WebPod: Persistent Web Browsing Sessions with Pocketable Storage Devices

Shaya Potter Jason Nieh
Computer Science Department
Columbia University, New York, NY USA
{spotter, nieh}@cs.columbia.edu
Columbia University Technical Report CUCS-047-04, November 2004

Abstract

We present WebPod, a portable device for managing web browsing sessions. WebPod leverages capacity improvements in portable solid state memory devices to provide a consistent environment to access the web. WebPod provides a thin virtualization layer that decouples a user's web session from any particular end-user device, allowing users freedom to move their work environments around. We have implemented a prototype in Linux that works with existing unmodified applications and operating system kernels. Our experimental results demonstrate that WebPod has very low virtualization overhead and can provide a full featured web browsing experience, including support for all helper applications and plug-ins one expects. WebPod is able to efficiently migrate a user's web session. This enables improved user mobility while maintaining a consistent work environment.

1 Introduction

In today's world of commodity computers and broadband, 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, are assimilating usage patterns, such as web browsing, e-mail and instant messaging, that were once limited to regular full scale computers.

The main problem that mobile users encounter is that they lack a common environment as they move around. The computer at the office is configured differently than the computer at home, which is different than 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 moves around.

Since the web browser is ubiquitous across all modern computers, it's been proposed to make the web browser the focal point of one's computer usage. Many traditional applications, such as e-mail [4, 5] and instant messaging [2], have been ported to a web services environment that is usable from within a simple web browser environment. The advantage this provides is that by using these web services, one is able to store their data on centrally managed servers and get access to the data where ever they go.

Even with keeping a common environment by using a web browser, one is still limited because a web browser contains lots of data, such as bookmarks and cookies and even browser history, that enable it to function in a more useful manner. The problem that occurs when one moves between computers, this data which is specific to a web browser installation can not 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 machine. From cookies, to page view history and the web browser's web page cache, a record of a user's usage can be recorded by a web browser not under one'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 also suffer while using web browsers due to their mobile nature. For instance, mobile users are known to pick up and move on short notice. While many web based applications one might use, such as instant messaging, are stateless, others, such as e-mail, contain state, such as messages one is in the process of composing. While many e-mail applications support the ability to save draft e-mail messages, this doesn't occur automatically. Similarly, while a user can attempt to bookmark all the pages he is looking at, he can't restart his state as it was when he resumes it on a new computer.

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 mostly out of luck. While one can try and create web based applications that fill the needs of these common applications, such as an application that converts PDF files to simple image files viewable from a web browser, this is largely a kludge. For instance, this

solution would cut out features, such as the ability to search the PDF, that are available in the native application.

Finally, even if one could solve these issues, for instance by allowing a user to install software on the computers they are using, one would still be running foreign applications on every machine he goes to. This severely limits the amount of machines one can move the between, 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 a user to moving between machines already under his control, thereby negating the majority of the need to carry around a personal web browser.

Today's computer environment is also seeing the rise of commodity storage devices that can be stowed in one's pocket, yet can also store large amounts of Data. Whether it be an Apple iPod which can hold anywhere from 20-60GB or a flash memory stick that can hold up to 1GB, these devices are becoming prevalent. We leverage these advances to solve the problems presented by introducing the WebPod, a migrating virtual machine for the mobile web user. WebPod decouples a user's computing session from the underlying computer hardware by running the processes within a specialized virtual machine. This virtual machine environment, enabled by the underlying host, allows users to run untrusted applications within the virtualized environment without the ability to harm any other application running on the same host. This virtual machine environment is combined with a checkpoint/restart mechanism that enables one to checkpoint the entire virtual machine in under a second to stable storage, such as a USB drive, and migrate it between physical computers by simply moving the USB drive to a new computer and restarting it there, even if it's running a different kernel. WebPod provides these benefits without modifying, recompiling or relinking any applications or the operating system kernel, and with only a negligible performance impact.

