Highly Reliable Mobile Desktop Computing in Your Pocket

Shaya Potter
Department of Computer Science
Columbia University
New York, NY USA
spotter@cs.columbia.edu

Jason Nieh
Department of Computer Science
Columbia University
New York, NY USA
nieh@cs.columbia.edu

Abstract

We present DeskPod, a portable system that provides a highly reliable desktop computing environment for mobile users by leveraging rapid improvements in capacity, cost, and size of portable storage devices. DeskPod enables a user's live computing environment to be suspended to portable storage, carried around, easily copied for fault-resilience, and resumed from the storage device to provide the user with the same persistent, personalized computing environment on another computer. DeskPod achieves this by providing a virtualization and checkpoint/restart mechanism that decouples a desktop computing environment from any single hardware device so that it can be stored and executed anywhere, improving desktop computing reliability by eliminating a potential single point of failure. We have implemented a Linux DeskPod prototype and demonstrate its ability to quickly suspend and resume desktop sessions, enabling a seamless mobile experience.

1. Introduction

In today's world of commodity computers, computers are truly ubiquitous. Users make use of computers at home, school, work, and on the road. However, the current personal computer framework ties a user's data, applications and execution state to a single underlying machine. While a user has access to many different machines, such as at the office and at home, each machine may provide a different computing environment. These different computing environments can provide the user with differing sets of applications as well as variable access to the data a user needs to complete a task. As computers can be prone to failure, applications and data tied to any particular machine can be unexpectedly rendered unavailable. This reliability problem is exacerbated for laptops, which often operate in more hostile working environments and are more prone to being stolen or to fail given the complexity of these devices.

To address these problems, we present DeskPod, a portable system that decouples a desktop computing environment from any single hardware device so that it can be stored and executed anywhere. DeskPod builds on our previous work on WebPod [8, 10] and leverages commodity pocketable storage devices, ranging from flash memory sticks with no moving parts that can hold 4 GB of data to portable hard disks, such as Apple's iPod, that can hold 60 GB and more of data. DeskPod enables these devices to hold a user's entire computing environment including applications, data and execution state so that it can be checkpointed to portable storage, carried around easily and restarted from the storage device on a completely different computer. Since all applications, data and execution state are stored on the portable storage device, DeskPod does not rely on any application resources or data stored on the host machine. This enables users to continue working even in the face of faulty components as they can simply move their DeskPod environment to another host machine, or create another copy on another storage device. DeskPod also enables mobile users to obtain the same persistent, personalized desktop computing experience from any computer.

DeskPod operates by encapsulating a user's desktop computing session in a virtualized execution environment and storing all state associated with the session on the portable storage device. This enables DeskPod to be easily backed up when it is plugged into a user's primary computer by replicating all recent changes to DeskPod's file system. If the device is lost, users can then simply replicate the file system onto a new device and continue their work. DeskPod virtualization decouples the desktop applications from the underlying operating system environment by introducing a private virtual namespace that provides consistent, host-independent naming of system resources. DeskPod also virtualizes the display so that the desktop session can be scaled to different display resolutions that may be available as a user moves from one computer to another. This enables a desktop computing session to run in the same way on any host despite differences that may exist among

Proceedings of the 30th Annual International Computer Software and Applications Conference (COMPSAC'06)
0-7695-2655-1/06 $20.00 © 2006 IEEE

different host operating system environments and display hardware. Furthermore, DeskPod virtualization protects the underlying host from untrusted applications that a user may run as part of a regular desktop session. DeskPod provides this functionality without modifying, recompiling or relinking any applications or the operating system kernel.

We have implemented a DeskPod prototype for commodity PCs running Linux and measured its performance. Our experimental results with real applications demonstrate that DeskPod has very low virtualization overhead and can checkpoint and restart desktop sessions with sub-second latencies. We show that DeskPod can reconstitute a user's desktop session more than an order of magnitude faster than if a user had to reopen the same set of applications without DeskPod. Our results also show that a complete DeskPod desktop computing session including file system state requires less than 512 MB of storage. DeskPod's modest storage requirements enable it to be used with the small form factor USB drives available on the market today and can be conveniently carried on a key chain or in a user's pocket.

