WebPod: Persistent Web Browsing Sessions with Pocketable Storage Devices

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 WebPod, a portable system that enables mobile users to use the same persistent, personalized web browsing session on any Internet-enabled device. No matter what computer is being used, WebPod provides a consistent browsing session, maintaining all of a user's plugins, bookmarks, browser web content, open browser windows, and browser configuration options and preferences. This is achieved by leveraging rapid improvements in capacity, cost, and size of portable storage devices. WebPod provides a virtualization and checkpoint/restart mechanism that decouples the browsing environment from the host, enabling web browsing sessions to be suspended to portable storage, carried around, and resumed from the storage device on another computer. WebPod virtualization also isolates web browsing sessions from the host, protecting the browsing privacy of the user and preventing malicious web content from damaging the host. We have implemented a Linux WebPod prototype and demonstrate its ability to quickly suspend and resume web browsing sessions, enabling a seamless web browsing experience for mobile users as they move among computers.

Categories and Subject Descriptors

D.4.5 [**Operating Systems**]: Reliability—*checkpoint/restart*; C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed applications*

General Terms

Design, Experimentation, Measurement, Performance

Keywords

web browsing, checkpoint/restart, portable storage, virtualization, process migration

1. INTRODUCTION

In today's world of commodity computers and broadband network connectivity, computer users are more mobile than ever. Users make use of computers at home, school and work. Computers are so much a part of daily life that many pervasive devices, such as cell phones and PDAs, are assimilating usage patterns, such as web browsing, e-mail and instant messaging, that were once limited to regular desktop computers.

A key problem that mobile users encounter is that they lack a common environment as they move around. The computer at the office is configured differently from the computer at home, which is different from the computer at the library. These locations can have different sets of software installed, which can make it difficult for a user to complete a task as the necessary software might not be available. Similarly, mobile users want consistent access to their files, which is difficult to guarantee as they move around.

Given the ubiquity of web browsers on modern computers, many traditional applications are becoming web-enabled. Common applications such as e-mail [4, 5] and instant messaging [1] have been ported to a web services environment that is usable from within a simple web browser. The advantage of this approach is that users effectively store their data on centrally managed servers which can be accessed wherever they go on any networked computer.

However, even by making applications accessible from a web browser, users are still constrained in their ability to browse effectively from any computer because of important missing data commonly stored in web browsers. This data includes bookmarks, cookies, and browser history, which enable web browsers to function in a much more useful manner. The problem that occurs when a user moves between computers is that this data, which is specific to a web browser installation, cannot move with the user.

From a security point of view, this state also provides a large trail of bread crumbs that a malicious user can try to make use of after a user has finished using a computer. From cookies, to page view history, and the web browser's web page cache, a record of usage can be recorded by a web browser not under the user's control. Users who have reason to want to protect their privacy, would be wary to use applications, such as web browsers, that record state and are not under their control.

Mobile users are also inconvenienced when attempting to use web browsers when they are moving around, especially when they pick up and move on short notice. While many web based applications are stateless, such as instant messaging, others, such as e-mail, contain important state, such as messages a user is in the process of composing. While many e-mail applications support the ability to save draft e-mail messages, this does not occur automatically. Similarly, while a user can attempt to bookmark all the pages he is looking at, he cannot restart his state as it was when he resumes it on a new computer.

Web users also depend on an assorted set of applications to be available on the computers they are using, such as Adobe Acrobat Reader for viewing PDF files. If the application is already installed on the host, the web browser can make use of it. Otherwise, the user is unable to complete the task at hand. Some web-based applications have been created to fill the needs of common applications, such as an application that converts PDF files to simple image files viewable from a web browser. However, these solutions are application-specific and often quite limited. For instance, converting PDF files to simple image files would cut out useful features that are available in the native application, such as the ability to search the PDF.

If users were allowed to install software on the computers, this would help alleviate this problem, but users would then potentially be forced to debug web browsing problems on each computer they used to attempt to identify missing applications. In practice, this would severely limit the number of machines that users could use as most system administrators would consider it to be a security hole to let regular users install untrusted applications onto their systems. This effectively limits this approach to being used by more computer-savvy users moving among machines already under their control.

