2009 Third International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies

GamePod: Persistent Gaming Sessions on Pocketable Storage Devices

Shaya Potter Ricardo Baratto Oren Laadan Jason Nieh
Computer Science Department
Columbia University
{spotter, ricardo, orenl, nieh}@cs.columbia.edu

Abstract

We present GamePod, a portable system that enables mobile users to use the same persistent, gaming environment on any available computer. No matter what computer is being used, GamePod provides a consistent gaming environment, maintaining all of a user's games, including active game state. This is achieved by leveraging rapid improvements in capacity, cost, and size of portable storage devices. GamePod provides a middleware layer that enables virtualization and checkpoint/restart functionality that decouples the gaming environment from a host machine. This enables gaming sessions to be suspended to portable storage, carried around, and resumed from the storage device on another computer. GamePod's middleware layer also isolates gaming sessions from the host, protecting the host by preventing malicious executable content from damaging the host. We have implemented a Linux GamePod prototype and demonstrate its ability to quickly suspend and resume gaming sessions, enabling a seamless gaming experience for mobile users as they move among computers.

1 Introduction

Commodity computers are increasingly a part of daily life for many people. Users make use of computers at home, school, work, and on the road. While computers are being used for many purposes, one of the primary uses for many people is playing computer games. The ubiquity of commodity computers have spawned a new era of ubiquitous computer game playing. Computer games have such broad appeal that many pervasive devices, such as mobile phones and PDAs, now offer a wide variety of games that were once limited to regular desktop computers or specialized gaming consoles.

A key problem encountered by mobile users is the inconvenience of using and managing multiple environments as they move around. For example, the computer at the office is configured differently from the computer at home, which is different from the computer at the library. These locations typically have different sets of software installed. Games that are installed at one location are often not available at another. As a result, today's mobile gamers are forced to adapt in one of two ways. One way is by carrying around bulky laptop computers for all their gaming and other computing needs, which can be cumbersome especially for air travelers given increased airport security measures. The other way is by carrying around a more convenient and smaller form factor device such as a mobile phone, PDA, or portable gaming console, which reduces the gaming experience to small screens and poor sound quality.

To address these problems, we introduce GamePod, a portable system that enables mobile users to obtain the same persistent, personalized gaming experience on any computer. GamePod leverages the ubiquity of commodity PCs and 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. GamePod decouples a user's gaming session from their 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. GamePod provides this functionality by introducing a thin middleware environment that operates without modifying, recompiling or relinking any gaming applications or the operating system kernel, and with only a negligible performance impact.

GamePod operates by encapsulating a user's gaming session in a virtualized execution environment and storing all state associated with the session on the portable storage device. GamePod virtualization decouples gaming sessions from the operating system environment by introducing a private virtual namespace that provides consistent, host-independent naming of system resources. GamePod also virtualizes the display and sound devices so that a gaming session can be scaled to dif-

978-0-7695-3834-1/09 $26.00 © 2009 IEEE
DOI 10.1109/UBICOMM.2009.61

ferent display resolutions and play with different sound hardware that may be available as a user moves from one computer to another. This enables a gaming session to run in the same way on any host despite differences that may exist among different host operating system environments, display hardware, and sound hardware. Furthermore, GamePod virtualization protects the underlying host from untrusted software that a user may run as part of a gaming session. GamePod virtualization also prevents other applications from outside of the gaming session that may be running on the host from accessing any of the session's data, protecting the privacy of the user. GamePod virtualization is combined with a checkpoint/restart mechanism. This enables a user to suspend the entire gaming 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. GamePod ensures that file system state as well as process execution state associated with the gaming session are preserved on the portable storage device. The result is that GamePod enables users to maintain a common gaming environment, no matter what computer they are using. Users can easily carry their gaming sessions with them without lugging around a bulky laptop or being restricted to a more portable device without sufficient display size or sound quality. Since GamePod does not rely on any of the application resources of the underlying host machine, any gaming helper applications and utilities that games expect to be available will always be available using GamePod. Also, because GamePod provides a fast checkpoint/restart mechanism, users can quickly save their entire gaming environment when they have to change locations without needing to manually 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 GamePod prototype for use with commodity PCs running Linux and measured its performance. Our experimental results with real game applications demonstrate that GamePod has very low virtualization overhead and can migrate gaming sessions with sub-second checkpoint and restart times. We show that GamePod can reconstitute a user's gaming session an order of magnitude faster than if a user had to restart the same games without GamePod. Our results also show that a complete GamePod gaming session including file system state requires less than 512 MB of storage. GamePod's modest storage requirements enable it to be used with small form factor USB drives available on the market today, which are smaller than a person's thumb and can be conveniently carried on a keychain or in a user's pocket.

