2009 11th IEEE International Symposium on Multimedia

MediaPod: A Personalized Multimedia Desktop In Your Pocket

Shaya Potter Ricardo Baratto Oren Laadan Leonard Kim Jason Nieh
Computer Science Department
Columbia University
{spotter, ricardo, orenl, lnk2101, nieh}@cs.columbia.edu

***Abstract*—We present MediaPod, a portable system that allows mobile users to maintain the same persistent, personalized multimedia desktop environment on any available computer. Regardless of which computer is being used, MediaPod provides a consistent multimedia desktop session, maintaining all of a user's applications, documents and configuration settings. This is achieved by leveraging rapid improvements in capacity, cost, and size of portable storage devices. MediaPod provides a virtualization and checkpoint-restart mechanism that decouples a desktop environment and its applications from the host, enabling multimedia desktop sessions to be suspended to portable storage, carried around, and resumed from the storage device on another computer. MediaPod virtualization also isolates desktop sessions from the host, protecting the privacy of the user and preventing malicious applications from damaging the host. We have implemented a Linux MediaPod prototype and demonstrate its ability to quickly suspend and resume multimedia desktop sessions, enabling a seamless computing experience for mobile users as they move among computers.**

***Keywords*-Multimedia computing, Operating system kernels, Personal computers, Computer peripherals, Virtual computers**

I. Introduction

In today's world of commodity computers, 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, video and music playing, that were once limited to desktop computers.

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 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. Beyond regular applications, such as word processors and spreadsheets, many users depend on a specific set of multimedia codecs to be installed on a computer to ensure that they can watch and listen to video and audio files. Similarly, mobile users seek consistent access to their files, which is difficult to guarantee as they move around.

To address these problems, we introduce MediaPod, a portable system that allows mobile users to obtain the same persistent, personalized multimedia desktop experience on any computer. MediaPod 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 to Apple iPods that can hold many gigabytes of data. MediaPod decouples a user's multimedia desktop session from one's 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. MediaPod provides this functionality by introducing a thin virtualization layer that operates without modifying, recompiling or relinking any desktop or multimedia applications or the operating system kernel, and with only a negligible performance overhead.

MediaPod operates by encapsulating a user's multimedia desktop session in a virtualized execution environment and storing all state associated with the session on the portable storage device. MediaPod virtualization decouples multimedia desktop sessions from the operating system environment by introducing a private virtual namespace that provides consistent, host-independent naming of system resources. MediaPod also virtualizes the display and sound devices so that the multimedia desktop's applications can be scaled to different display resolutions and play with different sound hardware that may be available as a user moves from one computer to another. This allows a multimedia desktop 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, MediaPod virtualization protects the underlying host from untrusted software that a user may run, such as an untrusted download, as part of their multimedia desktop session. MediaPod virtualization also prevents other applications from outside of the multimedia desktop session that may be running on the host from accessing any of the session's data, protecting the privacy of the user.

MediaPod virtualization is combined with a checkpoint-restart mechanism. This allows a user to suspend the entire multimedia desktop 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. MediaPod ensures that file system state, process execution and multimedia device state associated with the multimedia desktop session are preserved on the portable storage device.

The result is that MediaPod allows users to maintain a single multimedia desktop environment, no matter what computer they are using. Users can easily carry their multimedia desktop 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 MediaPod does not rely on any of the application resources of the underlying host machine, any multimedia helper applications and utilities that users expect to be available will always be available using MediaPod. Also, because MediaPod provides a fast checkpoint-restart mechanism, users can quickly save their entire multimedia desktop 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 MediaPod prototype for use with commodity PCs running Linux and measured its performance. Our experimental results with real desktop and multimedia applications demonstrate that MediaPod has very low virtualization overhead and can migrate multimedia desktop sessions with very quick checkpoint and restart times. We show that MediaPod can reconstitute a user's multimedia desktop session an order of magnitude faster than if a user had to restart the same desktop and multimedia applications without MediaPod. Our results also show that a complete MediaPod multimedia desktop session including file system state requires less than 1 GB of storage. MediaPod's modest storage requirements lets it be used with small form factor USB drives available on the market today. These gadgets, are smaller than a person's thumb and can be conveniently carried on a keychain or in a user's pocket. Users can also carry larger devices, such as an Apple iPod, letting them transport around a large amount of multimedia content.

