The Best of Both Worlds with On-Demand Virtualization

Thawan Kooburat, Michael Swift
University of Wisconsin-Madison
kooburat@cs.wisc.edu, swift@cs.wisc.edu

Abstract

Virtualization offers many benefits such as live migration, resource consolidation, and checkpointing. However, there are many cases where the overhead of virtualization is too high to justify for its merits. Most desktop and laptop PCs and many large web properties run natively because the benefits of virtualization are too small compared to the overhead.

We propose a new middle ground: *on-demand virtualization*, in which systems run natively when they need native performance or features but can be converted on-the-fly to run virtually when necessary. This enables the user or system administrator to leverage the most useful features of virtualization, such as or checkpointing or consolidating workloads, during off-peak hours without paying the overhead during peak usage.

We have developed a prototype of on-demand virtualization in Linux using the hibernate feature to transfer state between native execution and virtual execution, and find even networking applications can survive being virtualized.

1. Introduction

System virtualization offers many benefits in both data-center and desktop environments, such as live migration, multi-tenancy, checkpointing, and the ability to run different operating systems at once. These features enable hardware maintenance with little downtime, load balancing for better throughput or power efficiency, and safe software updates. As a result, virtualization has seen wide adoption for infrastructure Cloud providers, server consolidation in the enterprise, and for multi-OS application usage on the desktop.

However, when the benefit of virtualization is not always needed, the performance costs and limitations can be prohibitively high. Virtualization adds overhead to memory management, due to multiple levels of address translation, and to I/O, due to virtualizing devices. In addition, some classes of hardware, such as GPUs, are not well-supported by virtualization. While hardware support can reduce the overhead, virtual performance is still below native. As a result, most PC users and many large cloud providers, such as Google and Microsoft, do not run their core applications on virtual machines.

We propose that systems should provide the best of both worlds, with a seamless transition between the two: native performance on native hardware most of the time, with the ability to reap the benefits of virtualization when needed. An operating system runs natively, with full access to the performance and capabilities of its native platform, until virtualization is needed. At that point, the system starts a virtual machine monitor and migrates the OS onto virtual hardware, allowing full access to features such as migration and checkpointing. At any point, the OS can return to native execution on its original hardware. We call this approach *on-demand virtualization*, and it gives operating systems an ability to be virtualized without service disruption.

Reaping the benefits of virtualization without its costs has been proposed previously in other forms. Microvisors [14] provided the first step in the direction toward on-demand virtualization. However, they do not virtualize the entire system so their benefit is limited to online maintenance. Recent proposals suggest migration without virtualization [12] and multi-tenancy without virtualization [10]. These approaches are complementary to ours; they bring some of the benefits of virtualization all of the time, rather than most of the benefits some of the time.

We have built a prototype implementation of on-demand virtualization that converts a Linux OS running natively on Intel hardware into a virtual machine running inside KVM [11] on the same hardware. We use hibernation to capture OS and user-space state to disk, and then resume the system inside a virtual machine. Existing OS mechanisms, such as hot plugging and device indirection, allow the restored OS to talk to the virtual hardware. However, we find many complications at the lowest layers of the operating system with platform devices such as the PCI bus and interrupt controllers that are configurable only at boot time.

With our current prototype, we show that on-demand virtualization is possible with no special hardware and only minimal modifications that notify the guest OS about changed platform hardware. Our modified version of Linux can be virtualized on the fly, and running applications with open network connections survive the process.

2. Motivation

Despite its benefits, virtualization has not yet become ubiquitous. At both the high end, on data-center scale systems, and the low end, on PCs, virtualization often remains the exception rather than the rule.

2.1. Concerns with Virtualization

A primary concern over virtualization is performance: for systems that do not benefit from multi-tenancy, the overhead of virtualization is not captured by its benefits. Despite recent advances in hardware acceleration for virtualization [2, 15], the performance overhead of virtualization is prohibitive.

A primary source of overhead is I/O: virtual machine monitors (VMMs) implement virtual I/O devices [17] that require costly traps into the VMM on every I/O operation. While direct I/O reduces this cost by running drivers in the guest OS [5], it also reduces the usability of virtualization by complicating checkpoints and migration [9]. Furthermore, several common PC device functionalities, such as 3D graphics, wireless networking, and power management, are poorly supported by virtualization. For GPUs, performance drops dramatically due to the extra layers of abstraction [8]. Due to the lack of support from VMM, access to wireless networks is instead provided via Ethernet, so the guest cannot use its management tools to select a wireless network or enter network passwords. Similarly, power-management mechanisms such as ACPI [1] are not virtualized, preventing a guest OS from efficiently managing power usage on mobile devices. These devices are inherently difficult to virtualize because the hardware maintains significant amounts of configuration state.

