103.9
150	154.4
200	204.3

Free block discovery latency is measured as request sizes vary from 0 GB to 200 GB; 10 GB represents the typical VM state size. Results represent discovery completion times (sec), are the averages of 5 runs, and all standard deviations are below 2%.

Figure 13: Free Block Discovery Time.

4.2.5 Software Installation

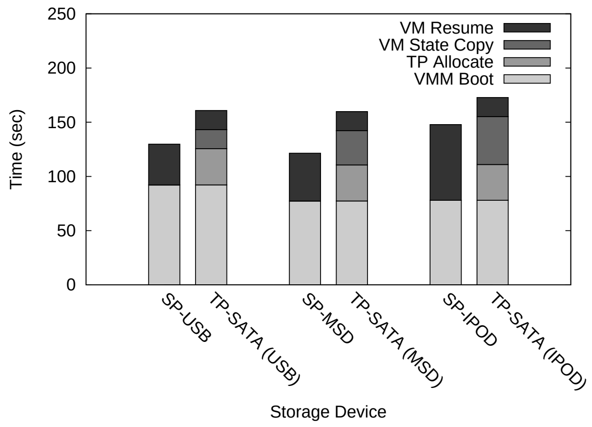
While software installation may not seem typical of a mobile user, one can imagine a mobile user on an extended trip needing to apply software updates to remove recently discovered security vulnerabilities, during her trip. We evaluate the effectiveness of TransPart for software installs by simulating a user performing software installation tasks using the well-known YUM [3] open-source package-management utility. To do so, we execute the install of a set of software packages consisting of a commonly used, open-source, office productivity suite (OpenOffice.org) and various related dependency packages. In total, 45 individual packages are installed and the combined size of the package files is 151 MB resulting in an additional 350 MB of disk space utilization once installed. To eliminate the effects of network I/O performance on this benchmark, we download the required packages to the guest VM ahead of time. Therefore, the benchmark results only measure the time to perform all local software installation operations. Figure 11 presents the results of this experiment.

From the figure, we observe that TransPart substantially outperforms all portable USB device cases. Compared to both SP-USB and SP-MSD, TransPart exhibits improvement by over a factor of 4, and results in a 74% improvement over SP-IPOD.

4.2.6 Bonnie++

Bonnie++ [10] was created in 2000 as an improvement over its predecessor Bonnie. The benchmark's workload is composed of low-level I/O performance tests, and is not necessarily typical of most mobile user tasks, but we include it in the interests of completeness. In this experiment, we report the results of the following tests over all device cases: *(i)* sequential read tests, *(ii)* sequential write tests, and *(iii)* random seek tests, using Bonnie++ version 1.0.3. Figure 12 presents the results of the Bonnie++ experiment, consisting of the data transfer rates (in MB/sec) for each of the four devices included in the evaluation.

From the figure, we observe that the TransPart case (TP-SATA) substantially outperforms all other cases for sequential write performance. For the random seek case, the SP-USB, SP-IPOD, and TP-SATA results are similar, while SP-MSD exhibits lower performance. For sequential reads we observe that TP-SATA outperforms the SP-MSD and SP-IPOD cases. Also, we observe an anomaly for SP-USB read performance. Investigating more deeply, we find that this is due to a combination of the asymmetric read and write performance of the SP-USB device (see Figure 2), two layers of disk buffering occurring in the system (guest VM and VMM buffer caches), and the Bonnie++ benchmark's choice of test file size (1 GB). We ran Bonnie++ using a larger test file (2 GB) and present the results in Figure 12 as the SP-USB2GB case. From these results, we observe that the anomaly disappears. We also ran the SP-MSD, SP-IPOD, and TP-SATA cases with the 2 GB test file, and did not observe any difference from the original results. For clarity, we choose not report them.