II. MediaPod Model

MediaPod is architected as a simple end-user device that users can carry in their pocket. This device contains a checkpointable virtual environment that a client PC can host, allowing users to maintain a single desktop and multimedia application session as they move between computers as shown in Figure 1. The session contains a virtual private environment that can be populated with the complete set of applications commonly deployed in the user's day-to-day multimedia desktop environment. To the user, the session appears no different than a private computer, even though the user's session coexists with the host computer.

A user starts a MediaPod session by plugging in a MediaPod storage device into the computer. The computer detects the device and automatically tries to restart the MediaPod session. It then attaches a MediaPod viewer to the session

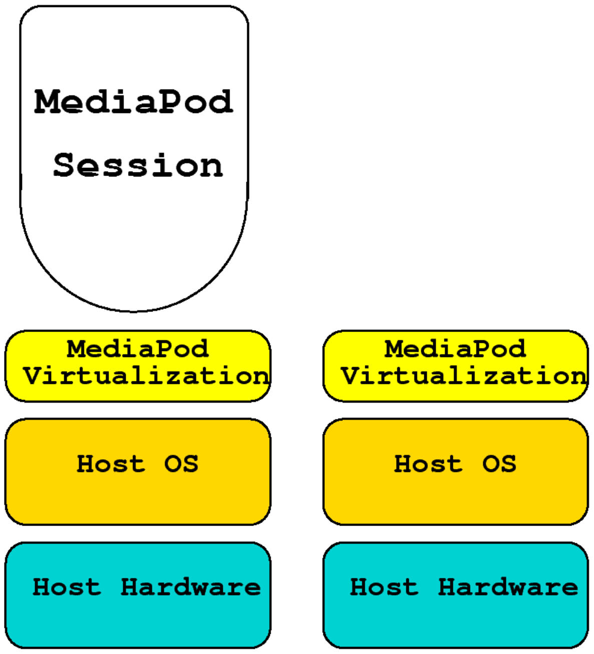

Figure 1. The MediaPod Model allows migrating a multimedia desktop session between computers running the MediaPod virtualization layer

to make the associated multimedia desktop session available and visible to the user. Applications running in a MediaPod session appear to the underlying operating system just like other applications that may be running on the host machine, and they make use of selected host resources, such as CPU, memory, network, sound, and display.

Once MediaPod is started, users can easily commence using their desktop and multimedia applications. When the user wants to leave the computer, the user simply closes the MediaPod viewer. This causes the MediaPod session to be quickly checkpointed to the MediaPod 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 MediaPod device and the session is restarted right where it left off. With MediaPod, there is no need for a user to manually launch their desktop or multimedia applications and reload their content. MediaPod's checkpoint-restart functionality maintains a user's multimedia desktop session persistently as a user moves from one computer to another.

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

To provide a private and mobile environment for the MediaPod session, MediaPod virtualizes two key resources: underlying devices and the operating system. MediaPod 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 multimedia desktop sessions.

MediaPod virtualizes devices by providing a set of virtual device drivers. Explicitly, MediaPod 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 MediaPod session. MediaPod also lets application make use of the sound device by virtualizing access to it to ensure that applications can have a common set of abilities as it moves between machines.

MediaPod operating system virtualization provides a virtual private namespace for the MediaPod session. For example, the MediaPod session contains its own host independent view of OS resources, such as PID/GID, IPC, memory, file system, and devices. MediaPod virtualization operates at a finer granularity than virtual machine approaches such as VMware [1] by virtualizing individual computing sessions instead of complete operating system environments. As a result, MediaPod sessions can be decoupled from the underlying operating system and migrated to other computers. This improves user mobility in a chaotic world.