To address these problems, we introduce WebPod, a portable system that enables mobile users to obtain the same persistent, personalized web browsing experience from any Internet-enabled device. WebPod leverages the rise of commodity storage devices that can easily fit in a user's pocket yet store large amounts of data. Such pocketable storage devices range from flash memory sticks that can hold 1 GB of data, to Apple iPods that can hold 60 GB of data. WebPod decouples a user's web browsing session from the underlying computer so that it can be suspended to a portable storage device, carried around easily, and simply resumed from the storage device on a completely different computer. WebPod provides this functionality without modifying, recompiling or relinking any applications or the operating system kernel, and with only a negligible performance impact.

WebPod operates by encapsulating a user's web browsing session in a virtualized execution environment and storing all state associated with the session on the portable storage device. WebPod virtualization decouples web browsing sessions from the underlying operating system environment by introducing a private virtual namespace that provides consistent, host-independent naming of system resources. WebPod also virtualizes the display so that a web browsing session can be scaled to different display resolutions that may be available as a user moves from one computer to another. This enables a web browsing session to run in the same way on any host despite differences that may exist among different host operating system environments and display hardware. Furthermore, WebPod virtualization protects the underlying host from untrusted applications that a user may run as part of a web browsing session. WebPod virtualization also prevents other applications from outside of the web browsing session that may be running on the host from accessing any of the session's data, protecting the browsing privacy of the user.

WebPod virtualization is combined with a checkpoint/restart mechanism. This enables a user to suspend the entire web browsing session to the portable storage device so that it can be migrated between physical computers by simply moving the storage device to a new computer and resuming the session there. WebPod ensures that file system state as well as process execution state associated with the web browsing session are preserved on the portable storage device.

The result is that WebPod enables users to maintain a common web browsing environment, no matter what computer they are using. Users can easily carry their web browser sessions with them, without lugging around a bulky laptop or being restricted to a more portable device without sufficient display size. Since WebPod does not rely on any of the application resources of the underlying host machine, web browser helper applications and plug-ins that users expect to be available will always be available using WebPod. A WebPod user's cookies, bookmarks, and other browser state are also always available as they are stored within the web browsing session. Since WebPod provides a fast checkpoint/restart mechanism, user's can quickly save their entire web browsing environment when they have to change locations, without the need to manually attempt to save all the individual elements of their state. Mobile users can simply unplug the device from the computer, move onto a new computer and plug in, and restart their session from the device to pick up where they left off.

We have implemented a WebPod prototype for use with commodity PCs running Linux and measured its performance. Our experimental results with real web applications demonstrate that WebPod has very low virtualization overhead and can migrate web browsing sessions with sub-second checkpoint and restart times. We show that WebPod can reconstitute a user's web browsing session an order of magnitude faster than if a user had to reopen the same set of web browser windows without WebPod. Our results also show that a complete WebPod browsing session including file system state requires less than 256 MB of storage. WebPod's modest storage requirements enable it to be used with the smallest form factor USB drives available on the market today, which are smaller than a person's thumb and can be conveniently carried on a key chain or in a user's pocket.

This paper focuses on the design and implementation of the WebPod virtualization and checkpoint/restart mechanisms. Section 2 presents the overall WebPod architecture and usage model. Section 3 motivates the need for WebPod virtualization and describes how this is done. Section 4 describes the WebPod checkpoint/restart mechanisms that enable WebPod to be used across operating system environments with different kernel versions. Section 5 presents experimental results measuring the performance of the WebPod system. Section 6 discusses related work. Finally we present some concluding remarks and directions for future work.

2. WEBPOD USAGE MODEL

WebPod is architected as a simple end user device that users can carry in their pockets. A WebPod session can be easily populated with the complete set of applications used in a user's normal web browsing environment so that environment is available on any computer. To the user, a WebPod session appears no different than private computer even though it runs on a host that may be running other applications. Those applications running outside of the WebPod session are not visible to a user within a WebPod session. To provide strong security, the WebPod stores the session on an encrypted file system. Therefore, even if the WebPod device is lost or stolen, an attacker will just be able to use it as his own personal storage device.

A user starts WebPod by simply plugging in a WebPod portable storage device into the computer. The computer detects the device and automatically tries to restart the WebPod session. This involves first authenticating the user by asking for a password. Authentication can also be done without passwords by using biometric technology in the form of built-in fingerprint readers available on some USB drives [13]. Once a user is authorized, WebPod mounts its encrypted file system, restarts its web browsing session, and attaches a WebPod viewer to the session to make the associated set of web browser windows available and visible to the user. Applications running in a WebPod session appear to the underlying operating system just like other applications that may be running on the host machine, and they make use of the host's network interface in the same manner.