WebPod enables users to maintain a common web browsing environment, no matter what computer they are running. By storing their entire environment within a checkpointable and restartable virtual machine, they can easily carry their web browser session with them, without the bulk necessitated by a laptop. Similarly, since they don't have to rely on any of the resources of the underlying host machine, they are ensured that the web browser helper applications and plug-ins they expect to be available will always be available. Also, by using the web browser stored within the WebPod virtual machine, they don't have to worry about leaving state on the host that a malicious user could retrieve. Similarly, the user's cookies and bookmarks will always travel with the user as they are also stored within the web browsing session. Finally, since WebPod is fully integrated with a checkpoint/restart system, user's can simply use the subsecond checkpoint facility to save their entire web browsing environment when they have to move to their USB device, without the need to manually attempt to save all the individual elements of their state. Users can then simply remove the device from the computer, move onto a new computer and restart their session from the device to pick up where they left off.

This paper presents the design and implementation of WebPod. Section 2 presents the overall WebPod architecture and usage model. Section 3 describes the WebPod operating system virtualization and checkpoint/restart mechanisms. Section 4 presents experimental results measuring the performance of the WebPod system. Section 5 discusses related work. Finally we present some concluding remarks.

2 WebPod Architecture

WebPod is architected as a simple end user device that users can carry in their pocket. This device contains a checkpointable virtual machine that a client PC's can host, allowing users to maintain a single web browsing session as they move between computers. The session contains a virtual private environment that can be populated with the complete set of applications that are made use of in their normal web browsing environment. To the user, the session appears no different than a private computer, even though the user's web browser session coexists with the host computer. When the user wants to leave the computer, the user simply closes the WebPod pod viewer. This causes the WebPod session to be quickly checkpointed to the USB device and can be kept in the users pocket. When the user reaches another computer, he simply plug in the USB device, when the computer retrieves the insertion information it automatically restarts the WebPod session in the same state it was when the user checkpointed it.

To provide a private and mobile environment for the WebPod sessions, WebPod virtualizes two key resources: underlying devices and the operating system. WebPod virtualization is designed to work with existing unmodified applications, operating system kernels, and network infrastructure and protocols. The two components work in concert to create a completely virtualized environment for web browsing sessions.

WebPod virtualizes devices by providing a set of virtual device drivers. Explicitly, WebPod provides the session's display by providing a virtual display driver. The display driver intercepts drawing commands from user's applications, and translates the commands into a display protocol between the host and the WebPod session.

WebPod operating system virtualization provides a virtual private namespace for the WebPod session. For example, the WebPod session contains its own host independent view of OS resources, such as PID/GID, IPC, memory, file system, and devices. WebPod virtualization operates at a finer granularity than virtual machine approaches such as VMware [30] by virtualizing individual computing sessions instead of complete operating system environments. As a result, WebPod sessions can be decoupled from the underlying operating system and migrated to other computers. This enables improved user mobility in a chaotic world.

3 WebPod Virtualization

WebPod encapsulates web browsing sessions within a host independent, virtualized view of the operating system. Unlike traditional operating systems, the WebPod session is a self contained unit that can be isolated from the host system, checkpointed to its USB device, moved to another machine, and transparently restarted. This is made possible because each WebPod session has its own virtual private namespace, which provides the only means for processes to access the underlying operating system. Similarly, it only has access to the underlying physical devices of the host through virtual device drivers. To guarantee correct operation of unmodified applications, WebPod session virtualization is completely transparent. This is accomplished by providing a traditional environment with unchanged application interfaces and access to operating system services and resources.

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 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.

The namespace is virtual in that all operating system resources, including processes, user information, files, and devices, are accessed through virtual identifiers. These virtual identifiers are distinct from the host-dependent, physical resource identifiers used by the operating system. The session's namespace uses the 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 namespace, it can preserve naming consistency for its processes, even if the physical namespace changes, as may be the case when sessions are migrated across computers.

