Optimizing Storage Performance for VM-Based Mobile Computing

STEPHEN SMALDONE, Rutgers University
BENJAMIN GILBERT and JAN HARKES, Carnegie Mellon University
LIVIU IFTODE, Rutgers University
MAHADEV SATYANARAYANAN, Carnegie Mellon University

5

This article investigates the transient use of free local storage for improving performance in VM-based mobile computing systems implemented as thick clients on host PCs. We use the term *TransientPC systems* to refer to these types of systems. The solution we propose, called *TransPart*, uses the higher-performing local storage of host hardware to speed up performance-critical operations. Our solution constructs a virtual storage device on demand (which we call *transient storage*) by borrowing free disk blocks from the host's storage. In this article, we present the design, implementation, and evaluation of a TransPart prototype, which requires no modifications to the software or hardware of a host computer. Experimental results confirm that TransPart offers low overhead and startup cost, while improving user experience.

Categories and Subject Descriptors: D.4.2 [**Operating Systems**]: Storage Management; D.4.3 [**Operating Systems**]: File Systems Management

General Terms: Design, Experimentation, Management, Performance

Additional Key Words and Phrases: Disk borrowing, file system, free block borrowing, mobile computing, opportunistic mobile computing, performance optimization, pervasive computing, storage, transient storage, TransPart, virtualization, virtual machine, VM

ACM Reference Format:
Smaldone, S., Gilbert, B., Harkes, J., Iftode, L., and Satyanarayanan, M. 2013. Optimizing storage performance for VM-based mobile computing. *ACM Trans. Comput. Syst.* 31, 2, Article 5 (May 2013), 25 pages.
DOI: http://dx.doi.org/10.1145/2465346.2465348

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 [Cáceres et al. 2005] is a well-known example of such a system, storing the entire state of a user VM in a bootable USB key. The Collective [Chandra et al. 2005; Sapuntzakis et al. 2002] and the Internet Suspend/Resume®

This research was partly supported by the National Science Foundation (NSF) under grant numbers CNS-0509004, CNS-0831268, CNS-1111811 and CNS-1117711. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation (NSF), Rutgers—The State University of New Jersey, or Carnegie Mellon University.
Internet Suspend/Resume(ISR)® and OpenISR® are registered trademarks of Carnegie Mellon University.
S. Smaldone is now with EMC^2 Corporation, Backup Recovery Systems Division.
Authors' addresses: S. Smaldone, EMC^2 Corporation, Backup Recovery Systems Division, Princeton, NJ; email: steve.smaldone@emc.com; L. Iftode, Rutgers—The State University of New Jersey, Department of Computer Science, Piscataway, NJ; email: iftode@cs.rutgers.edu; B. Gilbert, J. Harkes, and M. Satyanarayanan, Carnegie Mellon University, School of Computer Science, Pittsburgh, PA; email: {bgilbert, jaharkes, satya}@cs.cmu.edu.

DOI: http://dx.doi.org/10.1145/2465346.2465348

Table I. Portable Storage Device Characteristics

Storage Device	Label	Type	Size (GB)	Speed (RPM)	Transfer Rate (MB/sec) Read	Write
PNY Attache USB Flash Drive	SanDrive	USB	16	Flash	31	7
SanDisk MicroSD Card	MSD	USB	8	Flash	16	12
Apple iPod	IPOD	USB	20	4200	13	12
Internal SATA Drive	-	SATA	250	7200	42	32
Guest VM on SATA Drive	Host	SATA	250	7200	40	28

Transfer rate results are the mean of five measurements. The first four rows present raw device performance as measured from within the host OS. The fifth row (*Guest VM on SATA Drive*) measures SATA device performance from within the guest VM OS. All standard deviations are less than 6%.

system [Kozuch and Satyanarayanan 2002; Satyanarayanan et al. 2007] are two other examples of this genre of systems. MokaFive [MokaFive 2010] 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 *TransientPC systems* to refer to this broad class of mobile computing systems. Their usage model is quite different from the email, Web access, and social networking capabilities provided by BlackBerries, iPhones, and other mobile devices. The strength of TransientPC systems lies in their ability to precisely, safely, and rapidly recreate a user's Windows or Linux desktop environment as a thick client on host hardware at any time and place.

Zero-install TransientPC systems are those that are stored on light-weight, portable storage devices (e.g., USB disks) and carried by the user. These systems share three important attributes. First, no preinstalled 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. Therefore, TransientPC state only exists on the host during the user session initiated at host boot-up and terminated at host shutdown. During the user session, it is always stored persistently and accessed from an associated portable storage device.

Unfortunately, portable storage devices sacrifice I/O performance in order to obtain the highest capacity and robustness at the lowest cost, size, and weight. In addition, USB connectivity limits storage bandwidth well below that of internal storage connectivity such as SATA. As Table I shows, the I/O read and write performance of typical USB-attached storage devices is substantially lower than that of an internal disk. The consequence is that operating system performance can be severely impacted, including basic functionality such as swapping and application launch.

In this article, we propose a solution called *TransPart* to provide faster storage for VM-based mobility. TransPart leverages the free disk blocks of the host computer's internal disk to construct a virtual storage device from these free disk blocks and uses it to temporarily take the place of the portable storage device in a TransientPC system. We call this storage *transient storage*. TransPart ensures reciprocal protection between the VM and host by isolating their disk accesses. As long as the integrity of TransPart is preserved (the USB device is not modified), the integrity of its virtual storage device is also preserved. Hence, the privacy and integrity of data in the nonfree parts of the host disk are also preserved. Software running within a VM (malicious or not)

cannot view or modify nonfree parts of the host disk. In this article, we present the design, implementation, and evaluation of a TransPart prototype. Experimental measurements from this prototype confirm that it offers low overhead and start-up cost, while improving user experience.

We make the following contributions in this article.

— We identify a new storage-level concept, *transient storage*, that is well aligned with VM mobility.
— 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 substantially improves interactive I/O performance while incurring low start-up and shutdown costs.
— We discuss broader uses of the *transient storage* concept.

2. BACKGROUND AND RELATED WORK

2.1. TransientPC Systems

SoulPad [Cáceres et al. 2005] is a good example of a TransientPC system. Under SoulPad, a user's computing state is completely encapsulated 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 preinstall any software on target machines. Only the USB storage device needs to be configured with the correct boot image.

ISR [Kozuch and Satyanarayanan 2002; Satyanarayanan et al. 2007], the Collective [Chandra et al. 2005; Sapuntzakis et al. 2002], and MokaFive [MokaFive 2010], are TransientPC systems that transport VM state over the network from a server, or even from a user's smart phone [Smaldone et al. 2009]. While the necessary boot image is typically preinstalled on target machines, zero-install variants of these systems also exist. These boot up a target machine from a USB storage device to establish the correct VMM environment and then fetch VM state over the network. Figure 1 illustrates the components of a TransientPC system. Throughout the article, when we refer to *zero-install TransientPC systems*, we mean the subclass of TransientPC systems that initially boot up from USB storage devices.

2.2. Opportunistic Use of Free Storage

The opportunistic use of free blocks on a storage device was investigated by Cipar et al. [2007] in the *Transparent File System (TFS)*. Although TransPart and TFS exhibit similarity in the use of free disk blocks, there are two major differences between them. First, 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 must be modified to use TFS due to a weakening of file system persistence guarantees (see the following). In contrast, TransPart targets unmodified VM-based mobile computing systems (specifically TransientPC systems). Second, 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, and provides a file system interface. TransPart, on the other hand, is used only

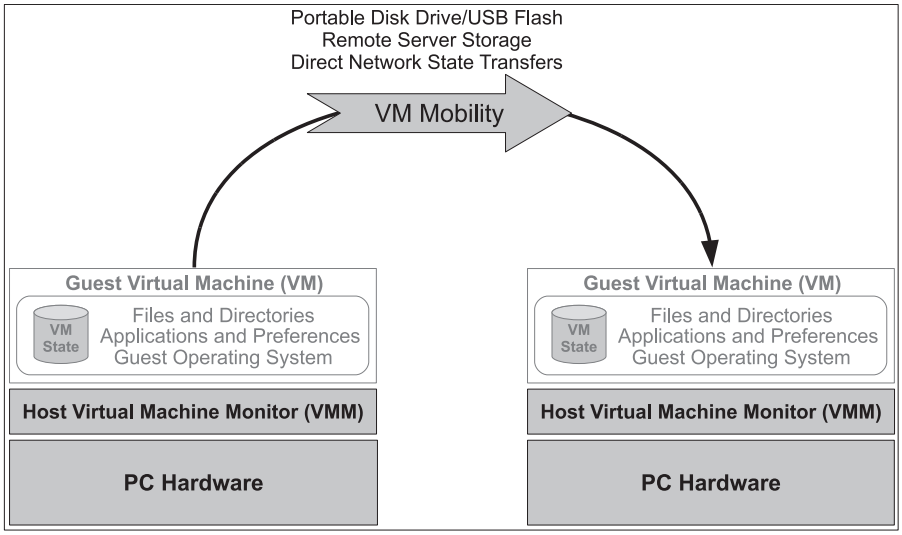


Fig. 1. Components of a TransientPC system.

when the host OS is inactive, hence its implementation completely avoids changes to the host file system. Instead, it is implemented in userspace, supports ext2/3/4 and NTFS file systems, and provides a block device interface that can support any desired file system or can be utilized for non-file system purposes, such as swap space.