This paper presents the design and implementation of DeskPod. Section 2 presents the overall DeskPod architecture and usage model. Section 3 describes DeskPod virtualization. Section 4 describes the DeskPod checkpoint/restart mechanisms that enable DeskPod to be used across operating system environments with different kernel versions. Section 5 presents experimental results measuring the performance of the DeskPod system. Section 6 discusses related work. Finally we present some concluding remarks.

2. DeskPod Usage Model

DeskPod is a simple end user device that users can carry in their pockets. A DeskPod session can be easily populated with the complete set of applications used in a user's normal desktop environment so that environment is available on any computer. To the user, a DeskPod session does not appear to be any different from a private computer even though it runs on a host that may be running other applications. Those applications running outside of the DeskPod session are not visible to a user within a DeskPod session. To provide strong security, the DeskPod stores the session on an encrypted file system. Therefore, even if the DeskPod device is lost or stolen, an attacker will not have any access to the data stored on it.

A user starts DeskPod by plugging in a DeskPod portable storage device into the computer. The computer detects the device and automatically tries to restart the DeskPod session. This involves first authenticating the user by asking for a password or by using biometric technology in the form of built-in fingerprint readers available on some USB drives [13]. Once a user is authorized, DeskPod mounts its encrypted file system, restarts its desktop computing session, and attaches a DeskPod viewer to the session to make the DeskPod session and it's applications available and visible to the user. Applications running in a DeskPod session appear to the underlying operating system just like other applications that are executing, and therefore they make use of the host's network interface in a normal manner.

Once DeskPod is started, a user can continue to use their applications, data and the saved execution state. When the user wants to leave the computer, the user simply closes the DeskPod viewer. This causes the DeskPod session to be quickly checkpointed to the DeskPod storage device, which can then be unplugged and carried around by the user. When another computer is ready to be used, the user simply plugs in the DeskPod device and the session is restarted right where it left off. Since DeskPod saves and restores the execution state, there is no need for a user to manually launch the applications and reload documents. DeskPod's checkpoint/restart functionality maintains a user's desktop computing session persistently as a user moves from one computer to another, even including ephemeral state such as copy/paste buffers. If the host machine used for DeskPod crashes, this will take down the current DeskPod session with it. However, since DeskPod doesn't provide its own operating system, it can be simply plugged into a new host machine and to start a fresh DeskPod session. Similar to a regular computer, only data that was not committed to disk when the host machine crashed is lost.

DeskPod only provides a desktop computing environment, not an entire operating system environment. There is no operating system installed on the DeskPod device. DeskPod instead makes use of the operating system environment available on the host computer into which it is plugged in. This provides two important benefits in terms of startup speed and management complexity. Since there is no operating system on the DeskPod device, there is no need to boot or configure a new operating system environment to use DeskPod. Since only DeskPod applications need to be restarted, this minimizes startup costs for using DeskPod and ensures that DeskPod can be used on any machine on which a compatible operating system is running. Furthermore, since DeskPod does not provide an operating system there is no need for DeskPod users to maintain and manage an operating system environment, reducing management complexity thereby increasing reliability.

DeskPod protects desktop computing sessions by isolating each session in its own private execution environment. Other user-level applications running on the same machine are not able to access any state associated with a DeskPod session, protecting the privacy of a DeskPod user. Deskpod does rely on the host hardware and operating system kernel, as is common practice for users today. As a result, it does not protect users from attacks that may arise from tampered hardware or a compromised operating system kernel.

DeskPod provides a consistent usage environment that is compatible with the regular desktop computing model. Because DeskPod is small and travels with the user, there is a risk of loss of the device and its associated data. DeskPod state is already encrypted on the storage device to minimize the damage suffered if the device is lost or stolen. To reduce the risk of data loss further, DeskPod is backed up periodically when it returns to the user's own computer, in the same manner as a user synchronizes a PDA. If the DeskPod device is lost, the user's desktop computing session can be easily restored from backup onto another device.