Once WebPod is started, a user can commence web browsing using the available web browsing environment. When the user wants to leave the computer, the user simply closes the WebPod viewer. This causes the WebPod session to be quickly checkpointed to the WebPod 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 WebPod device and the session is restarted right where it left off. With WebPod, there is no need for a user to manually launch the web browser, reopen web browser windows, and reload web content. WebPod's checkpoint/restart functionality maintains a user's web browsing session persistently as a user moves from one computer to another.

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

WebPod protects web browsing 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 WebPod session, protecting the browsing privacy of a WebPod user. WebPod does rely on the host hardware and operating system kernel as is common practice for web users today. As a result, it does not protect users from attacks that may arise from tampered hardware or a compromised operating system kernel.

WebPod provides a consistent usage environment compatible with the web services model. Because WebPod is small and travels with the user, there is a risk of loss of the device and its associated data. However, this risk is limited to loss of browser state such as bookmarks and cookies since all important user data does not reside on the local device but is always stored by web services on centrally managed servers. WebPod 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 browser data loss further, WebPod is backed up periodically when it returns to the user's own computer, in the same manner as a user synchronizes a PDA. Alternatively, WebPod can be incrementally backed up automatically using a network backup service whenever it is plugged in to a networked computer. Backup would only done when the web browsing session is not actively being used to avoid impacting the user's browsing experience. In either case, if a WebPod device is lost, the user's web browsing session can be easily restored from backup onto another device.

3. WEBPOD VIRTUALIZATION

To provide a private and mobile execution environment for web browsing sessions, WebPod virtualizes the underlying host operating system and display. WebPod virtualization is necessary to enable WebPod browsing sessions to be decoupled from the underlying host on which it is being executed. This is essential to allow WebPod 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 web applications and commodity operating systems, WebPod virtualization is designed to be completely transparent to work with existing unmodified applications and operating system kernels.

3.1 Operating System Virtualization

To understand the need for operating system virtualization, we briefly discuss how applications execute in the context of commodity operating systems. When an application runs, the operating system associates a process or set or processes with it. Operating system resource identifiers, such as process IDs (PIDs), must remain constant throughout the life of a process to ensure its correct operation. Web applications commonly manipulate these operating system resource identifiers as they execute. However, these identifiers are only local unique to a particular operating system instance. When an application is moved from one computer to another, there is no guarantee that the destination operating system can provide the same identifiers to the migrated application's processes. Those identifiers may already be in use by other processes running on the destination system, preventing the migrated process from executing correctly.

WebPod virtualizes the underlying host operating system by encapsulating web browsing sessions within a host independent, virtualized view of the operating system. This virtualization approach builds upon our previous work on MobiDesk [3] and the previous work of one of the authors on Zap [9].

WebPod virtualization provides each web browsing session with its own virtual private namespace. For example, a WebPod 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 Web-

Pod application instances to access the underlying operating system. WebPod introduces this namespace to decouple processes associated with applications running in WebPod sessions from the underlying host operating system.

The WebPod 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 IPC mechanisms. On the other hand, no IPC interaction is possible across the session's boundary, because outside processes on the WebPod host are not part of the private namespace. Processes inside a session and those outside of it are only able to communicate over RPC 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. The session's namespace uses virtual identifiers to provide a host-independent view of the system, which remains consistent throughout a process's and session's lifetime. Since the session's namespace is separate from the underlying host namespace, it can preserve naming consistency for its processes, even in the presence of changes to the underlying operating system.

The WebPod namespace enables WebPod 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 WebPod processes to enable WebPod sessions to migrate from one machine to another. Names within a session are trivially 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 WebPod session, processes within the session can be migrated as a group, while avoiding resource naming conflicts among other processes running on the host.

As an additional benefit, the private virtual namespace enables the WebPod session to be securely isolated from the host by providing complete mediation to all operating system resources. Since the only resources within the WebPod session are the ones that are accessible to the owner of the session, a compromised session is limited in its ability to affect any activities running on the host 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, WebPod is able to filter out those interactions as well.

WebPod 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.

WebPod 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.