Total start-up time is reported for each portable USB device and presented as stacked bars composed of the *VMM Boot* and *VM Resume* components for all cases, and *TP Allocate* and *VM State Copy* components for the TransPart cases. Results represent completion times in seconds and are averages of 5 runs. All standard deviations are less than 6%.

Figure 14: Start-up Time.

4.3 TransPart Start-up Costs

4.3.1 Free Block Discovery and Allocation Time

This experiment measures the time taken by BlockFinder to perform free block discovery on the internal hard disk within the host virtual machine monitor (VMM). To characterize the effects of disk size and requested TransPart device size on the latency, we vary the requested TransPart device size from 0 GB up to the total available free blocks of the hard disk (200 GB).

Figure 13 presents the time (in seconds) for BlockFinder to perform free block discovery on the host's internal SATA drive as the requested size of the TransPart device is varied from 0 GB to 200 GB. We highlight the 10 GB case, since it represents the typical guest VM state size as seen in SoulPad-like systems and ISR. Additionally, the size of an average low-cost portable USB storage device is currently on the order of tens of GB. From the figure, the time to discover enough free blocks to allocate a 10 GB TransPart device is 20.1 seconds. Furthermore, discovery latencies for devices in the tens of GB range seem quite tolerable ($<$ 1 min for up to 50 GB).

From the figure, we observe three additional interesting points regarding the measured latency. First, the 0 GB case represents the constant, minimal overhead of reading the free block bitmap from the SATA disk file system. Second, as we increase the request size, we incur increasing overheads to calculate the free block extents and store them in the free block map. Finally, the 200 GB case represents a request for the remaining free available blocks on the SATA drive, and exhibits the maximum possible overhead for the SATA drive used in this experiment.

Even though discovery for large ($>$ 50 GB) request sizes takes on the order of minutes, free block discovery is only executed once per VMM boot and can be performed concurrently while the host VMM is completing its initialization. As mentioned in Section 3, since discovery occurs in a single pass, its performance scales linearly with request size. In the worst case (largest possible request size), free block discovery performance is determined by the time to scan the entire block bitmap for the internal hard disk, create the free block extents, and insert them in the Free Block Map. In our experimental setup, this corresponds to the 200 GB result in Figure 13.

We also measured the time taken by BlockFinder to allocate the TransPart device as the requested device size was varied from 1 GB up to the total available free blocks of the hard disk (200 GB). In all cases, the measured device allocation latency was less than 2.2 seconds.

4.3.2 Total Start-up Time

This experiment measures the total start-up time taken to boot the borrowed PC from the different portable USB drives and resume the guest VM parcel. We measure the individual components of start-up time for each portable USB drive and the TransPart case, and present the results in Figure 14. We group the bars into three groups based upon portable USB device. For example, the SP-USB and TP-SATA (USB) are both booted from the SP-USB drive. In the former, the guest VM is resumed from the SP-USB drive, while in the latter, the guest VM state is first copied to a TransPart device. Therefore, the cost of using TransPart is the difference between the two bars in each group.

From Figure 14 we observe the net increase in total start-up time for TransPart to be in the range of 25-39 seconds. Considering the benefits of TransPart as exhibited earlier by the performance results presented in Section 4.2, the additional start-up costs due to TransPart are completely acceptable. In fact, the improvements in application launch latency alone more than make up for this additional start-up time.

Each stacked bar in the figure is composed of the time to boot the VMM from the portable USB drive (*VMM Boot*) and the time to resume the guest VM (*VM Resume*). For the TransPart cases, the free block discovery and allocation time (*TP Allocate)*, and the time to copy the guest VM state to the TP device (*VM State Copy*) are also presented. By comparing the TP-SATA cases with their respective non-TP cases, we observe a trade-off between the reduction in VM Resume time and the costs of TP Allocation and VM State Copy. Although VM Resume Time is reduced by 20-53 seconds when compared to respective non-TP cases, this reduction is offset by 51-77 seconds of additional start-up costs (TP Allocate and VM State Copy), accounting for the net increase in start-up time.