3. DeskPod Virtualization

DeskPod virtualizes the underlying host operating system and display to decouple DeskPod desktop computing sessions from the underlying host used for execution. This is essential to allow DeskPod applications to be isolated from the underlying system and other applications, to be checkpointed on one machine and restarted on another, and to be displayed on hosts with different display hardware and display resolution. Given the large existing base of desktop applications and commodity operating systems, DeskPod virtualization is designed to be completely transparent to work with existing unmodified applications and operating system kernels.

3.1. Operating System Virtualization

Operating system virtualization is necessary to ensure that operating system resource identifiers, such as process IDs (PIDs), remain constant throughout the life of a process to ensure its correct operation. Applications commonly manipulate these operating system resource identifiers as they execute. However, these identifiers are only unique to a particular operating system instance. When an application and its associated processes is moved from one computer to another, there is no guarantee that the destination operating system can provide the same identifiers to the migrated process. Those identifiers may already be in use by other processes running on the destination system, preventing the migrated process from executing correctly.

DeskPod virtualizes the underlying host operating system by encapsulating desktop computing sessions within a host independent, virtualized view of the operating system. DeskPod virtualization provides each desktop computing session with its own virtual private namespace. For example, a DeskPod session contains its own host independent view of operating system resources, such as PID/GID, IPC, memory, file system, and devices. The namespace is the only means for the processes associated with running DeskPod application instances to access the underlying operating system. DeskPod introduces this namespace to decouple processes associated with applications running in DeskPod sessions from the underlying host operating system.

The namespace is private in that only processes within the session can see the namespace, and the namespace in turn masks out resources that are not contained in the session. Processes inside the session appear to one another as normal processes, and they are able to communicate using traditional Inter-Process Communication (IPC) mechanisms. On the other hand, no IPC interaction is possible across the session's boundary, because outside processes on the DeskPod host are not part of the private namespace. Processes inside a session and those outside of it are only able to communicate network communication mechanisms, traditionally used to communicate across computers. As a result, processes within the namespace are isolated from processes outside of the namespace as though those within the namespace were running on a private computer.

The namespace is virtual in that all operating system resources, including processes, user information, files, and devices, are accessed through virtual identifiers. Virtual identifiers are distinct from the host-dependent, physical resource identifiers used by the operating system. Virtual identifiers remain consistent throughout a process's and session's lifetimes and are used to provide a host-independent view of the system. Since the session's namespace is separate from the underlying host namespace, it can preserve naming consistency for its processes, even if the underlying operating system instance changes.

The DeskPod namespace enables DeskPod processes to be isolated from the host system, checkpointed to its storage device, and transparently restarted on another machine. The private virtual namespace provides consistent, virtual resource names for DeskPod processes enabling DeskPod sessions to migrate from one machine to another. Names within a session are assigned in a unique manner in the same way that traditional operating systems assign names, but such names are localized to the session. Since the namespace is virtual, there is no need for it to change when the session is migrated, ensuring that identifiers remain constant throughout the life of the process, as required by applications that use such identifiers. Since the namespace is private to the DeskPod session, processes within the session can be migrated as a group, while avoiding resource naming conflicts among other processes running on the host.

The private virtual namespace also enables the DeskPod session to be securely isolated from the host by providing complete mediation to all operating system resources. Since the only resources within the DeskPod session are accessible to the owner of the session, a compromised session is preventing from causing an effect on any resources, such as processes or the file system, that exist outside of the session. Similarly, since any attempts by processes outside of the session to interact with processes running inside of the

session must also occur through operating system resources, DeskPod is able to filter out those interactions as well.

DeskPod virtualizes the operating system instance by using mechanisms that translate between the session's virtual resource identifiers and the operating system resource identifiers. For every resource accessed by a process in a session, the virtualization layer associates a *virtual name* to an appropriate operating system *physical name*. When an operating system resource is created for a process in a session, the physical name returned by the system is caught, and a corresponding private virtual name created and returned to the process. Similarly, any time a process passes a virtual name to the operating system, the virtualization layer catches and replaces it with the corresponding physical name. The key virtualization mechanisms used are a system call interposition mechanism and the `chroot` utility with file system stacking for file system resources.