2 GamePod Usage Model

GamePod is architected as a simple end user device that users can carry in their pockets. A GamePod session can be easily populated with the complete set of files needed by a game a user wants to play so that environment is available on any computer. To the user, a GamePod 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 GamePod session are not visible to a user within a GamePod session.

A user starts a GamePod game by simply plugging in a GamePod portable storage device into the computer. The computer detects the device and automatically tries to restart the GamePod session and attaches a GamePod viewer to the session to make the associated game available and visible to the user. Applications running in a GamePod 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 GamePod is started, a user can easily commence playing the game. When the user wants to leave the computer, the user simply closes the GamePod viewer. This causes the GamePod session to be quickly checkpointed to the GamePod 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 GamePod device and the session is restarted right where it left off. With GamePod, there is no need for a user to manually launch the game and load a saved game. GamePod's checkpoint/restart functionality maintains a user's gaming session persistently as a user moves from one computer to another.

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

GamePod protects gaming 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 GamePod session.

3 GamePod Virtualization

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

GamePod virtualizes the underlying host operating system by encapsulating gaming sessions within a host independent, virtualized view of the operating system. This virtualization approach builds upon our previous work on MobiDesk [1] and the previous work of one of the authors on Zap [7].

GamePod virtualization provides each gaming session with its own virtual private namespace. The namespace namespace is private in that only processes within the namespace can see the namespace and that it masks out resources that are not contained within it. It is virtual in that all kernel resources are accessed through virtual identifiers within the namespace that are distinct from the identifiers used by the kernel itself. For example, a GamePod 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 GamePod application instances to access the underlying operating system. GamePod introduces this namespace to decouple processes associated with applications running in GamePod sessions from the underlying host operating system.

GamePod 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.

GamePod 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.

3.2 Display Virtualization

GamePod virtualizes the display associated with a gaming 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 [1] and the previous work of one of the authors on THINC [2].

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

Rather than sending display commands to local display hardware, the GamePod 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 GamePod 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 GamePod display system. All it does it read the persistent display state managed by the GamePod display system. The viewer can be disconnected and reconnected to the GamePod session at any time without loss of display information since it does not maintain any persistent display state.

3.3 Sound Virtualization

To understand the need for audio virtualization, we briefly discuss how applications typically interact with the audio subsystem of a machine. Audio players, such as a video or mp3 player, initialize the sound device by configuring it to accept a specific type of audio stream. This audio stream is defined by its bitwise encoding representation, how many channels of output are contained within the stream as well as the sampling rate of the stream, which defines the quality of the stream. Once an application has configured the device, it simply writes out packets of data, *samples*, to the device that correspond to this configuration. The sound card then outputs the audio stream, doing an appropriate demultiplexing of the sound channels to the appropriate speakers as well as a digital to analog conversion if appropriate.

As opposed to the video subsystem, modern multimedia applications use sound devices in a fairly stateless manner. These applications just care about writing a continuous stream of sample data to the card and each sample is independent from those that came before it. However, when a multimedia application is moved from one computer to another, it is important that all of the configuration state of the sound device be captured, such as what type of sample data is being streamed to the card. This enables GamePod to configure the sound device on the new host computer exactly as it was configured before, and enables the application to continue sending its samples to the sound device and have them play as expected. One can simply capture this configuration state because modern operating systems provide a consistent kernel based API for its sound subsystem. Applications use this API to configure the sound devices and write samples to them. Therefore, applications are not tied to any particular physical sound device.

GamePod provides two ways types of sound support. First, GamePod has the ability to restrict the configuration settings that an application can set on the GamePod's sound device. For example, GamePod can restrict the settings that are allowed to the subset that are available on almost all sound cards in use today, such as 44 and 48khz sound, 16bit audio and stereo channels. This enables users to migrate a GamePod session between computers without being concerned about underlying hardware support on their target machines. Second, GamePod can enable full access to the capabilities of the underlying host sound card. For example, to play a DVD in full 5.1 surround sound. However, users who migrate their GamePods to machines that don't have support for such a feature, will not be able to restart their GamePod session and will have to re-launch that DVD player application.