WebPod virtualization employs the `chroot` utility and file systems stacking to provide each session with its own file system namespace. The WebPod 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 WebPod 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 multiple ways to break out of a chrooted environment, especially when the `chroot` system call is allowed to be used in a session. The primary way WebPod provides security is by disallowing the privileged root user from being used within the session. The WebPod 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 WebPod processes to access any files outside of the WebPod storage device's file system, as well as the inability for the processes to run with privilege, the processes are confined to the WebPod session and can't affect change on the WebPod host.

3.2 Display Virtualization

To understand the need for display virtualization, we briefly discuss how applications typically interact with the display subsystem of a machine. Modern graphical applications such as web browsers 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. The display state associated with an application is distributed at different times between the window system and the hardware framebuffer. When an application is moved from one computer to another, it is important that all of its display state be captured so that the application can be properly redisplayed on the destination system. However, a window system may contain display state for many applications and 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.

WebPod virtualizes the display associated with a web browsing session so that it can be viewed on different hosts that may have different display systems available. This display virtualization approach builds upon our previous work on MobiDesk [3] and the previous work of one of the authors on THINC [2].

WebPod virtualization provides each web browsing session with its own virtual display server and virtual device driver to decouple the display of the web browsing session from the display subsystem of the host. The virtual display server provides a WebPod session with its own window system separate from the window system on the host, thereby separating WebPod application display state from other applications running on the host outside of the WebPod session. The display server is considered a part of the WebPod session and is checkpointed when the WebPod session is suspended and restarted when the WebPod session is resumed. Our WebPod prototype implementation uses an XFree86 4.3 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 WebPod 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 WebPod session, this simplifies display state management during checkpointing and restarting a WebPod session. As a result, checkpointing the WebPod's display state can be done by simply saving the associated memory instead of extracting display state from the host-specific framebuffer.

WebPod's virtual display device is a video hardware layer approach allows it to take full advantage of existing infrastructure and hardware interfaces, while maximizing host resources and requiring minimal computation on the host. 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 web applications indicate that the vast majority of application display commands issued can be mapped directly to standard video hardware primitives. In addition, WebPod provides direct video support by leveraging alternative YUV video formats natively supported by almost all off-the-shelf video cards available today. Video data can simply be transferred from the WebPod'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 WebPod 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 WebPod 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 WebPod display system. All it does it read the persistent display state managed by the WebPod display system. The viewer can be disconnected and reconnected to the WebPod session at any time without loss of display information since it does not maintain any persistent display state.

The WebPod display system is designed so that a session can be viewed from multiple locations at the same time. While it is being viewed on the local host in which the WebPod device is plugged in, it can be shared with another user running a WebPod viewer on a remote host. This facilitates collaboration among users. To allow WebPod to be viewed locally or across a network, WebPod implements a simple, low-level, minimum-overhead display protocol. The protocol mimics the operations most commonly found in display hardware, allowing the host to do little more than take protocol commands sent from a WebPod session to a viewer and forward them to their local video hardware to be displayed, thus reducing the latency of display processing. To support host devices that can support varying resolutions, this protocol allows the viewer to be resolution independent and scale the display appropriately.

4. WEBPOD CHECKPOINT/RESTART

WebPod virtualization enables one to continue using a single WebPod session across many disparate computers that are separately managed. WebPod combines its virtualization with a checkpoint-restart mechanism that allows the WebPod device to be checkpointed, transported and restarted across computers with different hardware and operating system kernels. WebPod 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 25 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, WebPod'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. WebPod 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.

WebPod's intermediate representation format is chosen such that it offers the degree of portability needed for migrating 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 WebPod 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. WebPod 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.

WebPod leverages high-level native kernel services in order to transform the intermediate representation of the checkpointed image into the complete internal state required by the target kernel. Continuing with the previous example, WebPod 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 WebPod 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 exported kernel symbol interface, the WebPod system is able to perform cross-kernel migration without requiring modifications to the kernel.

To eliminate possible dependencies on low-level kernel details, WebPod'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, by taking advantage of existing semantics which direct the kernel to release a process from a wait queue upon receipt of a signal, WebPod is able to empty the wait queues by suspending all processes, and therefore avoid 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 WebPod 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, WebPod virtualization is protected from such changes in the same way legacy applications are protected. However, new system calls could be 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 WebPod session only have access to devices through the virtual device drivers provided by the WebPod, it makes it simple to checkpoint the device specific data associated with the processes. In particular, since the WebPod 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's a simple matter of saving that state on checkpoint and restoring it on restart. When the WebPod viewer on the host reconnects to the virtual display driver, it is able to display the complete display.