DeskPod virtualization uses system call interposition to virtualize operating system resources, including process identifiers, keys and identifiers for IPC mechanisms such as semaphores, shared memory, and message queues, and network addresses. System call interposition wraps existing system calls to check and replace arguments that take virtual names with the corresponding physical names, before calling the original system call. Similarly, wrappers are used to capture physical name identifiers that the original system calls return, and return corresponding virtual names to the calling process running inside the session. Session virtual names are maintained consistently as a session migrates from one machine to another and are remapped appropriately to underlying physical names that may change as a result of migration. Session system call interposition also masks out processes inside of a session from processes outside of the session to prevent any interprocess host dependencies across the session boundary.

DeskPod virtualization employs the `chroot` utility and file systems stacking to provide each session with its own file system namespace. The DeskPod session's file system is totally contained within its portable storage device, which guarantees that the same files can be made consistently available as the session is migrated from one computer to another. More specifically, when a DeskPod session is created or restarted on a host, a private directory is created in the host. This directory serves as a staging area for the session's virtual file system. Within the directory, the session's file system will be mounted from the device. The `chroot` system call is then used to set the staging area as the root directory for the session, thereby achieving file system virtualization with negligible performance overhead. This method of file system virtualization provides an easy way to restrict access to files and devices from within a session. This can be done by simply not including file hierarchies and devices within the session's file system namespace. If files and devices are not mounted within the session's virtual file system, they are not accessible to the session's processes.

Commodity operating systems are not built to support multiple namespaces securely. File system virtualization must address the fact that there are multiples ways to break out of a chrooted environment, especially when the `chroot` system call is allowed to be used. The primary way DeskPod provides security is by disallowing the privileged root user from being used within the session. The DeskPod session's file system virtualization also enforces the chrooted environment and ensures that the session's file system are the only files accessible to processes within session, by using a simple form of file system stacking to implement a barrier. This barrier directory prevents processes within the session from traversing it. Since the processes are not allowed to traverse the directory, they are unable to access files outside of the session's file system namespace. Therefore, by combining the inability for DeskPod processes to access any files outside of the DeskPod storage device's file system, as well as the inability for the processes to run with privilege, the processes are confined to the DeskPod session and can't affect change on the DeskPod host.

3.2. Display Virtualization

Display virtualization is necessary to ensure that applications can be properly redisplayed when they are moved from one computer to another. This requires that all of an application's display state be captured. Modern graphical applications display their output to a window system, which then processes the display commands to a video device driver to be rendered to the computer's framebuffer so that it appears on the computer's screen. As a result, the display state associated with an application is distributed at different times between the window system and the hardware framebuffer. Since a window system may contain display state for many applications, identifying and extracting the display state for a particular application in an application transparent manner is difficult. Since framebuffer hardware varies from one system to another, extracting the display state for a particular application from a particular framebuffer in a manner that is portable across different systems is also difficult given that such state is often tied closely to the specifics of the particular physical display device used.

DeskPod virtualizes the display associated with a desktop computing session so that it can be viewed on different hosts that may have different display systems available. Building on previous work by one of the authors [1], DeskPod virtualization provides each desktop computing session with its own virtual display server and virtual device driver to decouple the display of the desktop computing session from the display subsystem of the host. The virtual dis-

play server provides a DeskPod session with its own window system separate from the window system on the host, thereby separating DeskPod application display state from other applications running on the host outside of the DeskPod session. The display server is considered a part of the DeskPod session and is checkpointed when the DeskPod session is suspended and restarted when the DeskPod session is resumed. Our DeskPod prototype implementation uses an XFree86 server as its own display server.

Instead of rendering display commands to a real device driver associated with a physical display device on the host, the virtual display server directs its commands to a virtual device driver representing a virtual display device associated with the DeskPod session. The virtual display device processes display commands and directs their output to memory instead of a framebuffer. This approach abstracts away the specific implementation of video card features into a high-level view that is applicable to all video cards. Since the device state is not in the physical device but in the virtualized DeskPod session, this simplifies display state management during checkpointing and restarting a DeskPod session. As a result, checkpointing the display state of a DeskPod session can be done by simply saving the associated memory instead of needing to extract display state from the host-specific framebuffer.