As a result of these two differences, applications that wish to take advantage of TFS, have to be tolerant of the weak persistence guarantees it provides: TFS can unilaterally release space allocated to any file that is not currently open. In contrast, TransPart (during a user session) can offer the classic persistence guarantee of a file system: once created, a file remains in existence until it is explicitly deleted (or the TransientPC user session is terminated). This strict compatibility with the guarantees provided by existing file systems allows unmodified TransientPC systems (and, by extension, guest OSes and guest applications within a locally-executing VM) to use TransPart. Finally, 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.

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 [Beck et al. 2002]. Other work of this genre include IBM's Storage Tank [Menon et al. 2003] and the work of Kang and Reddy [2006] on virtual storage provisioning. These efforts address storage opportunism at network scale, while TransPart targets individual storage devices. Also relevant is the work of Lumb et al. [2002] on using free blocks to improve disk scheduling by overlapping high-priority and low-priority disk workloads.

3. DESIGN AND IMPLEMENTATION

In this section, we present the design and implementation of TransPart. Throughout this article, we use a number of specific terms as we refer to the assumed usage scenario, and the TransPart system model, design, and implementation. For clarity, we define our envisaged usage scenario and the associated terms first.

3.1. Usage Scenario and Notational Terminology

For the zero-install Transient PC model, we assume a user will borrow a host PC, connect her portable USB storage device and boot the host using the TransientPC software stored there. 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 host components. We also refer to the owner's PC as the host PC.

The virtual machine monitor or *VMM* is the software execution environment running on the host PC within which a guest virtual machine or *VM* may execute. Collectively, the VMM and guest VMs define the borrower's software stored on her portable USB storage device (the zero-install TransientPC software) that executes on the owner's PC hardware. The guest VM executes the 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: (1) it has not been allocated to a host file system, or (2) it has been allocated to a host file system, but is not currently in use by that file system.

3.2. Correctness Criteria

The design of TransPart is directed by a set of correctness conditions. We list these in the following.

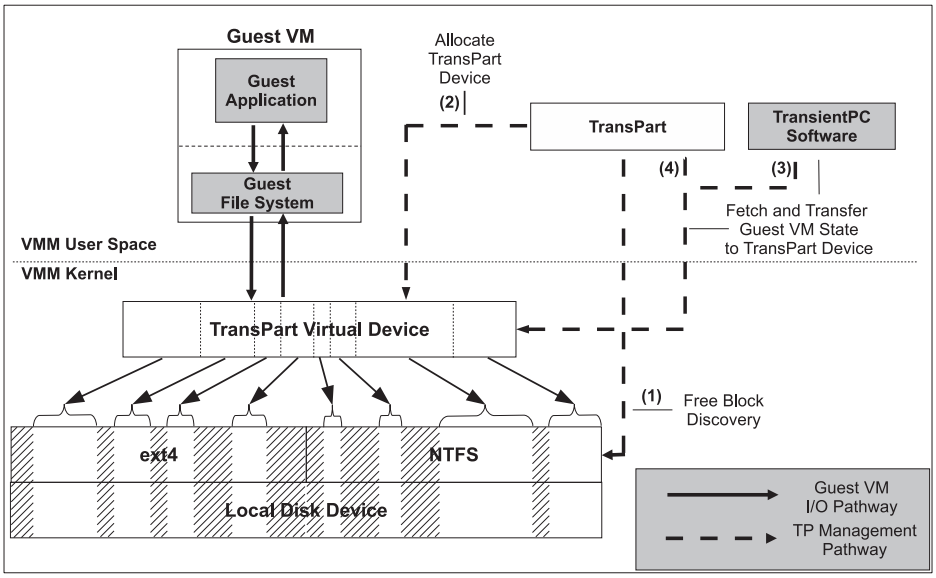
— Borrow the free disk blocks of a host hard disk to create a virtual storage device for a guest execution environment.
— Isolate the in-use host disk blocks from guest VM read and write operations.
— Achieve the first two goals without any modifications to host hardware, OS, or application software.
— Ensure that the modified blocks left behind by TransPart do not leak private information.

Finally, our design assumes that the host OS does not execute during the TransientPC user session. This enforces a strict isolation between the host OS and the TransientPC software's accesses to the host storage device. Furthermore, to avoid potential host file system consistency issues (see Sections 3.4 and 3.5) the VMM is restricted from even mounting any host file systems and using them directly.

3.3. Design Overview

To meet the correctness criteria, TransPart must perform several tasks. These operations are depicted in Figure 2. First, it must discover free blocks on host disks (1) and aggregate these blocks to allocate a TransPart logical disk (2). Then, guest VM state is transferred to the TransPart logical disk. This step is, in some cases, performed by the TransientPC software directly (3). In other cases, it may be performed by TransPart (4). Also, it may occur on-demand (demand-fetch) during guest VM execution, or prior to resuming (prefetch) the guest VM. Our prototype implementation utilizes the prefetch approach, but we include mention of the demand-fetch approach since our design does not preclude its inclusion as a user-selectable option.

Figure 2 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 logical disk, ultimately being serviced by one or more host disks. I/O performance is improved by conventional buffer caches in both the host and guest OSes; these are omitted from the figure for clarity.



(1) TransPart performs free block discovery. (2) TransPart performs device allocation. (3) & (4) VM state fetched (from network or USB disk) and transferred to the newly allocated TransPart logical disk. In the case of zero-install TransientPC systems, these steps may be performed by TransPart. Otherwise, it is handled by the TransientPC software.

Fig. 2. TransPart system overview.

Finally, we address the fourth correctness criteria by copying the VM state to the TransPart logical disk in encrypted form. Since some TransientPC systems provide encryption-based protection, TransPart can leverage this for those systems, having only to clean data that was unencrypted during VM execution. For TransientPC systems that do not provide such protection, the TransPart logical disk can be layered upon a block device-level encryption driver. In that case, TransPart can safely leave copies of data behind after the user session has been completed. The TransPart prototype used in our evaluation utilizes a TransientPC system (see Section 4.1) that does provide encryption-based privacy-protection, so we do not enable block device-level encryption for our experiments. We discuss the issue of data privacy further in Section 5.1.

3.4. Free Block Discovery and Allocation

During the host's boot process, and prior to execution of a guest VM, TransPart scans the host's storage devices to discover available free blocks and verify that they are free (even under conditions of host OS hibernation, or host file systems that were not cleanly unmounted). This process occurs in two phases. In the first phase, TransPart enumerates host devices searching for storage devices; then it discovers individual storage partitions stored on these devices. Such partitions include physical disk partitions as well as aggregate partitions such as software RAID and Logical Volume Manager volumes. Each storage partition usually contains a single file system or swap space.

In the second phase, TransPart searches through each storage partition to discover the available free blocks for each partition type or file system it supports (illustrated by Figure 2). Most, if not all, modern file systems maintain a set of block allocation tables as metadata on disk, for each formatted disk partition. For instance, the Linux

ext2 [Card et al. 1994] file system stores a block allocation bitmap in each block group, which maintains the allocation status for each block in the associated block group. TransPart utilizes file system on-disk semantics to properly parse the file system metadata and discover free disk blocks. To preserve host file system consistency, TransPart will only use host file systems that were unmounted cleanly when the host was powered off, and will avoid using disk blocks (in some cases whole partitions) that store host OS hibernation state. Section 3.5.1 presents further, low-level, details of free block discovery.

Finally, TransPart tracks free blocks not as individual units, but as free block *extents*, or runs of consecutive blocks. This minimizes the amount of data that must be tracked 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. Once TransPart has discovered free disk blocks, it allocates a TransPart logical disk for the guest VM. The minimum size for a given TransPart logical disk is determined by the size of the guest VM state. Section 3.5.2 presents further, low-level, details of free block allocation.

3.5. Implementation

Our TransPart prototype is implemented in about 550 lines of C and a small amount of Python 2.6 and shell script code. Together, the TransPart 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 are made to existing software in either the host or the guest.