Operating system resource identifiers, such as process IDs (PIDs), must remain constant throughout the life of a process to ensure its correct operation. However, when a process is moved from one operating system instance to another, there is no guarantee that the destination system will provide the same identifiers to the migrated process; those identifiers may in fact be in use by other processes in the system. The session's namespace addresses these issues by providing consistent, virtual resource names. 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. Finally, 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 would be unable to harm any other user's session.

WebPod sessions need to make use of devices located on the host, such as the physical display. However, this can break the WebPod isolation from the host, if one would allow a WebPod process to use a physical device directly. On the other hand, even if one was willing to allow the isolation to be broken and allow use of the physical device, migrating a physical host device is difficult as they contain a lot of state, much of which is tied closely with the physical device, such as the specific video card. WebPod solves these problems by virtualizing devices, and therefore provides a virtual display driver. The display driver intercepts drawing commands from user's applications, and translates the commands into a display protocol between the WebPod host and the WebPod session. This abstracts away the specific implementation of video card features into a high level view that is applicable to all video cards. Consequently, since the device state is not in the physical device, but in the virtualized WebPod session, it simplified migration.

3.1 Session Virtualization

WebPod virtualizes sessions 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.

Session 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.

Session 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 the USB 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. Therefore, session 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 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 USB 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.

WebPod display virtualization is designed as a virtual video device driver that intercepts display commands at the video hardware layer, by providing a separate virtual video device for the WebPod session. Rather than sending display commands to local display hardware, the virtual video driver packages up display commands associated with a user's computing session and sends them to a display viewer on the WebPod host. For this purpose, WebPod implements a simple, low-level, minimum-overhead protocol. The protocol mimics the operations most commonly found in display hardware, allowing the host to do little more than forward protocol commands to their local video hardware, thus reducing the latency of display processing. In order to support host devices that can support varying resolutions, this protocol allows the viewer to be resolution independent and scale the display appropriately.

WebPod's 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 desktop applications such as web browsers, 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.

3.2 Session Migration

WebPod allows one to continue using a single WebPod session across many disparate computers that are individually managed. This is accomplished through a checkpoint-restart mechanism that allows the WebPod device to be checkpointed, transported and restarted between 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. In particular, the Linux 2.4 kernel has more than 25 minor versions. Migration is limited to these instances because major version changes are allowed to break application compatibility, which may cause running processes to break. However, 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 for-

mat 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. For instance, since XFree86 and the WebPod virtual display driver run within userspace without any device dependencies, they can be checkpointed and restarted like any other program. Because their context is totally stored with 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.

4 Experimental Results

We implemented WebPod as two components, a simple viewer application for accessing a WebPod browsing session, 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 933Mhz Intel Pentium-III CPU and 512MB RAM. The machines each had a 100 Mbps NIC and were connected to one another via 100 Mbps Ethernet and

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	1142 ms

Table 1: Benchmark Description

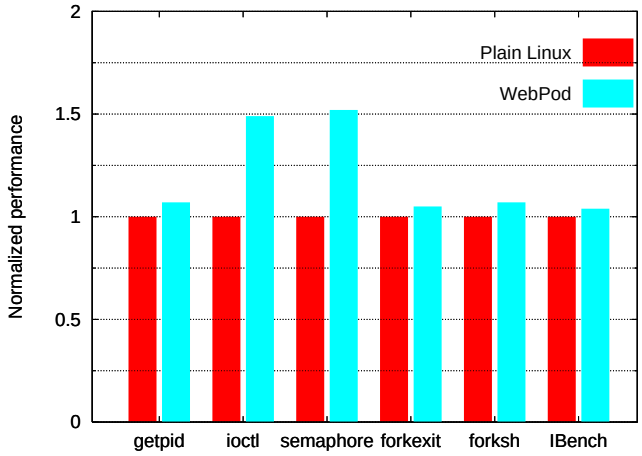

Figure 1: WebPod Virtualization Overhead

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 2.4.5 kernel and the other running Debian Unstable with a Linux 2.4.18 kernel.

We used a 40GB Apple IPod as the WebPod portable storage device. 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 the approriate web browser package while removing 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 113MB 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 256MB, 512MB and even 1GB. 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.