DeskPod's video hardware layer approach, allows it to take full advantage of existing infrastructure and application-hardware interfaces. Furthermore, new video hardware features can be supported with at most the same amount of work necessary for supporting them in traditional desktop display drivers. While there is some loss of semantic display information at the low-level video device driver interface, our experiments with desktop applications indicate that the vast majority of application display commands issued can be mapped directly to standard video hardware primitives. For example, DeskPod provides direct video playback support by leveraging existing application interfaces and standard YUV video formats natively supported by almost all off-the-shelf video cards available today. Video data can simply be transferred from the DeskPod's virtual display driver to the host's video hardware, which automatically does inexpensive, high speed, color space conversion and scaling.

Rather than sending display commands to local display hardware, the DeskPod virtual video driver packages up display commands associated with a user's computing session, writes them to memory, and enables them to be viewed using a DeskPod viewer application that runs in the context of the window system on the host. The viewer is completely decoupled though from the rest of the DeskPod display system. All it does is read the persistent display state managed by the DeskPod display system. The viewer can be disconnected and reconnected to the DeskPod session at any time without loss of display information since it does not maintain any persistent display state. To enable DeskPod to be viewed on hosts with different display resolutions, the DeskPod viewer is resolution independent and can scale the display to any size.

4. DeskPod Checkpoint/Restart

DeskPod combines its virtualization with a checkpoint-restart mechanism [9] that allows the DeskPod device to be checkpointed, transported and restarted across computers with different hardware and operating system kernels. DeskPod is limited to migrating between machines with a common CPU architecture, and where kernel differences are limited to maintenance and security patches. These patches often correspond to changes in minor version numbers of the kernel. For example, the Linux 2.4 kernel has more than 30 minor versions.

Migration is limited to these kinds of minor kernel version changes because major version changes are allowed to break application compatibility, which may cause running processes to break. Even with minor versions changes, there can be significant changes in kernel code. For example, during the Linux 2.4 series of kernels, the entire VM subsystem was extensively modified to change the page replacement mechanism. Similarly, migration is limited to scenarios where the application's execution semantics, such as how threads are implemented or dynamic linking is performed, stay constant. On the Linux kernel, this is not an issue as these semantics are enforced by user-space libraries. Since the session's user-space libraries migrate with it, the semantics stay constant.

To support migration across different kernels, DeskPod's checkpoint-restart mechanism employs an intermediate format to represent the state that needs to be saved. Although the internal state that the kernel maintains on behalf of processes can be different across kernels, the high-level properties of the process are much less likely to change. DeskPod captures the state of a process in terms of this higher-level semantic information rather than the kernel specific data. For example, part of the state associated with a Unix socket connection consists of the directory entry of the socket, its superblock information, and a hash key. It may be possible to save all of this state in this form and successfully restore on a different machine running the same kernel. But this representation is of limited portability across different kernels. On the other hand, a high-level representation consisting of a four tuple: {virtual source pid, source fd, virtual destination pid, destination fd}, is highly portable. This is because the semantics of a process identifier and a file descriptor are standard across different kernels.

DeskPod's intermediate representation format is chosen such that it offers the degree of portability needed for mi-

grating between different kernel minor versions. If the representation of state is too high-level, the checkpoint-restart mechanism could become complicated and impose additional overhead. For example, the DeskPod system saves the address space of a process in terms of discrete memory regions called VM areas. As an alternative, it may be possible to save the contents of a process's address space and denote the characteristics of various portions of it in more abstract terms. However, this would call for an unnecessarily complicated interpretation scheme and make the implementation inefficient. The VM area abstraction is standard even across major Linux kernel revisions. DeskPod views the VM area abstraction as offering sufficient portability in part because the organization of a process's address space in this manner has been standard across all Linux kernels and has never changed since its inception.

DeskPod leverages high-level native kernel services in order to transform the intermediate representation of the checkpointed image into the internal state required by the target kernel. Continuing with the previous example, DeskPod restores a Unix socket connection using high-level kernel functions as follows. First, two new processes are created with virtual PIDs as specified in the four tuple. Then, each one creates a Unix socket with the specified file descriptor and one socket is made to connect to the other. This procedure effectively recreates the original Unix socket connection without depending on many internal kernel details.