Our prototype supports free disk block discovery from ext2/3/4 and NTFS file systems, as well as Linux swap devices. For low-level semantic access to ext2/3/4 file systems, TransPart uses the Linux e2fsprogs [Ts'o 2010] library. For semantic access to NTFS file systems, it uses the ntfsprogs [NTFSProgs 2007] library. To create in-kernel TransPart logical disks, we use the existing Linux Device-Mapper [devmapper 2001]. By doing so, we take advantage of all the performance benefits of the mature Linux kernel code supporting logical volume management.

3.5.1. Finding Free Disk Space. The task of collecting available disk space into a TransPart logical disk is performed by the `gather_free_space` tool. It performs the following actions.

First, the e2fsprogs `blkid` library is used to gather a list of disk partitions and their file system types. Since `gather_free_space` executes after VMM initialization, Linux software RAID and Logical Volume Manager (LVM) [LVM 2008] partitions have already been assembled, and are included in the partition list returned by `blkid`.

Next, each partition is examined using the appropriate file system-specific libraries, unless it contains a Linux swap partition. In that case, it is examined directly by `gather_free_space`. Partitions containing file systems not supported by `gather_free_space` are skipped. For each supported file system, `gather_free_space` performs a read-only consistency check on the file system to ensure that it will not damage the file system. These checks are performed similar to standard `fsck`, by examining on-disk file system meta-data (ext4 superblock and NTFS Master File Table). If a consistency check indicates that a partition containing a file system was not unmounted cleanly or is known to have errors, or if any errors are encountered while reading a file system on a partition, the partition is skipped.

Normally, swap partitions are not available for use by TransPart, since host OS hibernation state may reside in what would otherwise be considered free blocks. Other partitions (those that do not contain hibernated state) are always safe to use as TransPart does not modify any on-disk file system metadata or data. However, if the

host PC has been shut down rather than hibernated, TransPart can also use disk blocks from any swap partitions it finds, since in this case, there is no risk of overwriting important data. The presence of hibernation state in a partition is determined similarly to the manner in which a host OS accomplishes it, by searching for the existence of magic numbers at specific locations of hibernation partitions (in the hiberfile.sys header for Microsoft Windows hosts and the low-order blocks of the swap partition(s) for Linux hosts).

For each file system that is deemed safe to use, `gather_free_space` obtains the file system block allocation bitmap. These bitmaps are scanned to generate a stream of extents, or runs of free blocks. The lengths and locations of the largest 100,000 extents yet encountered are recorded in a binary heap. To improve the performance of TransPart logical disks, extents smaller than 4 MB are ignored. Building TransPart logical disks from larger extents helps to decrease the average number of disk seeks required during I/O operations, thus improving overall throughput.

Finally, the Linux device-mapper infrastructure is directed, via the `libdevmapper` library, to create a block device from the extents recorded in the binary heap. Device-mapper [devmapper 2001] provides a mechanism for creating virtual block devices that redirect accesses to other block devices in a configurable, table-driven fashion. Before the list of extents is presented to `libdevmapper`, it is sorted by originating partition and sector offset within that partition. This reduces seeks during streaming accesses to the TransPart logical disk, since its sectors are more likely to be ordered with respect to sectors on the physical disks.

The device-mapper implements a logical block device as a table of extents. If the host's file systems were highly fragmented and `gather_free_space` were to include every free extent, the logical block device table could grow quite large. Since the Linux kernel places the table in the vmalloc allocation area (max size is 128 MB on x86 systems and cannot be swapped to disk), limiting the table to the largest 100,000 extents ensures that `gather_free_space` can create the largest TransPart logical disk possible without consuming more than a few megabytes of kernel memory.

3.5.2. Allocating Free Disk Space. The `gather_free_space` tool can create a TransPart logical disk, but does not allocate it to the guest VM. This task is performed by the `early_scratch_setup` initialization script, which executes during VMM boot.

First, `early_scratch_setup` executes `gather_free_space` to create a TransPart logical disk. By default, `early_scratch_setup` imposes a minimum size for the logical disk. If `gather_free_space` is unable to create a logical disk of at least this size, `early_scratch_setup` produces an error and terminates. If a logical disk creation succeeds, `early_scratch_setup` then creates an LVM volume group on top of the TransPart logical disk. Next, it creates a swap partition (2 GB default size) within the volume group, formats, and enables it for use. Finally, the remainder of the space in the volume group is formatted to create a file system (ext4 by default), which is then mounted within the VMM. To improve performance under the default ext4 case, mkfs.ext4 is run with the lazy itable init option, causing mkfs.ext4 to defer initialization of file system block groups to a background task, which runs when the file system is first mounted.

4. EVALUATION

Our experimental evaluation of the TransPart prototype addresses the following two questions.

— How much does guest VM application and OS performance improve due to TransPart?
— How much additional time is added during startup and shutdown due to TransPart?

Together, these questions allow us to explore both the benefits and costs of the TransPart system. Our goal is to demonstrate the benefits of the TransPart system towards improving the user experience within the framework of VM-encapsulated mobile computing.

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

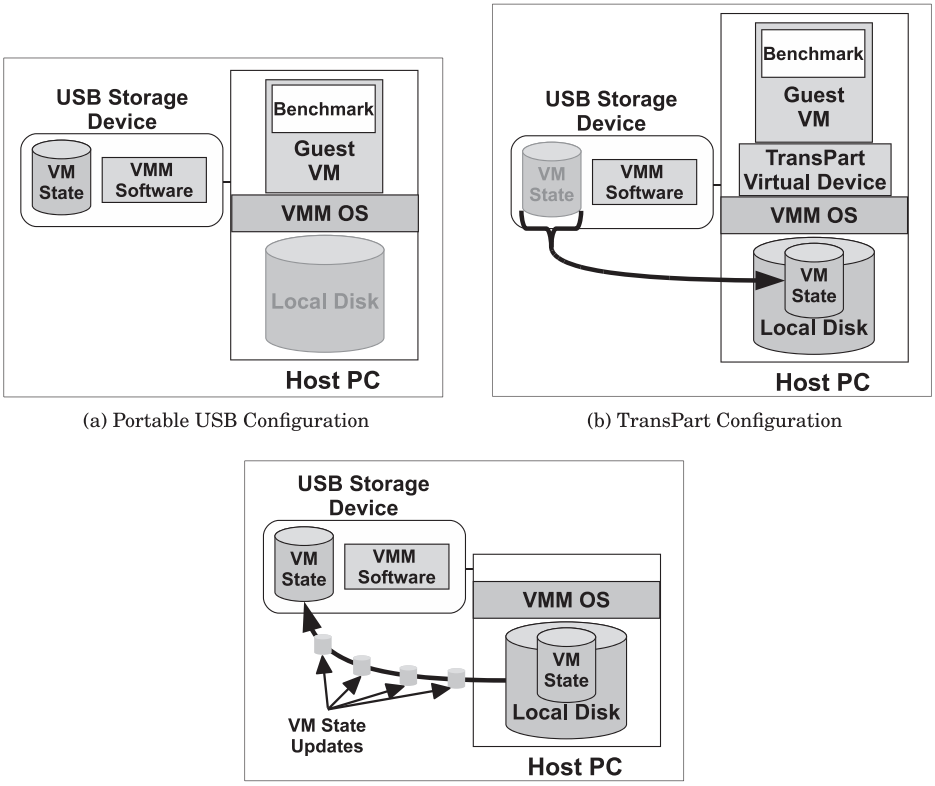
4.1. Experimental Methodology

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, as our host PC. As our TransientPC system, we utilize PocketISR; the zero-install variant of the Internet Suspend/Resume® system [Kozuch and Satyanarayanan 2002; Satyanarayanan et al. 2007] running OpenISR® version 0.9.9. We use a VMware guest VM configured to use 512 MB of memory and a 4 GB virtual disk. We chose the size of our guest VM to represent a small real-world personal computing environment that a user might actually possess on a USB key. The guest VM state is fully stored on the portable storage device. 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 ext4 file system.

Figure 3 shows a diagram of the individual components of our experimental configuration. For the base TransientPC case (Figure 3(a)), all guest VM state is accessed directly from the attached portable USB device during guest VM execution. For the TransPart case (Figure 3(b)), guest VM state is copied to the host PC's SATA disk and accessed from there during guest VM execution. After suspending the guest VM, guest VM state modifications are copied back to the portable USB storage device prior to disconnecting the portable storage device (Figure 3(c)).

We select a range of portable storage devices to hold the guest virtual machine state. Table I lists 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 sustainable sequential read and write performance for each device. Rows 1–3 of the table describe the three different portable USB devices used in our evaluation. Row 4 describes the performance of the internal SATA drive as measured from the host OS, and Row 5 for the same SATA drive but measured from within a guest VM running as a process within the host OS. By comparing rows 1-3 to rows 4 and 5 in the table, we observe that all three portable USB devices exhibit lower performance than the host SATA drive (either from the host OS or guest OS). Finally, by comparing rows 4 and 5, we observe a small, but measurable overhead associated with accessing the SATA device from within a VM.