III. MediaPod Virtualization

To provide a private and mobile execution environment for multimedia desktop sessions, MediaPod virtualizes the underlying host operating system, display and sound card. MediaPod virtualization is necessary to let MediaPod multimedia desktop computing sessions be decoupled from the underlying host on which it is being executed. This isolates the applications running within MediaPod from the underlying system and other applications on the same machine. It also allows the applications to be checkpointed on one machine and restarted on another. Given the large existing base of desktop and multimedia applications, MediaPod virtualization is designed to be completely transparent to work with existing unmodified desktop and multimedia applications as well as operating system kernels.

A. *Operating System Virtualization*

MediaPod 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 Zap [2]–[4].

MediaPod virtualization provides each multimedia desktop session with its own virtual private namespace. The 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 MediaPod session contains its own host independent view of operating system resources, such as Process IDs (PIDs), IPC, memory, file system, and devices. The namespace is the only means for the processes associated with a running application within a MediaPod to access the underlying operating system. MediaPod introduces this namespace to decouple processes associated with applications running in MediaPod sessions from the underlying host operating system enabling MediaPod sessions to be migrated from one machine to another.

MediaPod 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 is created and returned to the process. Similarly, every time a process passes a virtual name to the operating system, it is intercepted by the virtualization layer, which substitutes 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.

MediaPod 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.

B. Display Virtualization

MediaPod virtualizes the display associated with a multimedia 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 [5] and THINC [6].

MediaPod virtualizes the display associated with a multimedia desktop session so that it can be viewed on different hosts that may have different display systems available. MediaPod virtualization provides each multimedia desktop computing session with its own virtual display server and virtual device driver, thus decoupling the display of the session from the display subsystem of the host. The virtual display server provides a MediaPod session with its own window system separate from the window system on the host, thereby isolating the MediaPod's application display state from other applications running on the host outside of the session. The display server is considered to be part of the MediaPod session and is checkpointed when the session is suspended and restarted when the session is resumed.

Instead of rendering display commands to a real device driver associated with a physical device on the host, the virtual display server directs its commands to a user-level, virtual device driver, which is not associated to any particular display hardware. This approach abstracts away the specific implementation of video card features into a higher level view that is applicable to all video cards. Since the device state is not in the physical device but in the virtualized MediaPod session, this simplifies display state management during checkpoint and restart. As a result, checkpointing the MediaPod's display state can be done by simply saving the user-level virtual driver state, instead of extracting display state from the host-specific framebuffer.

Rather than sending display commands to local display hardware, the MediaPod virtual video driver packages up display commands associated with a user's computing session, writes them to memory, and lets them to viewed using a MediaPod viewer application that runs in the context of the window system on the host. The viewer is completely decoupled from the rest of the MediaPod display system, thus providing the connection between the fully-virtualized MediaPod environment and the hosting computer. The viewer's functionality consists of nothing more than reading the persistent display state managed by the MediaPod display system, and passing it on to the local computer. The viewer can be disconnected and reconnected to the MediaPod session at any time without loss of information since it does not maintain any persistent display state.

MediaPod provides support for full-screen, full-framerate video playback by natively implementing standard hardware video playback interfaces. These interfaces leverage alternative YUV video formats, natively supported by almost all off-the-shelf video cards available today. Video data from an application is simply transferred from the MediaPod's virtual display driver to the host's video hardware, through the viewer application, which automatically does inexpensive, high speed, color space conversion, and scaling.

C. 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 lets MediaPod configure the sound device on the new host computer exactly as it was configured before, and allows 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.

MediaPod provides two types of sound support. First, MediaPod has the ability to restrict the configuration settings that an application can set on the MediaPod's sound device. For example, MediaPod 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 48 KHz sound, 16 bit audio and stereo channels. This allows users to migrate a MediaPod session between computers without being concerned about underlying hardware support on their target machines. Second, MediaPod can allow 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 MediaPods to machines that don't have support for such a feature will not be able to restart their MediaPod session and will have to re-launch that DVD player application.

IV. MediaPod Checkpoint-Restart

MediaPod virtualization and checkpoint-restart mechanisms lets a session instance continue execution across many disparate computers that are separately managed. Checkpoint-restart provides the glue that permits a MediaPod 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, whereas migration between major versions is only feasible from older to newer.