This use of high-level functions helps with general portability when using DeskPod for migration. Security patches and minor version kernel revisions commonly involve modifying the internal details of the kernel while high-level primitives remain unchanged. As such, high-level functions are usually made available to kernel modules through the exported kernel symbol interface. Therefore, the DeskPod system is able to perform cross-kernel migration without requiring modifications to the kernel.

To eliminate possible dependencies on low-level kernel details, DeskPod's checkpoint-restart mechanism requires processes to be suspended prior to being checkpointed. Suspending processes creates a quiescent state necessary to guarantee the correctness of the checkpointed image, and it also minimizes the amount of information that needs to be saved. As a representative example, consider the case of semaphore wait queues. Although semaphore values can be easily obtained and restored through well known interfaces, saving and restoring the state of the wait queue involves the manipulation of kernel internals. However, DeskPod is able to take advantage of existing semantics which direct the kernel to release a process from a wait queue upon receipt of a signal. DeskPod uses this semantic to empty the wait queues by suspending all processes with a signal, and therefore avoids having to save the state of the queue.

Finally, we must ensure that any changes in the system call interfaces are properly accounted for. As DeskPod has a virtualization layer that uses system call interposition to maintain namespace consistency, a change in the semantics for any system call intercepted could be an issue in migrating across different kernel versions. But such changes usually do not occur, as it would require system libraries to be rewritten. In other words, DeskPod virtualization is protected from such changes in the same way legacy applications are protected. However, new system calls are added from time to time. Such system calls could have implications to the encapsulation mechanism. For instance, across all Linux 2.4 kernels, there were two new system calls that used identifiers that needed to be intercepted and virtualized, `gettid` and `tkill`.

Since processes within a DeskPod session only have access to devices through the virtual device drivers provided by the DeskPod, it makes it simple to checkpoint the device specific data associated with the processes. In particular, since the DeskPod display system is built using its own virtual display device driver which is not tied to any specific hardware device, such virtual device state can be more easily checkpointed. Because the virtual device state is totally stored in regular memory, it is a simple matter of saving that state on checkpoint and restoring it on restart. When the DeskPod viewer on the host reconnects to the virtual display driver, it is able to display the user's session.

5. Experimental Results

We implemented DeskPod as three components, a simple viewer application for accessing a DeskPod desktop computing session, an unmodified XFree86 display server with a DeskPod virtual display device driver, and a loadable kernel module in Linux that requires no changes to the Linux kernel. We present some experimental results using our Linux prototype to quantify the overhead of using the DeskPod environment on various applications.

Experiments were conducted on three IBM PC machines, each with a 933 MHz Intel Pentium-III CPU and 512 MB RAM. The machines each had a 100 Mbps NIC and were connected to one another via 100 Mbps Ethernet and a 3Com Superstack II 3900 switch. Two machines were used as hosts for running DeskPod and the other was used as a web server for measuring web benchmark performance. To demonstrate the ability of DeskPod to operate across different operating system distributions and kernels, each machine was configured with a different Linux distribution and Linux kernel version. One machine ran Debian Stable with a Linux 2.4.5 kernel and the other running Debian Unstable with a Linux 2.4.18 kernel.

We used a 40 GB Apple iPod as the DeskPod portable storage device, though a much smaller USB memory drive

Checkpoint	Restart	Startup
0.9s	0.9s	19s

Table 1. Checkpoint/Restart vs. Startup Time

could have been used. Each PC machine provided a FireWire connection which could be used to connect to the iPod. We built an unoptimized DeskPod file system by bootstrapping a Debian GNU/Linux installation onto the iPod and installing the appropriate packages needed for a regular desktop environment, while removing the extra packages needed to boot a full Linux system as DeskPod is just a lightweight desktop computing environment, not a full operating system. This resulted in a 418 MB file system image. This easily fits in the iPod with plenty of storage capacity to spare, and also easily fits in common USB memory drives. Our unoptimized DeskPod file system could be even smaller if the file system was built from scratch instead by just installing the exact programs and libraries that are needed.