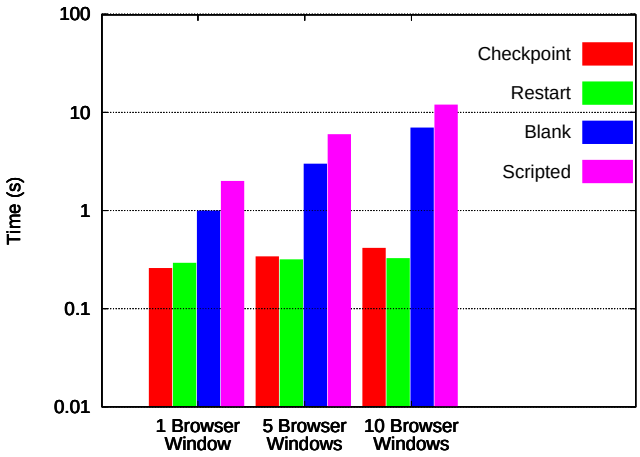

Figure 2: Checkpoint/Restart vs. Scripted Startup

1 Browser	5 Browsers	10 Browsers
12MB	18MB	26MB

Table 2: Checkpoint Size

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 wed 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.

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 `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 a small performance overhead of under four percent.

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 half 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 [20]. 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 demonstates 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.

5 Related Work

The work that is closet to WebPod are USB drives that contain a web browser, such as Stealth Surfer [9] and Portable Firefox [8]. However, these only solve the privacy issue of using a web browser not under your control. The various programs and plug-ins that a user depends to make there web experience more comfortable, do not work within this environment. Similarly, many users resort to carrying a laptop to provide the common environment they depend on. However, laptops are bulky and are not necessarily easy to carry where ever one goes, and hence one has to worry that it will be stolen if left alone for even a short period of time.

Thin clients provide some of the benefits of WebPod, such that one can have a common environment where ever they are. However, most thin client environments suffer from low bandwidth or high latency when tried to use within the chaotic Internet environment.

There are many web applications that are designed to be used from a web browser. Many popular genres include shopping [1, 3] and e-mail [4, 5]. Others include collaborative environments, such as instant messaging [2] and workgroup software [7].

A number of other approaches have explored the idea of virtualizing the operating system environment to provide application isolation. FreeBSD's Jail mode [17] provides a chroot like environment that processes can not break out of. However, since Jail is limited in what it can do, such as the fact it doesn't allow IPC within a jail[16] many real world application will not work. More recently, Linux Vserver [6] and Solaris Zones [29] offer a similar virtual machine abstraction to the WebPod session, but require substantial in-kernel modifications to support the abstraction. However, while all these allow one to run applications in an environment that prevents them from damaging the host, they don't allow one to move them between machines.

Virtual machine monitors (VMMs) have been used to provide secure isolation [30, 31, 12], and have also been used to to migrate an entire operating system environment [26]. Unlike WebPod, VMMs decouple processes from the underlying machine hardware, but tie them to an instance of an operating system. Therefore when one wants to migrate, one has to ensure that the virtual machine monitor environment is available everywhere. For instance, while Xen supports migrating it's virtual machines, since Xen becomes the operating system of the host machine, one needs Xen deployed on all the machines one wants to run on. On the other hand, the WebPod prototype can run on machines using unmodified Linux kernels

Many systems have been proposed to support process migration [24, 19, 27, 10, 25, 15, 11, 14, 18, 23, 22, 13], but not in the context of supporting migration across independent machines running different operating system versions. TUI [28] provides support for process migration across machines running different operating systems and hardware architectures. Unlike WebPod, TUI has to compile applications on each platform using a special compiler and does not work with unmodified legacy applications. WebPod builds on a pod abstraction introduced in Zap [21] to support transparent migration across systems running the same kernel version.

6 Conclusions

We have introduced WebPod, a device for mobile web browsing sessions. WebPod allows an entire web session to be stored a small solid state memory device by virtualizing the operating system. Operating system virtualization allows WebPod to migrate web browser sessions between differently configured and administrated computers providing improved end user mobility.