4 GamePod Checkpoint/Restart

GamePod virtualization and checkpoint-restart mechanisms enable a session instance to continue execution across many disparate computers that are separately managed. Checkpoint-restart provides the glue that permits a GamePod device to be checkpointed, transported and restarted across distinct computers with distinct hardware and operating system kernels. Migration is limited between machines with a common CPU architecture, and that run "compatible" operating systems.

Compatibility is determined by the extent to which they differ in their API and their internal semantics. Minor versions are normally limited to maintenance and security patches, without affecting the kernel's API. Major versions carry significant changes that may break application compatibility. In particular, they may modify the application's execution semantics, or introduce new functionality, nevertheless they usually maintain backward compatibility. For instance the Linux kernel has two major versions, 2.4 and 2.6, each with over 30 minor versions respectively. Linux 2.6 significantly differs in how threads behave, and also introduces various new system calls. This implies that migration across minor versions in general is not restricted, while migration between major versions is only feasible from older to newer.

GamePod's checkpoint-restart mechanism relies on an intermediate abstract format to represent the state that needs to be saved. While the low-level details as maintained by the operating system may change radically between different kernels, the high-level properties are unlikely to change since they reflect the actual semantics upon which the application rely. GamePod describes the state of a process in terms of this higher-level semantic information rather than the kernel specific data. To illustrate this, let us consider the data that describes inter-process relationships, e.g. parent, child, siblings, threads etc. The operating system normally optimize for speed by keeping multiple data structures to reflect these relationships. However this format is of limited portability across different kernels, and in Linux the exact technique indeed changed between 2.4 and 2.6. Instead, GamePod captures a high-level representation of the relationships that mirrors its semantics. In particular, it simply keeps a tree structure to describe these relationships. The same holds for other resources, e.g. communication sockets, pipes, open files, system timers etc: GamePod extracts the relevant state the way it is encapsulated in the operating system's API, rather than the details of its implementation. Doing so maximizes portability across kernel versions by adopting properties that are considered highly stable.

To accommodate for differences in semantics that inevitably do occur occasionally between kernel versions, GamePod uses specialized conversion filters. The checkpointed state data is saved and restored as a stream. The conversion filters operate on this stream and manipulate its contents. Although typically they are designed to translate between different representations, they can be used to perform other operations such as compression, encryption etc. Their main advantages are their extreme flexibility, and the fact that they are executed like regular helper applications. Building on the example above, since the thread model changes between Linux 2.4 and 2.6, a filter can easily be designed to upgrade the former abstract data to adhere to the new semantics. Additional filters can be built should semantics changes occur in the future. The outcome is a very robust and powerful solution.

GamePod 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 during restart. Continuing with the previous example, GamePod restores the structure of the process tree by exploiting the native `fork` system call. In accordance to the abstract process tree data, a determined sequence of `fork` calls is issued to replicate the original relationships. The main benefit is voiding the need to deal with any internal kernel details. Furthermore, high level primitives of this sort remain virtually unchanged across kernel changes (minor or major). Finally, these services are available for use by loadable kernel modules, enabling GamePod to perform cross-kernel migration without requiring modifications to the kernel.

Finally, we must ensure that changes in the system call interfaces are properly handled. GamePod has a virtualization layer that employs system call interposition to maintain namespace consistency. It follows that a change in the semantics for any system call that is intercepted could raise an issue in migrating across such differences. Fortunately such changes are rare, and when they occur, they are hidden by standard libraries from the application level lest they break the applications. Consequently, GamePod is protected the same way legacy applications are protected. On the other hand, the addition of new system calls to the kernel requires that the encapsulation be extended to support them. Moreover, it restricts the possibility of migration back to older versions. For instance, an application that invokes the new `waitid` system call in Linux 2.6 cannot be migrated back to 2.4, unless an emulation layer exists there.

5 Experimental Results

We implemented GamePod as three components, a simple viewer application for accessing a GamePod session, an unmodified XFree86 4.3 display server with a GamePod virtual display device driver, and a loadable kernel module that provides the GamePod middleware layer 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 GamePod environment on various applications.