MediaPod's checkpoint-restart mechanism relies on an intermediate abstract format to represent the state that needs to be saved. Although 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. MediaPod 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 optimizes 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, MediaPod 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: MediaPod 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 semantic differences that occur occasionally between kernel versions, MediaPod 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 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 robust and powerful solution.

MediaPod leverages high-level native kernel services 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, MediaPod 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 minor or major kernel changes. Finally, these services are available for use by loadable kernel modules, enabling MediaPod 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. MediaPod 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, MediaPod 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.

V. Experimental Results

We implemented MediaPod as three components, a viewer application for accessing a MediaPod session, an unmodified

XFree86 4.3 display server with a MediaPod virtual display device driver, and a loadable kernel module that provides the MediaPod virtualization 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 MediaPod environment on various applications.

We conducted experiments on two platforms. Virtualization overhead benchmarks were run on an IBM Netfinity 4500R machine with dual 933Mhz Intel Pentium-III CPU, 512MB RAM and a 9.1 GB SCSI HD running Debian GNU/Linux with a 2.6.8.1 kernel to show MediaPod's ability to run in single and multiprocessor environments. To measure the actual application scenarios' checkpoint and restart times we used an IBM T42p Thinkpad which is indicative of a consumer level machine to which a MediaPod is transported. The laptop has a 1.8GHz Pentium-M CPU with 1 GB RAM, a 60 GB 7200 RPM hard disk, and an ATI FireGL Mobility T2 video card with 128 MB of Ram and an Intel Gigabit Ethernet controller. The host laptop ran the Ubuntu 5.04 Linux distribution, while the MediaPod itself was based on Debian GNU/Linux 4.0.

We used a 40 GB Apple iPod as the MediaPod portable storage device, though a much smaller USB memory drive could have been used. Each PC machine provided a USB connection which could be used to connect to the iPod. We built an unoptimized MediaPod file system by bootstrapping a Debian GNU/Linux installation onto the iPod and installing the appropriate packages needed for a regular multimedia desktop environment. We removed the extra packages needed to boot a full Linux system as MediaPod is just a lightweight multimedia desktop computing environment, not a full operating system. This resulted in a 633 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 1 GB. Our unoptimized MediaPod 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 MediaPod virtualization, we used a range of benchmarks listed in Table I that represent various kernel operations that we virtualize and that occur in a multimedia and desktop applications. We measured the performance on both our Linux MediaPod prototype and a vanilla Linux systems. Additionally, the system call micro-benchmarks directly used the TSC register available on Pentium CPUs to record timestamps at the significant

Name	Description
getpid	runtime for returning process identifier
getppid	runtime for returning parent process identifier
getpgid	runtime for returning process group identifer
fork	runtime for creating a new process
msgget	runtime for creation of IPC message queue
msgsnd	runtime for sending messages to IPC message queue