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

References

[1] Amazon.com. http://www.amazon.com.

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

[3] eBay. http://www.ebay.com.

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

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

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

[7] Outlook Web Access for Exchange Server 2003. http://www.microsoft.com/exchange/owa/.

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

[9] Stealth Surfer. http://www.stealthsurfer.biz/.

[10] Y. Artsy, Y. Chang, and R. Finkel. Interprocess communication in charlotte. *IEEE Software*, pages 22–28, Jan 1987.

[11] A. Barak and R. Wheeler. MOSIX: An Integrated Multiprocessor UNIX. In *Proceedings of the USENIX Winter 1989 Technical Conference*, pages 101–112, San Diego, CA, Feb. 1989.

[12] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauery, I. Pratt, and A. Warfield. Xen and the Art of Virtualization. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, Bolton Landing, NY, Oct. 2003.

[13] J. Casas, D. Clark, R. Konuru, S. Otto, R. Prouty, and J. Walpole. MPVM: A migration transparent version of PVM. *Computing Systems*, 8(2):171–216, 1995.

[14] D. Cheriton. The V distributed system. *Communications of the ACM*, 31(3):314–333, Mar 1988.

[15] F. Douglis and J. Ousterhout. Transparent process migration: Design alternatives and the sprite implementatio. *Software - Practice and Experience*, 21(8):757–785, Aug. 1991.

[16] FreeBSD Project. Developer's handbook. `http://www.freebsd.org/doc/en_US.ISO8859-1/books/developers-handbook/secure-chroot.html`.

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

[18] M. Litzkow, T. Tannenbaum, J. Basney, and M. Livny. Checkpoint and migration of unix processes in the condor distributed processing system. Technical Report 1346, University of Wisconsin Madison Computer Sciences, Apr. 1997.

[19] S. J. Mullender, G. v. Rossum, A. S. Tanenbaum, R. v. Renesse, and H. v. Staveren. Amoeba a distributed operating system for the 1990s. *IEEE Computer*, 23(5):44–53, May 1990.

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

[21] 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.

[22] J. S. Plank, M. Beck, G. Kingsley, and K. Li. Libckpt: Transparent checkpointing under unix. In *Proceedings of Usenix Winter 1995 Technical Conference*, pages 213–223, New Orleans, LA, Jan 1995.

[23] J. Pruyne and M. Livny. Managing checkpoints for parallel programs. In *2nd Workshop on Job Scheduling Strategies for Parallel Processing (In Conjunction with IPPS '96)*, Honolulu, Hawaii, Apr. 1996.

[24] R. Rashid and G. Robertson. Accent: A communication oriented network operating system kernel. In *Proceedings of the 8th Symposium on Operating System Principles*, pages 64–75, Dec 1984.

[25] M. Rozier, V. Abrossimov, F. Armand, I. Boule, M. Gien, M. Guillemont, F. Herrman, C. Kaiser, S. Langlois, P. Léonard, and W. Neuhauser. Overview of the Chorus distributed operating system. In *Workshop on Micro-Kernels and Other Kernel Architectures*, pages 39–70, Seattle WA (USA), 1992.

[26] C. P. Sapuntzakis, R. Chandra, B. Pfaff, J. Chow, M. S. Lam, and M. Rosenblum. Optimizing the migration of virtual computers. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, December 2002.

[27] B. K. Schmidt. *Supporting Ubiquitous Computing iht Statelss Consoles and Computation Caches*. PhD thesis, Computer Science Department, Stanford University, 2000.

[28] P. Smith and N. C. Hutchinson. Heterogeneous process migration: The Tui system. *Software – Practice and Experience*, 28(6):611–639, 1998.

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

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

[31] A. Whitaker, M. Shaw, and S. D. Gribble. Scale and Performance in the Denali Isolation Kernel. In *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI 2002)*, Boston, MA, Dec. 2002.