To measure the cost of DeskPod virtualization, we measured the performance of a few desktop applications on both our Linux DeskPod prototype and a vanilla Linux system. We ran two application workloads, a web browsing benchmark based on a modified version of the Web Text Page Load test from the Ziff-Davis i-Bench 1.5 benchmark suite [6], and a Linux kernel compilation with up to ten processes active at one time. Our results for both of these real applications show that running them in the DeskPod environment vs running in vanilla Linux from local SCSI storage incurs negligible performance overhead. We have previously conducted additional measurements using various micro-benchmarks and demonstrated that the virtualization overhead incurred by DeskPod on most micro-benchmarks is less than 10% [10].

To measure the cost of checkpointing and restarting DeskPod sessions as well as demonstrating DeskPod's ability to improve the way a user works with desktop applications, we migrated a DeskPod session between the two machines described above. The DeskPod session included the KWrite Word processor, KSpread spreadsheet and Konqueror web browser each with a document loaded as well as a Konsole terminal. This is indicative of a regular user desktop computing environment. We compare how long it takes to restart this environment against starting a new desktop session without any of the desktop applications loaded.

Table 1 shows that it is significantly faster to checkpoint and restart a DeskPod desktop session than it is to have to start the same kind of desktop computing session from scratch. Checkpointing and restarting a DeskPod takes under a second. This enables a DeskPod user to very quickly disconnect from a machine and plug-in to another machine and immediately start working again. Furthermore, the fact that these experiments were run across two different machines with two different operating system environments and kernels demonstrates the ability of DeskPod to work across different software environments.

In contrast, Table 1 shows that starting a desktop computing session the traditional way of starting an X server and having it load a desktop environment without any application loaded is a much slower 19 seconds. Table 1 shows that checkpointing and restarting a DeskPod desktop computing session with actual applications running is faster than just starting a plain desktop environment.

The DeskPod only needed 50 MB of storage to checkpointing the entire DeskPod session without applying any compression techniques to reduce the image size. These results show that the checkpointed state that needs to be saved is very modest and easy to store on any portable storage device. Given the modest size of the checkpointed images, there is no need for any additional compression which would reduce the minimal storage demands but add additional latency due to the need to compress and decompress the checkpointed images. Our results show that total storage requirement of a basic DeskPod, including both the checkpointed image size and the file system size, is much less than what can fit in a small 512 MB USB drive.

6. Related Work

DeskPod builds upon our previous work on WebPod [8, 10], which first introduced a virtualized computing environment stored on a portable storage device in the context of web applications. DeskPod builds on WebPod and expands the utility of its architecture to support general desktop applications in addition to just web browsing. DeskPod and WebPod build on the previous work of one of the authors on Zap [7], which introduced operating system virtualization to support transparent migration. The operating system virtualization and checkpoint-restart mechanisms used in DeskPod and WebPod also build on our previous work on PeaPod [11] and AutoPod [9] which consider migration across different kernel versions in the context of system security and maintenance, respectively. The display virtualization mechanism used in DeskPod builds on the previous work of one of the authors on THINC [1].

Environments such as U3 [14] and Ceedo [3] are attempts to create a general application framework for applications to use to run on USB drives. While this enables a user to carry one's data and applications with them, it does not provide DeskPod's ability to carry the application state a well. Unlike U3, Ceedo does not require applications to be rewritten. However, it only virtualizes access to the Windows registry, thereby allowing applications to store data on the USB disk by default, but also enabling the applications to have full access to the underlying host machine.

Other approaches such as portable virtual appliances [4] and SoulPad [5], have used virtual machine monitors (VMMs) together with portable storage to provide a solution similar to DeskPod. Since these approaches are based on VMMs, they inherently tie processes to a complete operating system instance which can be more complex to manage than DeskPod. Furthermore, these approaches require taking over the entire host computer. For example, SoulPad has orders of magnitude higher startup latency because of the need to startup SoulPad's host operating system.

Thin-client systems [1, 2, 12] provide an attractive alternative usage model that also provides many of the benefits of DeskPod. Unlike DeskPod which is self-contained and runs locally on the host where the user is currently located, thin clients require network access to additional server infrastructure.