Table I
Kernel Virtualization Benchmarks

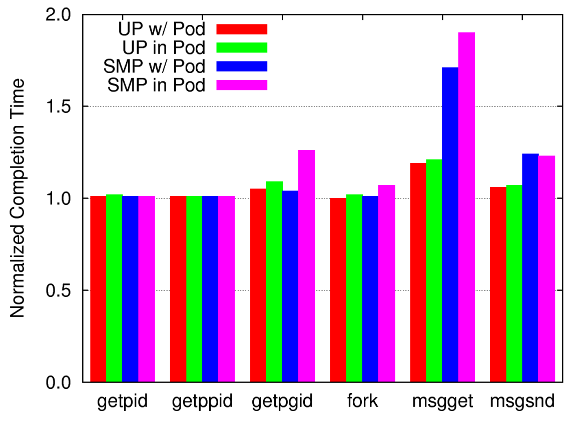

Figure 2. MediaPod Virtualization Overhead

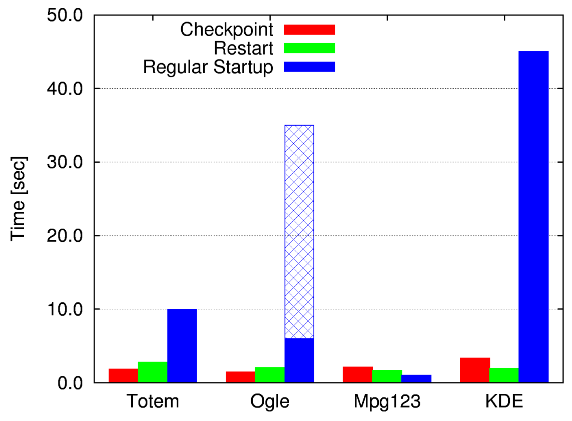

Figure 3. MediaPod Checkpoint/Restart vs. Regular Startup Latency

measurement events. Figure 2 summarizes the overhead imposed by MediaPod on regular applications running on a system with MediaPod (/with pod) as well as applications running within MediaPod itself (/in pod). Figure 2 shows that MediaPod virtualization overhead is small. MediaPod incur less than 10% overhead for most of the micro-benchmarks, this is due to the fact that for the majority of system calls MediaPod just has to perform a simple identifier translation. On the other hand, one system call, `msgget` is significantly larger due to its use of keys for interprocess communication. MediaPod has to perform multiple internal kernel calls to ensure that processes in different pods do not end up with the same underlying physical key. This would allow them to directly communicate and therefore break MediaPod's isolation model. However, this system call and the other illustrate the fact that MediaPod's kernel virtualization inserts very little overhead for regular applications because the large majority of an application's time is not spent doing system calls that MediaPod virtualizes. For example, a process only runs the `msgget` system call when its sets up its access to the queue. This is only done once per message queue for the process' life time.

To measure the cost of checkpointing and restarting MediaPod sessions as well as demonstrating MediaPod's ability to improve the way a user make use of their multimedia desktop environments, we migrated multiple MediaPod sessions containing different sets of running desktop and multimedia applications between reboots on the machine described above. We migrated two multimedia video sessions to demonstrate that MediaPod can play files encoded in different ways. We used the Totem media player to play an XviD encoded version of a DVD, while Ogle was used to play a straight DVD image copied to the MediaPod. We similarly migrated a MediaPod KDE desktop that was playing an mp3 file using the mpg123 program, as well as a regular user's KDE desktop containing the KWord word processor, KSpread spreadsheet, Konqueror web browser and a PDF file viewer all with content loaded. In all cases, MediaPod was able to checkpoint its complete state and resume it exactly as it was after migration. Figure 3 shows how long it took to checkpoint and restart MediaPod sessions containing the different multimedia and desktop applications. We compared the performance against the time it takes to simply restart the applications.

Figure 3 shows that it is significantly faster to checkpoint and restart a MediaPod multimedia desktop session than it is to have to start the same kind of desktop and multimedia application session from scratch. Checkpointing and restarting a MediaPod even with complex applications, such as an entire KDE desktop and a DVD player, is very fast compared to regular startup. This allows a MediaPod user to very quickly disconnect from a machine and plug-in to another machine and immediately start using his applications again. Although the mpg123 application appears to start faster than MediaPod, this is because it's not an apple to apple comparison. MediaPod restarts an entire KDE desktop environment, and its many megabytes of code, whereas the plain mpg123 test simply measures how long it takes to start the simple 136 KB mpg123 program.

In general, Figure 3 shows that starting applications the traditional way is much slower, even when installed on a 7200 RPM hard disk as opposed to a slow USB storage device. Part of this reason is that to start an environment from fresh involves a lot of runtime overhead as it parses what hardware and services are available. For instance, the totem media player checks to see if there is network connectivity, if the system has a CROM drive and if it can hook into the system's hardware abstraction layer to be notified of hardware events such as compact disk insertion. On the other hand, programs like DVD players take a while to start up because of filler, such as the FBI warning, trailers and the DVD menu, that are inserted at the beginning of the film and can not be skipped. In Figure 3 we show the times it takes for the Ogle DVD player to startup and reach the point where one can start watching.

Table II shows the amount of storage needed to store the checkpointed multimedia desktop sessions using MediaPod for each of three separate MediaPod 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 90 MB. Our results show that total MediaPod storage requirement, including both the checkpointed image size and the file system size, is much less than what can fit in a small 1 GB USB drive.