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 [Katcher 1997] 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 [Howard et al. 1988], which emulates a software development workload (Section 4.2.3). To determine user-perceived desktop application performance, we utilize a custom office productivity benchmark consisting of common operations within the LibreOffice.org [LibreOffice 2013] 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,



(a) Portable USB Configuration

(b) TransPart Configuration

(c) VM State Writeback at Suspend

TransientPC configuration (a) and TransPart configuration (b & c) are shown. Benchmarks are executed within guest VM on host PC. For TransPart case (b), guest VM state is first transferred to host PC disk, while (a) guest VM state is accessed directly from USB for the base TransientPC case. For TransPart case (c), modified VM state is written back to the USB storage device once the guest VM has been suspended.

Fig. 3. Experimental configuration.

the Bonnie++ [Coker 2001] benchmark tests the performance of a set of simple file system accesses to a single large file (Section 4.2.6).

The costs of TransPart can be broken down into two categories, startup costs and shutdown costs. At startup, a new TransPart logical disk must be allocated from the free blocks available on the host disk. Additionally, VM state must be fetched from the USB device and copied to the TransPart logical disk prior to VM Resume (Figure 3(b)). These two components comprise the additional overhead that does not occur when resuming the VM without TransPart. At shutdown, TransPart must copy the modified portions of the suspended VM's state (memory and disk) back to the USB device (Figure 3(c)). This comprises the additional (worst-case) TransPart costs during shutdown. To better understand the real impact of these costs, we perform experiments to measure the individual startup and shutdown times for each of the different portable USB devices with and without TransPart (Section 4.3).

All benchmarks in the evaluation are run a minimum of five times, and we report the average results. In all experiments, when using the USB devices (rows 1–3 of Table I), the PC is booted from the USB device, and the host OS and the VMM are loaded and executed directly from the USB device. Results measured when using one of the three portable USB devices are labeled SanDrive, MSD, and IPOD. We also include results

Table II. Postmark Configuration

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

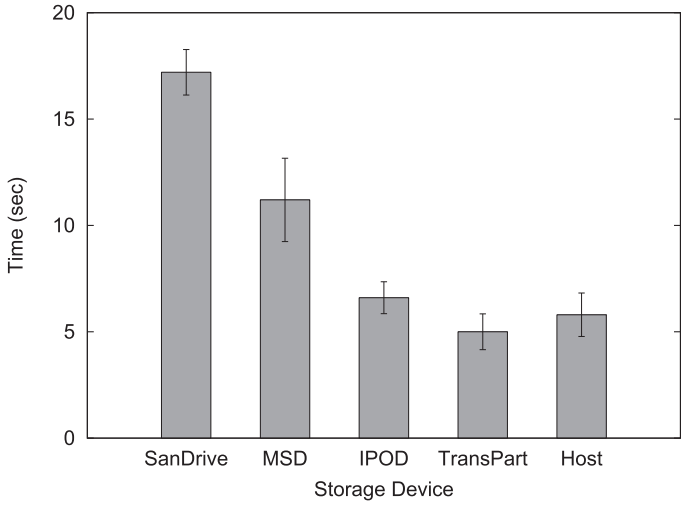


Fig. 4. Postmark results.

in the absence of TransPart. These are labeled Host and present the results for a guest VM that has been installed on and executes directly from the host PC. We expect the Host case to represent the optimal VM performance for a typical (non-TransPart) VM. Before each new run of a benchmark, we power down the VM and revert its state to a saved checkpoint, and we flush the host OS buffer cache. This ensures a consistent starting point for each experiment.

4.2. TransPart Performance Benefits

4.2.1. Postmark. In this experiment, we execute the Postmark [Katcher 1997] benchmark (version 1.51). Postmark was created in 1997 and its workload is characterized by many operations on short-lived, small files. It consists of many data and metadata operations. Since it does not attempt to approximate application processing, it performs very little CPU activity, and is I/O-intensive by design. Table II lists the Postmark configuration that was used in this experiment, selected based on the Traeger et al. [2008] and Soules et al. [2003] studies.

Figure 4 presents the results of this experiment. Each bar in the figure represents one of the configurations 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 diverse set of both data and metadata operations. As such, it attempts to generate a realistic workload under test. The IPOD and TransPart cases outperform both flash drive cases (SanDrive and MSD) by up to

a factor of 3, while the TransPart, IPOD, and Host cases are all similar to each other for this workload. TransPart and Host are very close (less than 2 seconds separating them), demonstrating that TransPart achieves near optimal performance for this benchmark.

From Figure 4, we also observe that the poor write performance of SanDrive outweighs its good read performance. Further, although MSD and IPOD can achieve similar sustained sequential throughput (Table I), IPOD performs better on this mixed workload. It appears on the surface that, although Postmark was configured to perform an equal amount of reads and writes (Table II)), write performance has a larger impact on the results than read performance. It is likely that there are at least three contributing factors.

First, the default file system block size is 4 KB for ext4 and NTFS, while for some (but not all) USB flash sticks, the erase block size is 128 KB. It is possible that a different choice in file system block size would cause better alignment in file system writes to block device erase block boundaries. Second, since all of the experiments are conducted from inside a guest VM executing within the host OS and VMM environment, there is the possibility of the occurrence of double buffering. That is, there may be disk blocks cached within the buffer cache of the guest VM that are also cached within the buffer cache of the underlying host OS. Wherever possible, we try to avoid the effects of this on our results by flushing both buffer caches between experimental runs as mentioned earlier (Section 4.1). However, we cannot avoid double buffering during the course of an experimental run. During the Postmark benchmark, the effects of double buffering can delay when writes are flushed to the device. Therefore, it is possible for reads from the VM to cause writes to be flushed from the host OS buffer cache. Third, as mentioned in Section 3.3, the TransientPC system we use with our prototype (ISR) encrypts the guest VM state as a privacy-preserving mechanism. As an optimization, ISR chunks the VM disk file and only decrypts chunks of the VM disk at the time of first access, saving the decrypted chunks to a separate sparse file on-disk. A consequence of this is that reads to encrypted chunks will also cause writes due to decryption. Therefore, even read-only workloads will experience a certain amount of writes. While not causing a 2:1 ratio of writes to reads, it does lead to a ratio slanted towards writes.

Finally, we also observe from Figure 4 that TransPart performs better than Host on this benchmark. We recall that for the TransPart case, only the guest VM is performing I/O operations to the host SATA disk, while the host OS and VMM execute from the USB key. We speculate that there is a slight benefit due to this parallelism for the TransPart case, as demonstrated by the slightly faster completion time for TransPart over Host. In the TransPart case, we believe that the separate I/O device for the host OS and VMM slightly offsets the virtualization overheads (including any disk I/O the host OS and the VMM perform during the test). While in the Host case, the SATA device handles all I/O (including any from the host OS and the VMM). It is also possible that the disk blocks being allocated on the TransPart logical disk have better ZCAV properties than those chosen by the host OS for Host. Since the results for TransPart and Host are so close (especially when factoring in the variance), we have left further exploration of this for future work.

4.2.2. Application Launch Latency. An important indicator of user experience is application launch latency [Lee et al. 1999]. In this experiment, we are specifically interested in quantifying the time it takes for a typical desktop application to be available for use 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

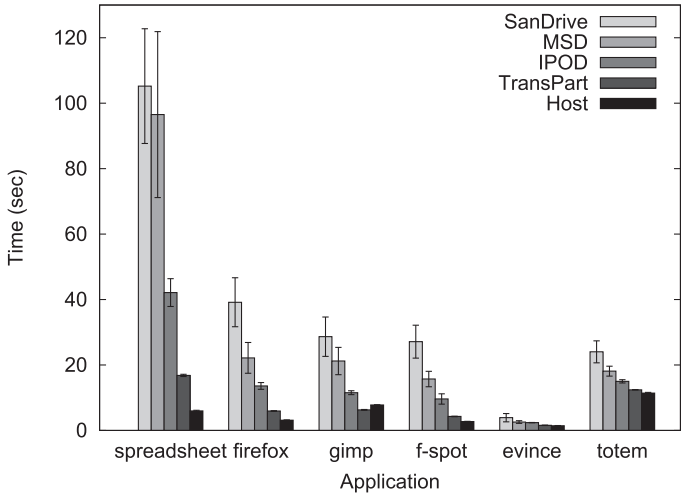


Fig. 5. Application launch latency.

guest VM for the three portable USB drives, TransPart, and a guest VM that executes within VMM software installed on the host. We use a custom script to launch and measure the launch latency for six applications: (1) the LibreOffice.org spreadsheet, (2) the Firefox web browser, (3) the Gimp image manipulation tool, (4) the F-Spot photo management application, (5) the Evince document viewer, and (6) the Totem multimedia (audio/video) player.