4.4 Summary

To summarize, we draw four conclusions from the results of our evaluation. First, under data-intensive workloads, TransPart performs as well as or better than most non-TransPart configurations. In only one specific case (Bonnie++ sequential reads), does one of the non-TransPart configurations (SP-USB) perform better than TransPart. Second, for mixed I/O workloads composed of both data and meta-data intensive operations, TransPart clearly outperforms all non-TransPart configurations. Third, even under conditions of light I/O workload, typical user interactive applications benefit from TransPart through reduced application launch latency and improved responsiveness. Finally, TransPart improves software update and installation performance to a more tolerable level, enabling more frequent software updates for mobile users.

5 Discussion

5.1 Reasonable Safety Assumptions

The primary goal of this work is to enable one user to borrow another user's PC, execute their personal computing environment utilizing the full available performance, without compromising the owner's privacy or the integrity of her data. Although it might be possible to provide a stronger security model by relaxing some of our design constraints (i.e., allowing OS modifications), we believe that our model provides a reasonable balance of safety for the owner and usability for both parties.

Although we have primarily focussed on protecting the owner's data, the proposed solution must also guarantee the guest VM's privacy with respect to the host. TransPart achieves this goal at two levels: *(i)* the host OS is always suspended during guest VM execution and *(ii)* all guest VM data is encrypted while it resides on the local disk, and is overwritten by TransPart once guest VM execution has been suspended and any modifications have been synchronized back to the portable USB storage device.

In general, we believe that the TransPart solution provides "reasonable" safety for both the guest VM and the host. Our system is analogous, in a way, to a handshake between friends. We do not expect a friend to carry a dagger up her sleeve, but it never hurts to check.

5.2 Potential Improvements

Multiple Guest VMs. Although TransPart's primary rationale corresponds to a single guest scenario, the proposed architecture can easily be extended to support multiple guest VMs. It is possible for a user to carry multiple portable USB storage devices with them, or have more than one guest VM on a single device. Consider a user who maintains both a work and a personal guest VM, to isolate the computing environments of those two aspects of her life. As most professional "multitaskers", she often wishes to execute both guest VMs at the same time, to better interleave the completion of her work and personal life tasks. To support this and other similar scenarios, TransPart must be able to isolate multiple concurrently executing guest VMs while still providing full access to the free blocks of the borrowed host.

Startup Cost vs. Session Length. For TransPart to be effective, its startup cost should be recaptured by the gains in I/O execution performance for I/O operations. This seems to imply that the usefulness of TransPart for any user session should be determined *a priori* by the potential gains in performance brought about by the type and duration of tasks a user intends to do during that session. Although, a simple calculation can be made to get a rough estimate of the potential performance gains for any session, it turns out to be unnecessary, under most circumstances. The experimental evidence presented earlier in Figure 8 indicates that even for relatively low I/O applications (such as a web browser or word processor), the reduction in application launch latency time due to TransPart, can be sufficient to offset the typical startup cost (20 sec startup cost for a 10 GB TransPart device vs. 30-90 sec latency reduction).

Furthermore, the reduction in application execution time as shown in Figure 10 provides further evidence (up to 73 sec during a 15 min word processing session) to support this claim.

Free Block Allocation Policies. As previously mentioned in Section 3.5, our design allows for a number of possible approaches for allocation of free blocks to TransPart devices. In fact, since there are already multiple layers of indirection starting with the guest OS down to the PC hardware, there may be some benefit to consider the entire software stack in determining the correct block allocation approach for any particular guest VM. With TransPart, since each guest VM is allocated a set of blocks dynamically, it is possible to customize block allocation to match the needs of each guest. We intend to explore the investigation of free block allocation policies within TransPart as future work.

Effects of Disk Fragmentation. The purpose of TransPart is to improve the performance of a user's computing experience. One issue that may affect disk I/O performance is fragmentation. Fragmentation affects TransPart in two ways.