WebPod's checkpoint mechanism preserves all application state necessary to restart the application at a later time in exactly the same state as it was when it was checkpointed, with one notable exception. WebPod does not preserve open network connections as part of its checkpointed state. WebPod closes all open network connections before it checkpoints the browsing session. When WebPod is restarted, web applications will need to reconnect to the appropriate servers as opposed to reusing previous connections. However, the need to reestablish network connections does not result in any user-perceived differences for most web applications. Web browsers were originally designed using HTTP 1.0 without persistent connections, so that browsers frequently reconnect to web servers during normal operation. While persistent HTTP connections are known to provide better performance, most users are completely oblivious to whether persistent or nonpersistent connections are used. Previous work has demonstrated that additional infrastructure can be used to preserve open network connections during migration [9]. However, we opted for a simpler design with WebPod given the ability of web applications to easily reestablish network connections without any user-perceived changes in application behavior.

5. EXPERIMENTAL RESULTS

We implemented WebPod as three components, a simple viewer application for accessing a WebPod browsing session, an unmodified XFree86 4.3 display server with a WebPod 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 WebPod 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 WebPod and the other was used as a web server for measuring web benchmark performance. To demonstrate the ability of WebPod 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 2.4.18 kernel and the other running Debian Unstable with a Linux 2.4.21 kernel.

We used a 40 GB Apple iPod as the WebPod portable storage device, though a much smaller USB memory drive could have been used. Each PC machine provided a FireWire connection which could be used to connect to the iPod. We built an unoptimized WebPod file system by bootstrapping a Debian GNU/Linux installation onto the iPod and installing a simple KDE 2.2.2 system and the Konqueror 2.2.2 web browser that comes with it, as well as the Black-Box 1.4.5 window manager. We also remove the extra packages needed to boot a full Linux system as WebPod is just a lightweight web browsing environment, not a full operating system. This resulted in a 163 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 that can store 256 MB, 512 MB and even 1 GB. Our unoptimized WebPod 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 WebPod virtualization, we used a range of benchmarks that represent various operations that occur in a web browsing environment and measured their performance on both our Linux WebPod prototype and a vanilla Linux system. We used a set of microbenchmarks that represent operations executed by web browsers as well as a real web browsing application benchmark. Table 1 shows the six benchmarks we used along with their performance on a vanilla Linux system in which all benchmarks were run from a local disk. These benchmarks were then run for comparison purposes in the WebPod portable storage environment. To obtain accurate, repeatable results, we rebooted the system between measurements. Additionally, the system call micro-benchmarks directly used the TSC register available on Pentium CPUs to record timestamps at the significant measurement events. Each timestamp's average cost was 58 ns. The files for the benchmarks were stored on the WebPod's iPod based file system. All of these benchmarks were performed in a WebPod environment running on the PC machine running Debian Unstable with a Linux 2.4.18 kernel. Figure 1 shows the results of running our benchmarks under both configurations, with the vanilla Linux configuration normalized to one, time to run the benchmark, a small number is better for all benchmarks results.

Name	Description	Linux
getpid	average `getpid` runtime	350 ns
ioctl	average runtime for the FIONREAD ioctl	427 ns
semget-semctl	IPC Semaphore variable is created and removed	1370 ns
fork-exit	process forks and waits for child which calls exit immediately	44.7 us
fork-sh	process forks and waits for child to run `/bin/sh` to run a program that prints "hello world" then exits	3.89 ms
iBench	Measures the average time it takes to load a set of web pages	826 ms