Figure 5 presents the results of this experiment. From the figure, we make three observations. First and most important, TransPart improves performance for all applications tested. These improvements are most evident when focusing on the spreadsheet and Firefox cases, but TransPart also performs best over all TransientPC cases. When compared to the SanDrive, MSD, and IPOD cases, TransPart exhibits up to a factor of 10, 7, and 4 performance improvements, respectively. Second, application launch latencies for the SanDrive and MSD cases are both high and variable, likely beyond the range of tolerance for a typical user. The latencies for the IPOD case are still noticeably large to a user, but in a more tolerable range. The latencies also vary less for IPOD than the SanDrive and MSD cases. Finally, TransPart exhibits performance close to the Host case for all applications, but some of the results show that there is still room for improvement. The Gimp application performs a number of small data and metadata operations on launch, and so TransPart slightly outperforms the Host case for this application.

4.2.3. Andrew Benchmark. We execute a modified Andrew benchmark [Howard et al. 1988] against the Linux kernel sources (version 2.6.31.6) such that it will exceed the memory capacity of the VM under test in order to force I/O accesses to the disk. The benchmark consists of five phases: *(MakeDir)*, recursively create subdirectories, *(Copy)*, copy the source tree, *(ScanDir)*, query the status of each source file in the tree without accessing data, *(ReadAll)*, read the data of each source file, and *(Make)*, compile and link the sources.

The results are presented in Table III. 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 TransientPC cases. The result is a 109% improvement over the SanDrive case, 76% over MSD, and 11% over IPOD. When examining the results for the individual

Table III. Modified Andrew Benchmark Results

Benchmark	Storage Device				
Phase	SanDrive	MSD	IPOD	TransPart	Host
Overall	5904	4970	3142	2820	2275
MakeDir	34	39	22	14	21
Copy	782	616	178	164	136
ScanDir	43	14	16	9	7
ReadAll	57	137	80	50	42
Make	4987	4506	2846	2583	2070

Completion times (in seconds) of a Modified Andrew benchmark for overall benchmark and for individual phases. All standard deviations are less than 6%.

benchmark phases, we observe that in all but two cases, both rotational disk devices (IPOD and TransPart) substantially outperform the flash devices (SanDrive and MSD). Finally, when comparing TransPart to Host, we observe that TransPart takes 23% longer than Host.

We also observe, by focusing on the SanDrive, MSD, and IPOD columns of Table III, that the sequential read performance for SanDrive is better than MSD and IPOD, as evidenced by the ReadAll phase results (and the prior results in Table I). Yet, both MSD and IPOD perform better than SanDrive for the ScanDir phase. This inconsistency is likely due to the effects of double buffering within the VM OS and host OS buffer caches (see Section 4.2.1). Since all writes are flushed from the VM buffer cache between benchmark phases, but not necessarily flushed from the host buffer cache, the flushing of the pages from the host buffer cache written during the Copy phase can overlap with reads occurring during the ScanDir phase. Additionally, since SanDrive has substantially worse write performance than both MSD and IPOD, it is more likely to be affected in this fashion, due to the additional time flushing the host OS buffer cache will take.

4.2.4. Office Productivity. The goal of this experiment is to measure the user experience for a user who 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 LibreOffice.org Writer application. The benchmark consists of a set of Python scripts that interact with the open API provided by the LibreOffice.org suite [LibreOffice 2013].

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 the study by Karat et al. [1999], which shows the average content creation rate for an average typist. In the ideal case (zero-latency I/O operations), the minimum running time of the benchmark is 15 minutes (900 seconds) due to synthetic think time. This represents the best possible performance for the benchmark.

From the results shown in Figure 6, we observe a modest overall performance improvement between 22 and 73 seconds (from 2% to 8%) when comparing the portable USB device cases to TransPart. These improvements are due to a reduction of application startup 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 host PC. For the portable USB device cases, we experienced additional and varied delays in application

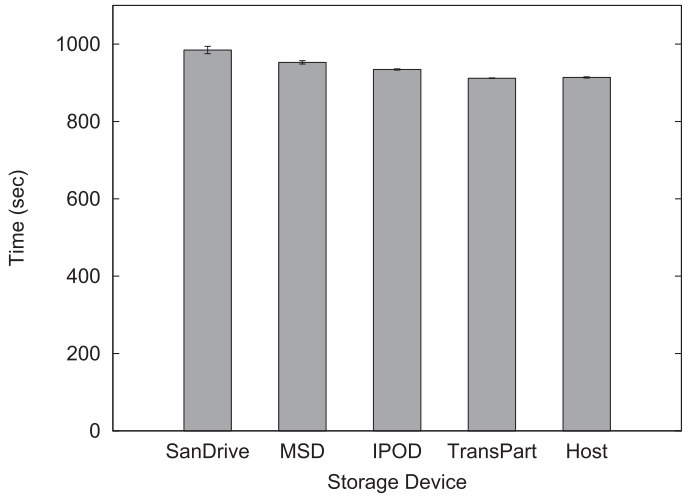


Fig. 6. Office productivity benchmark results.

responsiveness. Finally, when we compare the TransPart and Host cases, we observe that they are similar and TransPart is again near optimal.

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 [CNN 2010]. We evaluate the effectiveness of TransPart for software installs by simulating a user performing software installation tasks using the well-known YUM [2010] 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 (LibreOffice.org) and various related dependency packages. In total, 45 packages are installed with a combined size of 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 software installation operations. Most modern system update processes first download all of the update packages and then install them during a second phase, to ensure that updates are applied together and that success is not subject to network performance. So, we can safely ignore the network download phase for the purposes of this experiment.

Figure 7 presents the results of this experiment. From the figure, we observe that TransPart substantially outperforms all portable USB device cases. Compared to both SanDrive and MSD, TransPart exhibits improvement by over 75%, and results in a 43% improvement over IPOD. Finally, the performance of TransPart and Host are equivalent and TransPart achieves near optimal performance for this benchmark.

4.2.6. Bonnie++. Bonnie++ [Coker 2001] was developed 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 interest of completeness. In this experiment, we report the results of the following tests over all device cases: (1) sequential read tests, (2) sequential write tests, and (3) random seek tests, using Bonnie++ version 1.0.3.

Figure 8 presents the results, 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

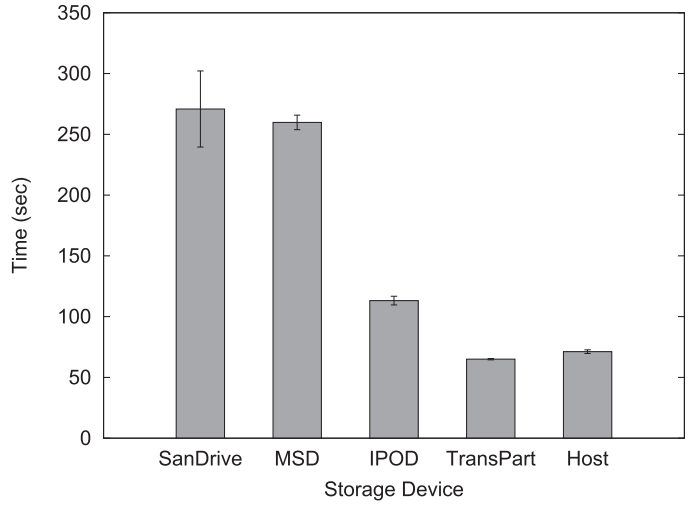


Fig. 7. Software installation benchmark results.

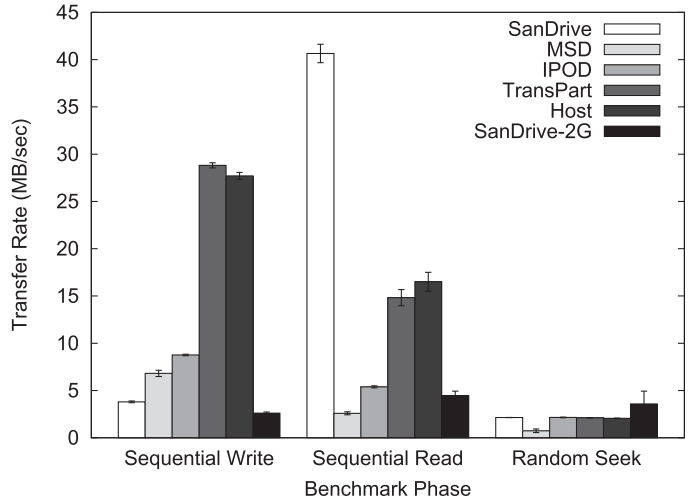
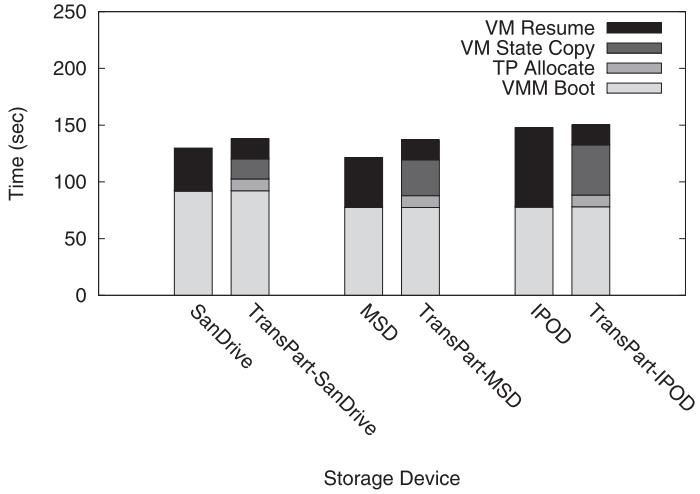


Fig. 8. Bonnie++ results.