First, during free block discovery, disk fragmentation may lengthen the time to discover a requested size of blocks, but the worst case time is still ultimately bounded by the time it takes to discover all of the free blocks on a disk. In this study, we have not evaluated the effects of disk fragmentation due to the difficulties presented in accurately simulating the fragmentation of a disk, but we do consider it an area for future study.

The second effect is exposed when choosing the free blocks to present to a guest VM. As the level of existing local disk fragmentation increases, the possible options for free block allocation are reduced. This is mitigated, though, by the increasing trends toward caching and indirection in modern disk hardware. The effects of these trends are already evident in the reduction in complexity of more recent file system block layout schemes (ext2/3) versus earlier efforts (FFS).

5.3 Beyond Safe Borrowing for Mobility

In this paper, we have introduced the concept of safe borrowing of blocks, but we have done so in the context of VM-based personal mobile computing. We believe that the safe borrowing concept has broader application and can be extended, beyond the mobile computing domain.

We envisage the use of safe borrowing in the context of VM server farms, as an efficient mechanism for ad-hoc, temporary VM provisioning. Today, as VM servers (Virtual Servers) become more densely concentrated on individual high-end hardware platforms, maintenance of the hardware becomes a difficult task. The criticality of a specific piece of hardware is multiplied by the number of critical VMs that must maintain Five 9's availability. This poses a particularly onerous task for administrators as they attempt to schedule downtime for each individual VM server hardware maintenance event. Allowing a VM to become *transient*, temporarily move to another existing platform, and share all existing resources by borrowing free disk blocks from existing *permanent* VM residents has the potential to reduce all of the overheads originally introduced by the VM migration model at scale. Implementing this requires a relaxation of one of the design constraints introduced earlier, though, since under this context multiple concurrently executing guest VMs will need to share access to the same set of free blocks.

Another direction for future work involves exploring the design space of where to place the safe borrowing primitives. To this point, we have only considered positioning the safe borrowing primitives within the VMM layer. In the future, we plan to consider alternative placements. One obvious place would be to explore the placement of safe borrowing within the disk. Moving from the VMM to the disk simplifies some aspects of the design, since the disk continuously receives all requests. Although the recent addition of the ATA TRIM command exposes some file system semantics to the disk, placing the safe borrowing primitives within the disk interface is still challenging since it requires careful consideration of precisely which additional semantics would be required to support it.

Finally, we believe that the idea of using the free blocks of the local disk to provide transient virtual storage for guest VMs can be broadened to include virtualization of the entire storage layer. Emerging fast backup technologies, such as the solutions provided by Data Domain [2], enable an extended memory hierarchy in which the *online* disk storage layer can be viewed as a cache of the *near-line* backup storage layer. In this way, the backup storage layer could act as a backing store for the disk storage layer. We plan to explore this idea as future work.

6 Conclusion

We began this paper with the down-to-earth example of one user borrowing hardware from another user. We identified the trust assumption, relative to disk state, that is implicit in this simple human transaction. The trust assumption is that the borrower may use free disk blocks without any restrictions, but may not read or write any other disk blocks. We observed that this trust assumption is directly relevant to a new computing model of growing importance in which user execution state is transported between machines using portable storage or the Internet.

We have described the design and implementation of TransPart, a lightweight mechanism for enforcing this trust assumption. This mechanism establishes a root of trust by verifying the signature of a minimal software environment on portable storage, and then booting that environment. A TransPart device is then created, consisting solely of free disk blocks. The boot image of the user's execution environment is then bound to that device. Experimental results from our prototype confirm that the I/O performance experienced by the user is determined by the borrowed hardware rather than by the portable device containing the root of trust. Finally, although motivated by the relatively narrow usage scenario of borrowing hardware, we show that TransPart has the potential to be useful in many other usage scenarios.