	Totem	**Ogle**	**mpg123**	**KDE**
Checkpoint	44 MB	27 MB	17 MB	81 MB
File System	633 MB	633 MB	633 MB	633 MB
Total	677 MB	660 MB	650 MB	714 MB

Table II
MEDIAPOD STORAGE REQUIREMENTS

VI. RELATED WORK

The emergence of cheap, portable storage devices has led to the many interesting applications. Portable multimedia systems such as the Apple iPod [7] are very popular as their small form factor allows them to be easily carried anywhere. Unlike MediaPod, these devices are self-contained and can play multimedia content directly. However, unlike MediaPod which takes advantage of whatever hardware is available, these devices limit you to a system with a small screen, limited battery life and inferior sound.

Various applications run from USB drives have been created. Portable Firefox [8] can run a Web browser from a USB drive, but does not provide a generic environment for running desktop and multimedia applications. The Collective [9], Moka5 [10], and SoulPad [11] allow suspending and resuming a virtual machine stored on a USB drive. Unlike MediaPod, SoulPad does not rely on any software installed on the host as it uses Knoppix Linux to boot the host from the USB drive. However, it requires minutes to start up given the need to boot and configure an entire operating system for the specific host being used. It does not fully support saving multimedia device state and therefore cannot suspend and resume audio and video applications correctly. MediaPod is designed specifically for a user's multimedia desktop applications, which lets it be much more lightweight. MediaPod 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. We build on our previous work providing portable, persistent computing using USB drives for Web browsing [12] and desktop computing [13], but extend it in MediaPod to support more general multimedia applications and multimedia devices such as audio which are crucial for migrating persistent multimedia applications between machines.

Thin clients, such as Sun's SunRay [14] and Microsoft's Windows Terminal Services [15], provide some of the benefits of MediaPod, leveraging a common environment everywhere. More recent thin clients such as THINC [6] support multimedia applications as well. These approaches require persistent network connectivity to a thin-client server. Unlike thin clients, since MediaPod moves the applications to the machine where the user is located, it can provide its functionality even when network connectivity is intermittent or not available.

VII. Conclusions

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

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

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

Acknowledgments

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

References

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

[2] 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.

[3] O. Laadan and J. Nieh, "Transparent Checkpoint- Restart of Multiple Processes on Commodity Operating Systems," in *Proceedings of the 2007 USENIX Annual Technical Conference*, Jun. 2007.

[4] O. Laadan, R. Baratto, D. Phung, S. Potter, and J. Nieh, "DejaView: A Personal Virtual Computer Recorder," in *Proceedings of the* 21^{th} *ACM Symposium on Operating Systems Principles (SOSP)*, Oct. 2007.

[5] 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, Sep. 2004.

[6] 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.

[7] Apple Computer, Inc., "iPod," http://www.apple.com/ipod/.

[8] "PortableApps.com," http://portableapps.com/.

[9] R. Chandra, N. Zeldovich, C. Sapuntzakis, and M. S. Lam, "The Collective: A Cache-Based System Management Architecture," in *2nd conference on Symposium on Networked Systems Design and Implementation*, Apr. 2005, pp. 259–272.

[10] Moka5, "Moka5 Technology Overview," http://www.moka5.com/node/381, November 2006.

[11] R. Cáceres, C. Carter, C. Narayanaswami, and M. Raghunath, "Reincarnating pcs with portable soulpads," in *MobiSys '05: Proceedings of the 3rd international conference on Mobile systems, applications, and services*. New York, NY, USA: ACM, 2005, pp. 65–78.

[12] 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.

[13] ——, "Highly Reliable Mobile Desktop Computing in Your Pocket," in *Proceedings of the IEEE Computer Society Signature Conference on Software Technology and Applications (COMPSAC)*, Sep. 2006.

[14] "Sun Ray Clients," http://www.sun.com/sunray.

[15] B. Cumberland, G. Carius, and A. Muir, *Microsoft Windows NT Server 4.0, Terminal Server Edition: Technical Reference*. Redmond, WA: Microsoft Press, Aug. 1999.