TransPart case substantially outperforms all other TransientPC cases for sequential write performance. For the random seek case, the SanDrive, IPOD, and TransPart results are similar, while MSD exhibits poorer performance. For sequential reads, we observe that TransPart outperforms the MSD and IPOD cases. Also, we observe an anomaly for SanDrive read performance.

Investigating more deeply, we find that this is likely due to a combination of the effects of double buffering (Section 4.2.1), the asymmetric read and write performance of the SanDrive device (see Table I), and the Bonnie++ benchmark's choice of test file size (1 GB). To validate this, we ran Bonnie++ again, this time using a larger test file (2 GB), and present the results in Figure 8 as the SanDrive-2GB case. From these results, we observe that the anomaly disappears. We also ran the MSD, IPOD, and TransPart cases with the 2 GB test file, and did not observe any difference from



Stacked bars are composed of the *VMM Boot* and *VM Resume* components for all cases, and *TP Allocate* and *VM State Copy* components for the TransPart cases. All standard deviations are less than 6%.

Fig. 9. Startup time.

the original results. For clarity, we choose not to report them. Finally, the TransPart and Host cases are quite similar, with Host performing slightly better for sequential reads, while TransPart performs slightly better for sequential writes. We interpret these results to mean that TransPart performs close to optimally for this benchmark.

By comparing the results presented in Figure 8 with those from Table I, we observe that the better raw read performance of SanDrive over MSD and IPOD once again does not translate to equivalent gains in sequential read performance for the benchmark. We believe that this is due to the additional writes issued during decryption of portions of the guest VM state on first access (Section 4.2.1). These additional writes effectively offset the gains in read performance for SanDrive and SanDrive-2GB. The SanDrive results do not reflect this due to the benefits from double buffering (described in the previous paragraph). It becomes evident though for the SanDrive-2GB case, where the effects of the double buffering have been removed. The other cases do not exhibit this behavior, due to the fact that they all have nearly balanced read and write performance. The additional writes also have an impact on random read performance, which is compounded by the lack of benefit due to prefetching that sequential reads enjoy.

4.3. TransPart Costs

4.3.1. Startup Time. This experiment measures the total startup time taken to boot the borrowed PC from the different portable USB drives and resume the guest VM. We measure the individual components of startup time for each portable USB drive with and without TransPart. For these experiments, TransPart discovers enough free blocks to allocate a 100 GB TransPart logical disk. The results are presented in Figure 9, where we group the bars by portable USB device. For example, the SanDrive and TransPart-SanDrive cases are both booted from the SanDrive USB device. In the former, the guest VM is resumed directly from SanDrive, while in the latter, the guest VM state is first copied to a TransPart logical disk. Therefore, the cost of using TransPart is the difference between the two bars in each group.

From Figure 9, we observe the net increase in total startup time for TransPart to be in the range of 2–15 seconds. Considering the benefits of TransPart as exhibited

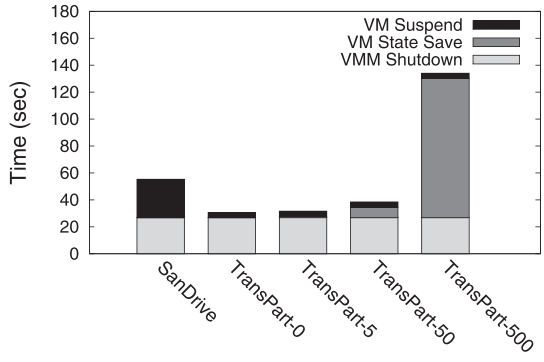
earlier by the performance results presented in Section 4.2, we believe that the modest additional startup costs due to TransPart would be acceptable to users under the usage scenarios we described. In fact, the improvements in application launch latency alone more than make up for this additional startup time. Nevertheless, at this time we have not conducted any rigorous user studies and leave this as future work.

Each stacked bar in Figure 9 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 TransPart logical disk (*VM State Copy*), are also presented. By comparing the TransPart cases with their respective non-TransPart 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 22–68 seconds of additional startup costs (TP Allocate and VM State Copy), accounting for the net increase in startup time.

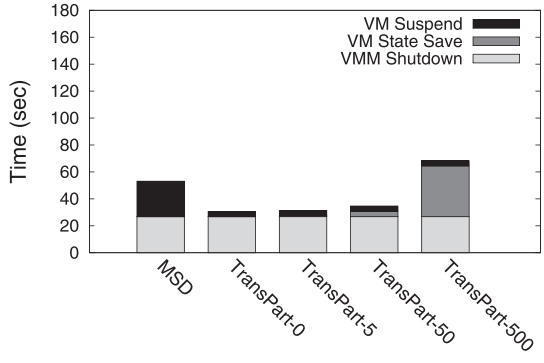
We also observe that the better sustained read performance of SanDrive over MSD (Table I), does not translate to equivalent improvements in VM Resume time from the results in Figure 9. We believe it is due to the fact that startup involves a mix of reads and writes. During startup, persistent VM state is decrypted by the TransientPC system. Additional writes are further contributed due to the decryption of portions of the guest VM state on first access (see Section 4.2.1). Since the performance of both MSD reads and writes is better than SanDrive writes, the poor write performance of SanDrive can offset the gains due to better read performance.

4.3.2. Shutdown Time. This experiment measures the total time taken to suspend the guest VM parcel to the different portable USB drives and power off the borrowed PC. We measure the individual components of shutdown time for each portable USB device and the TransPart case, and present the results in Figure 10. We group the bars into three groups based upon portable USB device. For example, the SanDrive and TransPart-5 (Figure 10(a)) cases are both booted from the SanDrive drive. In the former, the guest VM is suspended directly to the SanDrive drive, while in the latter, the guest VM is first suspended to the host SATA disk and then the modified portions of VM state (memory and disk) are saved to the USB device. To account for a broad range of use cases, we vary the amount of modified state at suspend time between 0 MB and 500 MB. The 5 MB case represents light disk I/O workload, such as Web browsing or online shopping, while the 500 MB case represents a heavy disk I/O workload, such as downloading large files or streaming video. We base our decision to vary modified state between 0–500 MB on prior work [Smaldone et al. 2009], in which we measured the amount of state generated by typical user activities to create various macrobenchmark workloads. Finally, in both the base portable USB device and TransPart-0 MB cases, the VM is resumed, allowed to idle for a fixed period of time, and then shut down without any other VM state modifications due to user activity.

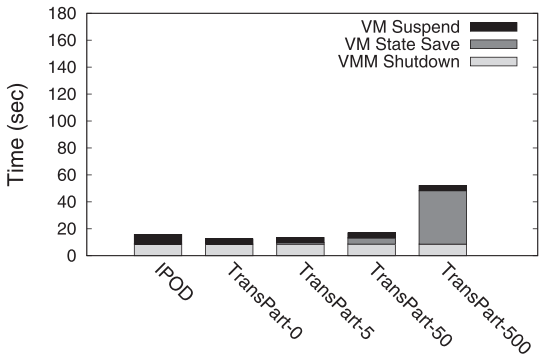
From Figure 10, we observe a net decrease in shutdown time for all TransPart 0 MB to 50 MB cases except for TransPart-50 IPOD (Figure 10(c)). This benefit is in the range of 2-23 seconds. Conversely, for the workloads with the heaviest data modification (TransPart-500), we see a net increase in total shutdown time for TransPart to be in the range of 11–78 seconds. Considering the benefits of TransPart as exhibited earlier by the performance results presented in Section 4.2, we believe that the additional shutdown costs due to TransPart would be acceptable to users under the usage scenarios we present. As with the startup case, the improvements in application launch latency alone (Figure 5) more than make up for this additional shutdown time, but we have not conducted any rigorous user studies and leave that as future work.