References

[1] MokaFive home page. `http://www.mokafive.com`, 2005-2008.

[2] Data Domain, Inc. - Home Page. `http://www.datadomain.com`, 2009.

[3] YUM - Yellowdog Updater Modified. YUM Package Manager Project Home Page. `http://yum.baseurl.org`, 2009.

[4] ARBAUGH, W. A., FARBER, D. J., AND SMITH, J. M. A secure and reliable bootstrap architecture. In *SP '97: Proceedings of the 1997 IEEE Symposium on Security and Privacy* (Washington, DC, USA, 1997), IEEE Computer Society, p. 65.

[5] BECK, M., MOORE, T., PLANK, J.S. An End-to-End Approach to Globally Scalable Network Storage. In *Proceedings of the ACM SIGCOMM Conference* (Pittsburgh, PA, 2002).

[6] CÁCERES, R., CARTER, C., NARAYANASWAMI, C., AND RAGHUNATH, M. Reincarnating PCs with Portable SoulPads. In *MobiSys '05: Proceedings of the 3rd International Conference on Mobile Systems, Applications, and Services* (Seattle, WA, 2005).

[7] CARD, R., TSO, T., AND TWEEDIE, S. Design and implementation of the second extended filesystem. In *Proceedings of the First Dutch International Symposium on Linux* (1994).

[8] CHANDRA, R., ZELDOVICH, N., SAPUNTZAKIS, C., AND LAM, M. The Collective: A Cache-Based System Management Architecture. In *Proceedings of the Second Symposium on Networked Systems Design and Implementation* (May 2005).

[9] CIPAR, J., CORNER, M. D., AND BERGER, E. D. Contributing Storage Using the Transparent File System. *ACM Transactions on Storage 3*, 3 (2007).

[10] COKER, R. Bonnie++ home page. `http://www.coker.com.au/bonnie++`, 2001.

[11] DEVICE-MAPPER, L. Device-Mapper Resource Page. `http://sources.redhat.com/dm/`, 2001.

[12] GARRISS, S., CÁCERES, R., BERGER, S., SAILER, R., VAN DOORN, L., AND ZHANG, X. Trustworthy and personalized computing on public kiosks. In *MobiSys '08: Proceeding of the 6th international conference on Mobile systems, applications, and services* (New York, NY, USA, 2008), ACM, pp. 199–210.

[13] HOWARD, J. H., KAZAR, M. L., MENEES, S. G., NICHOLS, D. A., SATYANARAYANAN, M., SIDEBOTHAM, R. N., AND WEST, M. J. Scale and performance in a distributed file system. *ACM Trans. Comput. Syst. 6*, 1 (1988), 51–81.

[14] KARAT, C.-M., HALVERSON, C., HORN, D., AND KARAT, J. Patterns of entry and correction in large vocabulary continuous speech recognition systems. In *CHI '99: Proceedings of the SIGCHI conference on Human factors in computing systems* (New York, NY, USA, 1999), ACM, pp. 568–575.

[15] KATCHER, J. Postmark: A new file system benchmark. Tech. Rep. TR3022, Network Appliance, 1997. `http://communities.netapp.com/servlet/JiveServlet/download/2609-1551/Katcher97-postmark-netapp-tr3022.pdf`.

[16] KOZUCH, M., SATYANARAYANAN, M. Internet Suspend/Resume. In *Proceedings of the Fourth IEEE Workshop on Mobile Computing Systems and Applications* (Callicoon, NY, June 2002).

[17] LAMPSON, B., ABADI, M., BURROWS, M., AND WOBBER, E. Authentication in distributed systems: theory and practice. *ACM Trans. Comput. Syst. 10*, 4 (1992), 265–310.

[18] LEE, D., BAER, J.-L., BERSHAD, B., AND ANDERSON, T. Reducing startup latency in web and desktop applications. In *WINSYM'99: Proceedings of the 3rd conference on USENIX Windows NT Symposium* (Berkeley, CA, USA, 1999), USENIX Association, pp. 17–17.