Table 1: Benchmark Description

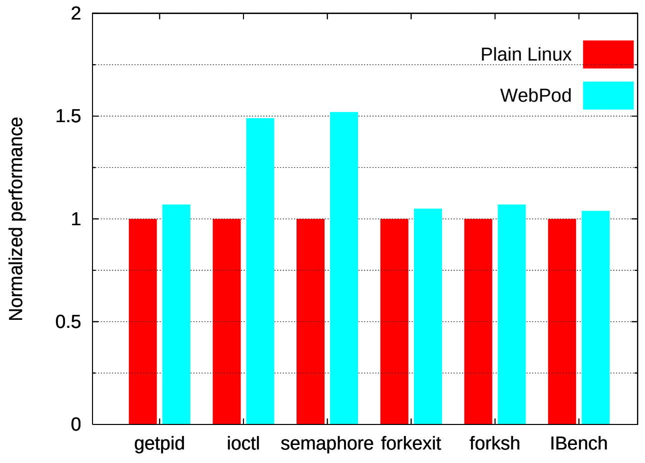

Figure 1: WebPod Virtualization Overhead

Figure 1 shows that WebPod virtualization overhead is small. WebPod incur less than 10% overhead for most of the micro-benchmarks and less than 4% overhead for the iBench application workload. The overhead for the simple system call `getpid` benchmark is only 7% compared to vanilla Linux, reflecting the fact that WebPod virtualization for these kinds of system calls only requires an extra procedure call and a hash table lookup. The most expensive benchmarks for WebPod is `semget+semctl` which took 51% longer than vanilla Linux. The cost reflects the fact that our untuned WebPod prototype needs to allocate memory and do a number of namespace translations. Kernel semaphores are widely used by web browsers such as Mozilla and Konqueror to perform synchronization. The `ioctl` benchmark also has high overhead, because of the 12 separate assignments it does to protect the call against malicious processes. This is large compared to the simple FIONREAD `ioctl` that just performs a simple dereference. However, since the `ioctl` is simple, we see that it only adds 200 ns of overhead over any

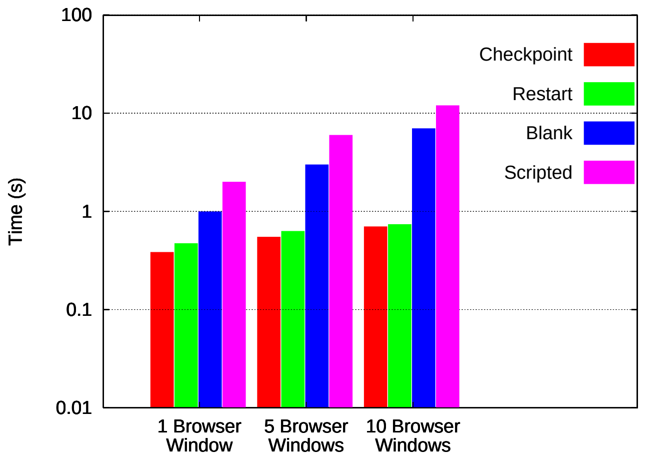

Figure 2: WebPod Checkpoint/Restart vs. Scripted Startup Latency

`ioctl`. As can be seen, there's minimal overhead for functions such as `fork` and the `fork/exec` combination. This is indicative of what happens when the web browser loads a plugin, such as Adobe Acrobat, where the web browser runs the acrobat process in the background.

Figure 1 shows that WebPod has low virtualization overhead for real applications as well as micro-benchmarks. This is illustrated by the performance on the iBench benchmark, which is a modified version of the Web Text Page Load test from the Ziff-Davis i-Bench 1.5 benchmark suite. It consists of a JavaScript controlled load of a set of web pages from the web benchmark server. iBench also uses the JavaScript to measure how long it takes to download and process each web page, then determine the average download time per page. The pages contain both text and bitmap graphics, with pages varying in the proportions of text and graphics. The graphics are embedded images in GIF and JPEG formats. Our results show that running the iBench benchmark in the WebPod environment vs running in vanilla Linux from local SCSI storage incurs no performance overhead.

To measure the cost of checkpointing and restarting WebPod sessions as well as demonstrating WebPod's ability to improve the way a user works with a web browser, we migrated multiple WebPod sessions containing different numbers of open browser windows between the two separate machines described above. Figure 2 shows how long it takes to checkpoint and restart WebPod sessions containing varying numbers of open browser windows. We compare this against how long it would take to automatically open the same browsers windows via a shell script. We compared the performance when each browser window was opened with a blank page, then compared the performance when each browser window was opened was the last visited page for the given window.

Figure 2 shows that it is significantly faster to checkpoint and restart a WebPod web browsing session than it is to have to start the same kind of web browsing session from scratch. Checkpointing and restarting a WebPod even with ten browser windows opened each take well under a second. This enables a WebPod user to very quickly disconnect from a machine after a web browsing session has been completed and plug-in to another machine and immediately start web browsing again. Some usability studies have shown that web pages should take less than one second to download for the user to experience an uninterrupted browsing process [8]. These results show that WebPod performance is fast enough that the latencies incurred in disconnecting and plugging-in are even less than the one second threshold for users to experience an uninterrupted browsing process. Furthermore, the fact that these experiments were run across two different machines with two different operating system environments and kernels demonstrates the ability of WebPod to work across different software environments.