Experiments were conducted on an IBM T42p Thinkpad with a 1.8GHz Pentium-M CPU with 1 GB

RAM and a 60 GB 7200 RPM hard disk. The machine had a ATI FireGL Mobility T2 video card with 128 MB of Ram and an Intel Gigabit Ethernet controller connected to a 100 Mbps network. The host laptop ran the Ubuntu 5.04 Linux distribution, while the GamePod itself was based on a plain Debian unstable distribution.

We used a 512 MB SanDisk Cruzer USB memory key as the GamePod portable storage device, though larger or smaller devices could be used as well depending on the file system requirements of the specific GamePod device. The GamePod device we created contained a full X11 system, as well as a large collection of games including Quake 2, XChess, Solitaire, Tetris, Blackjack, Mahjong and many others. This GamePod system required a file system of only 283 MB. We built the unoptimized GamePod file system by bootstrapping a Debian GNU/Linux installation onto the 512 MB USB memory key and installing a simple XFree86 4.3 environment 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 GamePod is just a lightweight gaming environment, not a full operating system. Depending on the game one wants to carry, the extra overhead can be minimal in regards to smaller games such as Chess, to multiple gigabytes for games such as Unreal Tournament 2004. Our base unoptimized GamePod 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 for each individual game.

In order to test the overhead the GamePod system imposes on real games, we benchmarked the performance of Quake 2 using two time demos. Quake 2's timedemo feature enables the Quake 2 engine to play a prerecorded game demo at the highest possible speed while recording the average frames per second (FPS) the engine was able to sustain. We tested against the default demo built into the game. We tested this demo under the GamePod environment and compared the results to a plain Linux system. For the built in demo, GamePod achieves an average of 36 FPS, while plain Linux was able to average 42 FPS, resulting in GamePod having an overhead of 17%.

To measure the cost of checkpointing and restarting GamePod sessions as well as demonstrating GamePod's ability to improve the way a user plays games, we migrated multiple game sessions containing different games between the two machines described above. Figure 1 shows how long it takes to checkpoint and restart GamePod sessions containing the different games. We compare this against how long it would take to automatically open the same games manually. We compared the performance against the time it takes to simply restart the game.

Figure 1 shows that it is significantly faster to checkpoint and restart a GamePod gaming session than it is to have to start the same kind of gaming session from scratch. Checkpointing and restarting a GamePod even with a complex application like Quake 2 takes well under a second. This enables a GamePod user to very quickly disconnect from a machine after a gaming session has been completed and plug-in to another machine and immediately start playing games again. In contrast, Figure 1 shows that starting a game the traditional way is much slower, even though it's installed on a 7200 RPM hard disk as opposed to a slow USB memory key.

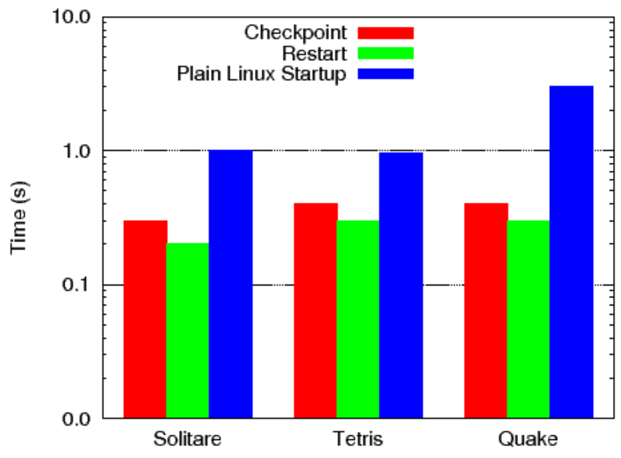

Figure 1: GamePod Checkpoint/Restart vs. Scripted Startup Latency

	Quake 2	Tetris	Solitare
Checkpoint	44 MB	22 MB	50 MB
File System	283 MB	283 MB	283 MB
Total	327 MB	305 MB	383 MB

Table 1: GamePod Storage Requirements

Table 1 shows the amount of storage needed to store the checkpointed gaming sessions using GamePod for each of three separate GamePod environments. 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. The checkpointed image size in all cases was less than 50 MB. Our results show that total GamePod storage requirement, including both the checkpointed image size and the file system size, is much less than what can fit in a small 512 MB USB drive.