[19] LUMB, C. R., SCHINDLER, J., AND GANGER, G. R. Freeblock Scheduling Outside of Disk Firmware. In *FAST '02: Proceedings of the Conference on File and Storage Technologies* (Berkeley, CA, USA, 2002).

[20] MENON, J., PEASE, D. A., REES, R., DUYANOVICH, L., AND HILLSBERG, B. Ibm storage tank – a heterogeneous scalable san file system. *IBM System Journal 42*, 2 (2003).

[21] RAVI, N., NARAYANASWAMI, C., RAGHUNATH, M., AND ROSU, M. Towards securing pocket hard drives and portable personalities. *IEEE Pervasive Computing 6*, 4 (2007).

[22] SAILER, R., ZHANG, X., JAEGER, T., AND VAN DOORN, L. Design and implementation of a tcg-based integrity measurement architecture. In *SSYM'04: Proceedings of the 13th conference on USENIX Security Symposium* (Berkeley, CA, USA, 2004), USENIX Association, pp. 16–16.

[23] SAPUNTZAKIS, C., CHANDRA, R., PFAFF, B., CHOW, J., LAM, M., ROSENBLUM, M. Optimizing the Migration of Virtual Computers. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation* (Boston, MA, Dec. 2002).

[24] SATYANARAYANAN, M., GILBERT, B., TOUPS, M., TOLIA, N., SURIE, A., O'HALLARON, D. R., WOLBACH, A., HARKES, J., PERRIG, A., FARBER, D. J., KOZUCH, M. A., HELFRICH, C. J., NATH, P., AND LAGAR-CAVILLA, H. A. Pervasive Personal Computing in an Internet Suspend/Resume System. *IEEE Internet Computing 11*, 2 (2007), 16–25.

[25] SESHADRI, A., LUK, M., SHI, E., PERRIG, A., VAN DOORN, L., AND KHOSLA, P. Pioneer: verifying code integrity and enforcing untampered code execution on legacy systems. In *SOSP '05: Proceedings of the twentieth ACM symposium on Operating systems principles* (New York, NY, USA, 2005), ACM, pp. 1–16.

[26] SMALDONE, S., GILBERT, B., BILA, N., IFTODE, L., DE LARA, E., AND SATYANARAYANAN, M. Leveraging smart phones to reduce mobility footprints. In *Mobisys '09: Proceedings of the 7th international conference on Mobile systems, applications, and services* (New York, NY, USA, 2009), ACM, pp. 109–122.

[27] SOULES, C. A. N., GOODSON, G. R., STRUNK, J. D., AND GANGER, G. R. Metadata efficiency in versioning file systems. In *FAST '03: Proceedings of the 2nd USENIX Conference on File and Storage Technologies* (Berkeley, CA, USA, 2003), USENIX Association, pp. 43–58.

[28] SUN MICROSYSTEMS, I. OpenOffice.org - The Free and Open Productivity Suite. OpenOffice.org Project Home Page. `http://www.openoffice.org/`, 2009.

[29] SURIE, A., PERRIG, A., SATYANARAYANAN, M., AND FARBER, D. J. Rapid trust establishment for pervasive personal computing. *IEEE Pervasive Computing 6*, 4 (2007), 24–30.

[30] TOLIA, N., HARKES, J., KOZUCH, M., AND SATYANARAYANAN, M. Integrating portable and distributed storage. In *FAST '04: Proceedings of the 3rd USENIX Conference on File and Storage Technologies* (Berkeley, CA, USA, 2004), USENIX Association, pp. 227–238.

[31] TRAEGER, A., ZADOK, E., JOUKOV, N., AND WRIGHT, C. P. A nine year study of file system and storage benchmarking. *Trans. Storage 4*, 2 (2008), 1–56.