In contrast, Figure 2 shows that starting a web browsing session the traditional way of starting the web browser application and opening a number of windows is much slower than the one second threshold for an uninterrupted web browsing process. Starting up a browsing session takes seven seconds when opening ten browser windows each with a blank page and takes twelve seconds when opening the same set of browser windows with actual web content. Even starting a web browsing session by opening a single browser window takes more than a second for both cases. Note that these experiments provide a conservative comparison as they were conducted with a local web server connected via a 100 Mbps LAN connection. In the more common case when the web server is located further away from the host over a WAN connection, the latency for starting a web browsing session without WebPod will be even worse.

Figure 2 shows that checkpointing and restarting a WebPod browsing session with web browser windows opened with actual web content is faster than just opening the same set of browser windows with blank pages without WebPod. The performance difference is even greater when comparing against the case without WebPod when actual web content is downloaded into each browser window. This is not an apples to apples comparison, as the script loads the latest version of the page from the web server. This provides a different restart model from WebPod, which restarts the web browser windows with the web content that was saved when the WebPod session was checkpointed. The WebPod approach allows one to easily access the web data that was available when the checkpoint occurred, preserving information that may no longer be available using a normal startup without WebPod, which only provides access to what is currently available from the respective web server.

Table 2 shows the amount of storage needed to store the checkpointed web browsing sessions using WebPod for each of three browsing sessions with different numbers of web browser windows opened. The results reported show checkpointed image sizes 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.

	1 Browser	**5 Browsers**	**10 Browsers**
Checkpoint	25 MB	35 MB	46 MB
File System	163 MB	163 MB	163 MB
Total	188 MB	198 MB	209 MB

Table 2: WebPod Storage Requirements

The checkpointed image size in all cases was less than 50 MB. Our results show that total WebPod storage requirement, including both the checkpointed image size and the file system size, is much less than what can fit in a small 256 MB USB drive.

Our measurements focus on the performance aspects of WebPod to demonstrate that it provides good web browsing performance. Another important aspect of demonstrating the benefits of the WebPod browsing experience would be to conduct WebPod usability studies. While our own informal experience with WebPod has been positive, we have not yet conducted realistic usability studies involving web users from a more diverse user population that would include less experienced computer users. This remains an important area of future work.

6. RELATED WORK

The emergence of cheap, portable storage devices has led to the development of web browsers for USB drives, including Stealth Surfer [12] and Portable Firefox [10]. These approaches only provide the ability to run a web browser on a USB drive. Unlike WebPod, they do not provide an entire web browsing environment. The various programs and plug-ins that a user depends to make there web experience more comfortable, do not work within this environment.

M-Systems and SanDisk have recently proposed the U3 [15] platform for providing a standard way to allow USB drives to store data and launch applications. Only limited information is currently available about U3. No U3 products currently exist and no U3 prototypes have been announced to date. However, the platform does have the potential to provide a more general framework than web browsers that can run on a USB drive. Unlike WebPod, U3 focuses on launching applications and storing user data, but does not address the needs of mobile users in providing persistent application sessions that can be checkpointed and restarted.

SoulPad [11] provides a solution similar to WebPod but based on using Knoppix Linux and VMware [16] on a USB drive. Knoppix Linux provides a Linux operating system that can boot from a USB drive for certain hardware platforms. VMware provides a virtual machine monitor (VMM) that enables an entire operating system environment and its applications to be suspended and resumed from disk. SoulPad is designed to take over the host computer it is plugged into by booting its own operating system. SoulPad then launches a VMware VM that runs the migratable operating system environment. Unlike WebPod, SoulPad does not rely on any software installed on the host. However, it requires minutes to start up given the need to boot and configure an entire operating system for the specific host being used. WebPod is designed specifically for mobile web users, which enables it to be much more lightweight. WebPod requires less storage so that it can operate on smaller USB drives and does not require rebooting the host into another operating system so that it starts up much faster.