(a) SanDrive



(b) MSD



(c) IPOD

Stacked bars are composed of the *VM Suspend* and *VMM Shutdown* components for all cases, and *VM State Save* component for the TransPart cases. The number (0, 5, 50, 500) is the amount of modified state (in MB) that must be saved back to the USB device. All standard deviations are less than 6%.

Fig. 10. Shutdown time.

Also from Figure 10, by comparing the TransPart cases with their respective non-TransPart cases, we observe a trade-off between the reduction in VM suspend time and the additional costs of copying the VM modifications back to the USB device after the VM has been suspended. Each stacked bar in the figure is composed of the time to suspend the guest VM (*VM Suspend*) and the time to power down the host PC from the VMM (*VMM Shutdown*). For the TransPart cases, the time to copy back the varying amounts of modified VM state is also reported (*VM State Save*). We estimate the break-even point for our experimental scenario to occur when there is approximately 150 MB of modified guest VM state to save.

4.4. Summary of Results

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 (SanDrive) perform better than TransPart. Second, for mixed I/O workloads composed of both data and metadata 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

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 host 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 host PC (e.g., in an Internet Cafe, while kiosk computing, or while using a compute cluster, etc.), a level of surveillance (either human or video) would exist to deter such physical tampering from occurring.

Second, we assume that the host OS does not execute during the TransientPC user session. This enforces a strict isolation between the host and the TransientPC (guest VM) software accesses to the host storage device.

Under these two assumptions, we can extend the TransPart model to include a remote software attestation process. One way to do so is for a host owner to connect to a well-known trusted third party verification site using his Internet connection. Then, a local software component can calculate a secure hash of the TransientPC VMM software stored on the connected USB and send it to the verification site. The verification site would then respond with the results of verification by comparing the transmitted hash against known-good hashes. Since all user personalization occurs in a guest VM, 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 third party would establish a root of trust starting with the verification site down to the VMM software stored on the user's USB disk (we refer the reader to Lampson et al. [1992], Arbaugh et al. [1997], Garfinkel et al. [2003], Sailer et al. [2004], Ta-Min et al. [2006], Surie et al. [2007], Ravi et al. [2007], Garriss et al. [2008], McCune [2008, 2010], and Sirer et al. [2011] for the relevant details of these techniques.) The guest VM need not be verified, since only the

VMM needs to be trusted to ensure that a guest VM may safely use the free blocks on the host PC's disk. The guest OS and host OS disk blocks are isolated from each other by the trusted VMM protecting both the integrity and privacy of each set of data. We intend to investigate this and other possible approaches as future work.

The primary goal of this work is to enable one user to borrow another user's PC and execute their personal computing environment utilizing the full available performance without compromising the host PC 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 (allowing OS modifications), we believe that our model provides a reasonable balance of safety for the host PC owner and usability for both parties.

Although we have primarily focused on protecting the host PC owner's data, the proposed solution should also guarantee the guest VM's privacy with respect to the host. TransPart achieves this goal at two levels: (1) the host OS is always suspended during guest VM execution, and (2) the TransPart logical disk can be enabled to encrypt all data stored on it. Item (2) adds a layer of protection to the data by ensuring that unencrypted data is never written to a storage device backing a TransPart logical disk. As stated in Section 3.3, some TransientPC systems already provide privacy-protection for user data. In those cases, a user can safely disable TransPart logical disk encryption, or choose to leave it enabled as an additional layer of protection. Since the TransientPC system used with our prototype provided reasonably good protection, we chose to perform all experiments without TransPart encryption. Anecdotally, we have also conducted a sampling of tests with encryption enabled and find similar performance with and without encryption. This is largely due to the existence of high performance in-kernel block encryption modules, such as the Linux dm-crypt [dm-crypt 2010] module.

5.2. Potential Improvements

Unsupported Partition Types. The current TransPart prototype does not support borrowing the free blocks from encrypted or hidden [TrueCrypt 2010] partitions. Nor does it support other existing volume types, such as Microsoft's venerable FAT file system or DBMS raw partition formats. The TransPart design does not preclude these partition types from being utilized by TransPart, though. At present, there is just a lack of support for such access within Linux, but if suitable partition access libraries were to be developed, then such support could be integrated into TransPart similarly to what is already included.

Free Block Allocation Policies. Our design allows a number of possible approaches for allocation of free blocks to TransPart logical disks. In fact, since there are already multiple layers of indirection, starting with the guest OS down to the host PC hardware, there may be some benefit to considering 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 free block allocation policies within TransPart as future work.

Effects of Disk Fragmentation. Fragmentation of the host's disks can dramatically reduce the effective throughput of the TransPart logical disk. The small free space extents that are produced by file system fragmentation incur the same access latency as larger extents, but many such extents must be combined to produce even a few megabytes of storage capacity. To limit performance degradation, our prototype therefore ignores extents smaller than 4 MB. For sufficiently large or fragmented file systems, a more sophisticated policy might further improve performance; for example, on disks with plenty of free space available, TransPart could afford to be even more

selective in its choice of free extents. We have not fully evaluated the effects of disk fragmentation for this study, but we do consider it an area for future work.

Optimizing Placement of VM State. Recently, some OS vendors such as Microsoft [ReadyBoost 2012] have included the capability of utilizing USB flash drives as file caches to improve the performance of random file accesses. A similar approach could potentially be taken as an extension to TransPart, where both the host storage and the portable USB device can be used in concert to improve file access performance during guest VM execution. This could potentially be satisfied by utilizing some portion of the USB storage as a lookaside cache [Tolia et al. 2004]. On the other hand, it might require caching mechanisms tailored specifically for determining which portions of guest VM state to cache based on disk access patterns (sequential vs. random). We consider an exploration of this idea as a potential future extension to TransPart.

5.3. Beyond Transient Borrowing for Mobility

In this article, we have introduced the concept of transient borrowing of blocks, but we have done so in the context of TransientPC systems. We believe that the transient borrowing concept has broader application and can be extended beyond the mobile computing domain.

We envisage the use of transient borrowing in the context of clusters of VM servers, as an efficient mechanism for elastic storage provisioning. Today, as VM servers (virtual servers) become more densely concentrated on individual higher-end hardware platforms, maintenance of the hardware and storage provisioning become difficult tasks. This is especially true for non-enterprise environments, where costly storage area networks or specialized network-attached storage devices are not available. While individual VMs may not require Five 9s availability, the overall criticality of a specific piece of hardware is increased by the number and diversity of the hosted VMs. 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. Additionally, there are cases where applications within a VM need storage for a brief time, yet would be willing to relinquish such storage for periods of time. If the frequency of this were high enough, an administrator would be tempted to permanently allocate the storage to the VM. Providing a type of elastic storage, possibly based on the transient borrowing model, has the potential to satisfy the need for elasticity of such requests while reducing the administrative overheads associated with frequent storage (de)allocation requests. Implementing this requires a relaxation of one of the design constraints introduced earlier, 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 transient borrowing primitives. To this point, we have only considered positioning the transient 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 transient 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 transient 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.

6. CONCLUSION

VM-based approaches that decouple user state from execution hardware have grown in importance over the past few years. These approaches preserve crisp responsiveness for interactive applications, which is the essence of personal computing, while allowing temporary use of any available hardware. Zero-install implementations of these approaches are particularly attractive because they require no software installation or configuration on borrowed hardware prior to use, and leave behind no modifications after use. We use the term *TransientPC systems* to refer to these approaches. Unfortunately, the performance characteristics of portable storage that underlie zero-install TransientPC solutions can severely limit system performance. This article shows how this problem can be solved by exploiting the unused parts of the disk on the host PC hardware, while preserving zero-install attributes. Our solution synthesizes a virtual disk from these free disk blocks, and uses it for the I/O-intensive aspects of VM-based mobility. Our experiments confirm that this approach can result in significant user-visible performance improvement.

From a broader perspective, our work addresses a long-standing anomaly in the transient use of hardware. 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.

In this context, TransPart can be viewed as a mechanism for transient borrowing of free disk blocks. 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. TransPart can be viewed as a lightweight mechanism that enforces this implicit owner intent.

ACKNOWLEDGMENTS

We thank the editor and the anonymous reviewers for their valuable comments and suggestions.

REFERENCES

William A. Arbaugh, David J. Farber, and Jonathan M. Smith. 1997. A secure and reliable bootstrap architecture. In *Proceedings of the IEEE Symposium on Security and Privacy (SP)*.

Micah Beck, Terry Moore, and James S. Plank. 2002. An end-to-end approach to globally scalable network storage. In *Proceedings of the ACM SIGCOMM Conference*.