6 Related Work

This works build on our previous work with WebPod [9] and DeskPod [10]. It improves on those approaches by optimizing the display system to enable the high speed rendering needed by games, as well as introducing sound virtualization and migration.

The emergence of cheap, portable storage devices has led to the development of web browsers for USB drives, including Stealth Surfer [14] and Portable Firefox [8]. These approaches only provide the ability to run a web browser on a USB drive. Unlike GamePod, they do not provide a generic environment for running a variety of applications.

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 GamePod, 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.

Many portable gaming systems have been used over the years. Famous one such as the Nintendo GameBoy [6] are still popular as their small form factor enables them to be easily carried anywhere. Others, such as the recent Sony PlayStation Portable [13] include many advanced features such as 3D graphics support. Unlike GamePod, these devices enable to you to play a game wherever you are, as they are an entire self contained unit. However, unlike GamePod which lets one take advantage of whatever hardware is available, these devices limit you to a system with a small screen, limited battery life and inferior sound.

SoulPad [12] provides a solution similar to GamePod 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 GamePod, 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. GamePod is designed specifically for playing games, which enables it to be much more lightweight. GamePod 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 GamePod operating system virtualization approach. VMMs virtualize the underlying machine hardware while GamePod virtualizes the operating system. VMMs can checkpoint and restart an entire operating system environment. However, unlike GamePod, VMMs cannot checkpoint and restart games without also checkpointing and restarting the operating system. GamePod 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 [4] provides a chroot like environment that processes can not break out of. More recently, Linux Vserver [5] and Solaris Zones [11] offer a similar virtual machine abstraction to the GamePod session. Unlike GamePod, all of these approaches require substantial in-kernel modifications to support the abstraction, and none of them provide the checkpoint/restart functionality available using GamePod.

Some Enterprise Java Bean application servers, such as BEA's WebLogic Server [3], enable programmers to create StateFull Session Beans (SFSB) that can support the failover property. This enables the application server to migrate the SFSB to another server in case of failure. However, unlike GamePod, this requires the programmer to explicitly write a SFSB eliminating the use of any legacy code and does not provide an environment that is conducive for game development.

7 Conclusions and Future Work

We have introduced GamePod, a portable system that enhances user's gaming experience by providing them with a persistent gaming session wherever they are located and on whatever computer they are using. GamePod allows an entire gaming 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.

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

GamePod opens up new possibilities for publishers of video games. By using the GamePod middleware layer to decouple the game's execution from the underlying hardware, game publishers can ensure that their games run in a consistent environment on all machines that they are executed on. While our GamePod prototype is usable by a user who builds it himself, the GamePod concept can be extended to support generic users who can buy it in a store and simply plug it into their computer and start playing immediately. Similarly, GamePod can be integrated with a Digital Rights Management interface to enable users to share and copy GamePod's bought in a store in a controlled manner.

We have implemented and evaluated the performance of a GamePod prototype in Linux. Our implementation demonstrates that GamePod supports regular games without any changes to the applications or the underlying host operating systems kernels. Our experimental results with real games shows that GamePod has low virtualization overhead and can migrate gaming sessions with very fast checkpoint/restart times. GamePod is unique in it's ability to provide a complete, persistent, and consistent gaming environment that is not limited to a single machine.

8 Acknowledgments

This work was supported in part by NSF grants CNS-0426623, CNS-0717544, and CNS-0914845.

References

[1] 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.

[2] R. A. Baratto, L. N. Kim, and J. Nieh. THINC: A Virtual Display Architecture for Thin-Client Computing. In *Proceedings of the 20^{th} ACM Symposium on Operating Systems Principles (SOSP)*, Oct. 2005.

[3] BEA Systems. `http://dev2dev.bea.com/products/wlplatform81/index.jsp`.

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

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

[6] Nintendo of America. `http://www.gameboy.com/`.

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

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

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

[10] S. Potter and J. Nieh. Highly Reliable Mobile Desktop Computing in Your Pocket. In *Proceedings of the IEEE Computer Society Signature Conference on Software Technology and Applications (COMPSAC)*, Sept. 2006.

[11] D. Price and A. Tucker. Solaris Zones: Operating System Support for Consolidating Commercial Workloads. In *18th Large Installation System Administration Conference (LISA 2004)*, Nov. 2004.

[12] 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.

[13] Sony Computer Entertainment America Inc. `http://www.us.playstation.com/psp.aspx`.

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

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

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