Providing virtualization, checkpoint, and restart capabilities using a VMM such as VMware represents an interesting alternative to the WebPod operating system virtualization approach. VMMs virtualize the underlying machine hardware while WebPod virtualizes the operating system. VMMs can checkpoint and restart an entire operating system environment. However, unlike WebPod, VMMs cannot checkpoint and restart web applications without also checkpointing and restarting the operating system. WebPod virtualization operates at a finer granularity than virtual machine approaches by virtualizing individual sessions instead of complete operating system environments. Using VMMs can be more space and time intensive due to the need to include the operating system on the portable storage device.

A number of other approaches have explored the idea of virtualizing the operating system environment to provide application isolation. FreeBSD's Jail mode [6] provides a chroot like environment that processes can not break out of. More recently, Linux Vserver [7] and Solaris Zones [14] offer a similar virtual machine abstraction to the WebPod session. Unlike WebPod, all of these approaches require substantial in-kernel modifications to support the abstraction, and none of them provide the checkpoint/restart functionality available using WebPod.

Thin-client systems such as MobiDesk [3] provide an alternative usage model that also provides many of the benefits of WebPod. Such systems push all application logic to centrally managed servers and provide a simple viewer application that runs on the client to transmit user input to the server and display updates back to the client. Since all application logic runs on the server, mobile web users can access a consistent web browsing environment from any Internet-enabled device by simply connecting to the thin-client server. Unlike WebPod which is self-contained and runs locally on the host where the user is currently located, thin clients require access to additional server infrastructure.

7. CONCLUSIONS AND FUTURE WORK

We have introduced WebPod, a portable system that enhances the web browsing experience of mobile users by providing them with the same persistent, personalized web browsing session wherever they are located and on whatever computer they are using. WebPod allows an entire web 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. WebPod is more than just a web browser as it stores everything necessary for web browsing, including all helper applications and plug-ins.

WebPod provides its functionality by virtualizing operating system and display resources, decoupling a web browsing session from the host on which it is currently running. WebPod virtualization works together with a checkpoint/restart mechanism to enable WebPod users to suspend their web browsing sessions, move around, and resume their respective sessions at a later time on any computer right where they left off. WebPod's ability to migrate web browser sessions between differently configured and administered computers provides improved end user mobility.

We have implemented and evaluated the performance of a WebPod prototype in Linux. Our implementation demonstrates that WebPod supports web applications without any changes to the applications or the underlying host operating systems kernels. Our experimental results with real web applications shows that WebPod has low virtualization overhead and can migrate web sessions with sub-second checkpoint/restart times, providing superior mobility support than other proposed solutions. WebPod is unique in it's ability to provide a complete, persistent, and consistent web browser environment that is not limited to a single machine.

WebPod raises a number of interesting research areas.

WebPod focuses on improving web usage for mobile users, but the same principles could be applied to other application domains as well. WebPod leverages available portable storage technologies, but it would be worthwhile to consider what additional benefits might be possible by adding some processing capabilities on a portable device, such as enhancing user privacy and security on untrusted computers. Finally, as portable storage devices increase in capacity and ubiquity, the decoupling of storage from the computer may open new directions in how computers should be designed and how they will be used in the future.

8. ACKNOWLEDGMENTS

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

9. REFERENCES

[1] AOL Instant Messenger - AIM Express. `http://www.aim.com/get_aim/express/`.

[2] 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.

[3] 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)*, Philadelphia, PA, Sept. 2004.

[4] GMail. `https://gmail.google.com/`.

[5] Hotmail. `http://www.hotmail.com`.

[6] P.-H. Kamp and R. N. M. Watson. Jails: Confining the omnipotent root. In *2nd International SANE Conference*, MECC, Maastricht, The Netherlands, May 2000.

[7] Linux VServer Project. `http://www.linux-vserver.org/`.

[8] J. Nielsen. *Designing Web Usability: The Practice of Simplicity*. New Riders Publishing, Indianapolis, Indiana, 2000.

[9] 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)*, Boston, MA, Dec. 2002.

[10] Portable Firefox. `http://johnhaller.com/jh/mozilla/portable_firefox/`.

[11] M. Raghunath, C. Narayanaswami, C. Caster, and R. Caceres. Reincarnating pcs with portable soulpads. Technical Report RC23418 (W0411-057), IBM Research Division Thomas J. Watson Research Center, Nov. 2004.

[12] Stealth Surfer. `http://www.stealthsurfer.biz/`.

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

[14] A. Tucker and D. Comay. Solaris zones: Operating system support for server consolidaiton, May 2004.

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

[16] VMware, Inc. `http://www.vmware.com`.