Ramón Cáceres, Casey Carter, Chandra Narayanaswami, and Mandayam Raghunath. 2005. Reincarnating PCs with portable SoulPads. In *Proceedings of the 3rd International Conference on Mobile Systems, Applications, and Services (MobiSys)*.

Remy Card, Theodore Ts'o, and Stephen Tweedie. 1994. Design and implementation of the second extended filesystem. In *Proceedings of the 1st Dutch International Symposium on Linux*.

Ramesh Chandra, Nickolai Zeldovich, Constantine Sapuntzakis, and Monica S. Lam. 2005. The Collective: A cache-based system management architecture. In *Proceedings of the 2nd Symposium on Networked Systems Design and Implementation*.

James Cipar, Mark D. Corner, and Emery D. Berger. 2007. Contributing storage using the transparent file system. *ACM Trans. Storage 3*, 3.

CNN. 2010. Buggy McAfee update whacks Windows XP PCs - CNN.com. http://www.cnn.com/2010/TECH/04/22/cnet.mcafee.antivirus.bug/index.html.

Russell Coker. 2001. Bonnie++ home page. http://www.coker.com.au/bonnie++.

devmapper. 2001. Linux Device-Mapper. Device-Mapper resource page. http://sources.redhat.com/dm/.

dm-crypt. 2010. dm-crypt. dm-crypt: A device-mapper crypto target http://www.saout.de/misc/dm-crypt/.

Tal Garfinkel, Ben Pfaff, Jim Chow, Mendel Rosenblum, and Dan Boneh. 2003. Terra: A virtual machine-based platform for trusted computing. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP)*.

Scott Garriss, Ramón Cáceres, Stefan Berger, Reiner Sailer, Leendert van Doorn, and Xiaolan Zhang. 2008. Trustworthy and personalized computing on public kiosks. In *Proceedings of the 6th International Conference on Mobile Systems, Applications, and Services (MobiSys)*. ACM, New York, NY, 199–210. DOI:http://dx.doi.org/10.1145/1378600.1378623.

John H. Howard, Michael L. Kazar, Sherri G. Menees, David A. Nichols, M. Satyanarayanan, Robert N. Sidebotham, and Michael J. West. 1988. Scale and performance in a distributed file system. *ACM Trans. Comput. Syst. 6*, 1, 51–81. DOI:http://dx.doi.org/10.1145/35037.35059.

Sukwoo Kang and A. L. Narasimha Reddy. 2006. An approach to virtual allocation in storage systems. *ACM Trans. Storage 2*, 4.

Clare-Marie Karat, Christine Halverson, Daniel Horn, and John Karat. 1999. Patterns of entry and correction in large vocabulary continuous speech recognition systems. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI)*. ACM, New York, NY, 568–575. DOI:http://dx.doi.org/10.1145/302979.303160.

Jeffrey Katcher. 1997. PostMark: A new file system benchmark. Tech. rep. TR3022. Network Appliance. http://communities.netapp.com/servlet/JiveServlet/download/2609-1551/Katcher97-postmark-netapp-tr3022.pdf.

Michael Kozuch and M. Satyanarayanan. 2002. Internet suspend/resume. In *Proceedings of the 4th IEEE Workshop on Mobile Computing Systems and Applications*.

Butler Lampson, Martín Abadi, Michael Burrows, and Edward Wobber. 1992. Authentication in distributed systems: Theory and practice. *ACM Trans. Comput. Syst. 10*, 4, 265–310. DOI:http://dx.doi.org/10.1145/138873.138874.

Dennis Lee, Jean-Loup Baer, Brian Bershad, and Tom Anderson. 1999. Reducing startup latency in Web and desktop applications. In *Proceedings of the 3rd Conference on USENIX Windows NT Symposium (WINSYM)*. USENIX Association, Berkeley, CA, 17–17.

LibreOffice. 2013. LibreOffice.org - LibreOffice.org Project Home Page. http://www.libreoffice.org/.

Christopher R. Lumb, Jiri Schindler, and Gregory R. Ganger. 2002. Freeblock scheduling outside of disk firmware. In *Proceedings of the Conference on File and Storage Technologies (FAST)*.

LVM. 2008. LVM2. LVM2 Resource Page. http://sourceware.org/lvm2/.

Jonathan M. McCune, Yanlin Li, Ning Qu, Zongwei Zhou, Anupam Datta, Virgil Gligor, and Adrian Perrig. 2010. TrustVisor: Efficient TCB reduction and attestation. In *Proceedings of the IEEE Symposium on Security and Privacy (SP)*.

Jonathan M. McCune, Bryan J. Parno, Adrian Perrig, Michael K. Reiter, and Hiroshi Isozaki. 2008. Flicker: An execution infrastructure for TCB minimization. In *Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems (Eurosys)*.

J. Menon, D. A. Pease, R. Rees, L. Duyanovich, and B. Hillsberg. 2003. IBM storage tank – A heterogeneous scalable SAN file system. *IBM Syst. J. 42,* 2.

MokaFive. 2010. MokaFive home page. http://www.mokafive.com.

NTFSProgs. 2007. Linux-NTFS project. Linux-NTFS. http://www.linux- ntfs.org/.

Nishkam Ravi, Chandra Narayanaswami, Mandayam Raghunath, and Marcel Rosu. 2007. Towards securing pocket hard drives and portable personalities. *IEEE Pervasive Computing 6*, 4.

ReadyBoost. 2012. ReadyBoost - Microsoft Windows. http://windows.microsoft.com/en-US/windows7/products/features/readyboost.

Reiner Sailer, Xiaolan Zhang, Trent Jaeger, and Leendert van Doorn. 2004. Design and implementation of a TCG-based integrity measurement architecture. In *Proceedings of the 13th Conference on USENIX Security Symposium (SSYM)*. USENIX Association, Berkeley.

Constantine R. Sapuntzakis, Ramesh Chandra, Ben Pfaff, Monica S. Lain, Mendel Rosenblum, and Jim Chow. 2002. Optimizing the migration of virtual computers. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation*.

M. Satyanarayanan, Benjamin Gilbert, Matt Toups, Niraj Tolia, Ajay Surie, David R. O'Hallaron, Adam Wolbach, Jan Harkes, Adrian Perrig, David J. Farber, Michael A. Kozuch, Casey J. Helfrich, Partho Nath, and H. Andres Lagar-Cavilla. 2007. Pervasive personal computing in an Internet suspend/resume system. *IEEE Internet Comput. 11*, 2, 16–25.

Emin Gün Sirer, Willem de Bruijn, Patrick Reynolds, Alan Shieh, Kevin Walsh, Dan Williams, and Fred B. Schneider. 2011. Logical attestation: An authorization architecture for trustworthy computing. In *Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP)*.

Stephen Smaldone, Benjamin Gilbert, Nilton Bila, Liviu Iftode, Eyal de Lara, and M. Satyanarayanan. 2009. Leveraging smart phones to reduce mobility footprints. In *Proceedings of the 7th International Conference on Mobile Systems, Applications, and Services (MobiSys)*. ACM, New York, NY, 109–122. DOI:http://dx.doi.org/10.1145/1555816.1555828.

Craig A. N. Soules, Garth R. Goodson, John D. Strunk, and Gregory R. Ganger. 2003. Metadata efficiency in versioning file systems. In *Proceedings of the 2nd USENIX Conference on File and Storage Technologies (FAST)*. USENIX Association, Berkeley, CA, 43–58.

Ajay Surie, Adrian Perrig, M. Satyanarayanan, and David J. Farber. 2007. Rapid trust establishment for pervasive personal computing. *IEEE Pervasive Comput. 6,* 4, 24–30. DOI:http://dx.doi.org/10.1109/MPRV.2007.84.

Richard Ta-Min, Lionel Litty, and David Lie. 2006. Splitting interfaces: Making trust between applications and operating systems configurable. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI)*.

Niraj Tolia, Jan Harkes, Michael Kozuch, and M. Satyanarayanan. 2004. Integrating portable and distributed storage. In *Proceedings of the 3rd USENIX Conference on File and Storage Technologies (FAST)*. USENIX Association, Berkeley, CA, 227–238.

Avishay Traeger, Erez Zadok, Nikolai Joukov, and Charles P. Wright. 2008. A nine year study of file system and storage benchmarking. *Trans. Storage 4*, 2, 1–56. DOI:http://dx.doi.org/10.1145/1367829.1367831.

TrueCrypt. 2010. TrueCrypt. Free open-source disk encryption software for Windows 7/Vista/XP, Mac OS X, and Linux. http://www.truecrypt.org/.

Theodore Ts'o. 2010. E2fsprogs: Ext2/3/4 Filesystem utilities. http://e2fsprogs.sourceforge.net/.

YUM. 2010. YUM - Yellowdog Updater Modified. YUM Package Manager Project Home Page. http://yum.baseurl.org.

Received April 2012; revised December 2012; accepted March 2013