7. Conclusions

We have created DeskPod, a portable system that introduces a new form of fault-resilient computing. It enables users to decouple their computing experience from multiple pieces of hardware that are prone to faults, while providing them with the same persistent, personalized desktop computing session they expect from a regular machine. DeskPod allows an entire desktop computing session to be stored on a small portable storage device that can be easily carried on a key chain or in a user's pocket, thereby allowing the user increased mobility.

DeskPod provides its functionality by virtualizing operating system and display resources, decoupling the desktop computing session from the host on which it is currently running. DeskPod virtualization works together with a checkpoint/restart mechanism to enable DeskPod users to suspend their desktop computing sessions, move around, and resume their respective sessions at a later time on any computer right where they left off. DeskPod's ability to migrate desktop computing sessions between differently configured and administered computers provides for increased fault-resilience in the face of faulty computers.

We have implemented and evaluated the performance of a DeskPod prototype in Linux. Our implementation demonstrates that DeskPod supports desktop applications without any changes to the applications or the underlying host operating systems kernels. Our experimental results with real desktop applications shows that DeskPod has low virtualization overhead and can migrate desktop computing sessions with sub-second checkpoint/restart times, providing superior fault-resilience than other proposed solutions. DeskPod is unique in its ability to provide a complete, persistent, and consistent desktop computing environment among multiple disparate fault-prone computers.

8. Acknowledgments

This work was supported in part by a DOE Early Career Award, NSF grants CNS-0426623 and ANI-0240525, and an IBM SUR Award.

References

[1] R. Baratto, J. Nieh, and L. Kim. THINC: A Remote Display Architecture for Thin-Client Computing. Technical Report CUCS-027-04, Department of Computer Science, Columbia University, July 2004.

[2] R. Baratto, S. Potter, G. Su, and J. Nieh. MobiDesk: Mobile Virtual Desktop Computing. In *Proceedings of the Tenth Annual ACM International Conference on Mobile Computing and Networking (MobiCom 2004)*, pages 1–15, Philadelphia, PA, Sept. 2004.

[3] Ceedo. `http://www.ceedo.com`.

[4] R. Chandra, N. Zeldovich, C. Sapuntzakis, and M. S. Lam. The Collective: A Cache-Based System Management Architecture. In *Proceedings of the Second Symposium on Networked Systems Design and Implementation (NSDI 2005)*, Boston, MA, May 2005.

[5] R. Cceres, C. Carter, C. Narayanaswami, and M. Raghunath. Reincarnating PCs with Portable SoulPads. In *MobiSys 05: The Third International Conference on Mobile Systems, Applications, and Services*, Seattle, WA, June 2005.

[6] i-Bench version 1.5. `http://etestinglabs.com/benchmarks/i-bench/i-bench.asp`.

[7] S. Osman, D. Subhraveti, G. Su, and J. Nieh. The Design and Implementation of Zap: A System for Migrating Computing Environments. In *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI 2002)*, pages 361–376, Boston, MA, Dec. 2002.

[8] S. Potter and J. Nieh. WebPod: Persistent Web Browsing Sessions with Pocketable Storage Devices. Technical Report CUCS-047-04, Department of Computer Science, Columbia University, Nov. 2004.

[9] S. Potter and J. Nieh. Reducing Downtime Due to System Maintenance and Upgrades. In *Proceedings of the 19th Large Installation System Administration Conference (LISA 05)*, San Diego, CA, Dec. 2005.

[10] S. Potter and J. Nieh. WebPod: Persistent Web Browsing Sessions with Pocketable Storage Devices. In *Proceedings of the 14th International World Wide Web Conference (WWW 2005)*, Chiba, Japan, May 2005.

[11] S. Potter, J. Nieh, and D. Subhraveti. Secure Isolation and Migration of Untrusted Legacy Applications. Technical Report CUCS-005-04, Department of Computer Science, Columbia University, Jan. 2004.

[12] T. Richardson, Q. Stafford-Fraser, K. R. Wood, and A. Hopper. Virtual Network Computing. *IEEE Internet Computing*, 2(1):33–38, Jan-Feb 1998.

[13] Trek Thumbdrive TOUCH. `http://www.thumbdrive.com/touch.htm/`.

[14] U3 Platform. `http://www.u3.com`.