A second large cost is memory virtualization: every memory access must be translated by an OS-level page table and a VMM-level page table. Nested page tables reduce the overhead by removing the maintenance cost of shadow page tables [2]. However, the additional cost of walking multiple page tables to satisfy a TLB miss can still reduce performance by ten percent or more [6]. In a large data center this loss could require purchasing additional machines to serve peak workloads.

2.2. Arguments for On-demand Virtualization

Despite these costs, virtualization still offers benefits if it is available when needed. We see uses for on-demand virtualization in both data centers and desktop environment. Within a data center the overhead of virtualization during peak workload hours can make native execution more efficient, despite lower utilization from single tenancy. When workloads fall, services can switch to a virtualized mode for consolidation onto fewer machines, increasing utilization and decreasing power consumption. With lower load and ample machines, the overhead of virtualization is not significant.

Additionally, deterministic VM record/replay can be useful to provide low-overhead dynamic program analysis [7]. When encountering an issue in a production environment, machines can be virtualized and start recording execution logs for online or offline analysis without modifying existing applications.

In a desktop environment, on-demand virtualization allows users to checkpoint a running system for backup which allows the system to roll back if the system fails or is compromised. In addition, users can use Internet suspend/resume [13] to migrate their environment to another computer for mobility.

While the overhead of virtualization can be reduced, for scenarios where it is only needed occasionally on-demand virtualization offers the best of both worlds: native performance with virtual benefits. For this to be useful, on-demand virtualization must have no overhead when running natively, and a seamless transition without service disruption when enabled.

3. Challenges

With on-demand virtualization, an operating system runs natively, with full access to the hardware, most of the time. When a user or administrator initiates virtualization, the OS pauses execution, starts a VMM and transfers its state into a virtual machine. The system replaces the real hardware with virtual hardware implemented by the VMM before resuming the OS.

This process raises numerous challenges due to the tight ties between the OS and the real hardware.

3.1. Capturing OS and Device State

The first step of virtualizing is capturing the state of a running operating system, so that it can be converted into a virtual machine. In a fully virtualized system, the VMM can capture guest state nonintrusively by pausing the guest and saving the contents of memory and virtual hardware. In contrast, on-demand virtualization requires that the OS which is about to be virtualized must capture *its own* state, including hardware state that may not easily be extracted. For example, the configuration state of *platform* devices, such as timers, interrupt controllers and I/O buses must be extracted and transferred into the virtual machine.

3.2. Discovering Devices inside the VM

We do not require that the VMM emulate the same set of devices as its host machine—to do otherwise would place a high burden on developing emulators for thousands of common devices. However, this approach requires that the OS must be able to find and use virtual hardware when it switches to a virtual machine.

For many peripherals, device hotplug allows the OS to find, initialize, and use new devices. However, there is a large set of platform devices that that the OS assumes can never change. As a result, current operating systems do not discover or configure these devices after boot.

3.3. Transferring Device State

After new devices have been discovered, the OS must configure and connect to the devices. Any OS structures referring to native devices must be transferred to the corresponding virtual device. For example, open files must be transferred from the real disk to a virtual disk so that file access continues after conversion. Similarly, network state, such as open connections, must also migrate to the virtual device. Thus, any driver or device state associate with the real hardware must be transferred to the virtual hardware after conversion.

4. Solutions

Despite the formidable challenges of on-demand virtualization, we observe that existing mechanisms provided by the OS can be repurposed to achieve this task. For example, hibernation allows the OS to save its state to disk, effectively capturing OS and device state. As a result, we modify the guest OS to leverage these mechanisms to simplify the conversion process. However, these are not sufficient to solve all the problems of on-demand virtualization.

4.1. Capturing OS State

We rely on the hibernation mechanism to transfer OS state from a native machine to a virtual machine: we suspend the native OS to disk, and then resume it within a virtual machine. Hibernation is normally used for zero-power suspend. However, it provides the essential capability needed for on-demand virtualization, which is to capture the state of the OS, including its drivers, and applications. The Linux hibernation mechanism snapshots the state of memory to disk, and relies on power management features to suspend and resume devices. Figure 2 (a) shows the steps of a normal hibernation.

During the suspend process, the hibernation code freezes processes to stop them from modifying memory. It asks drivers to suspend, causing them to disable the devices and copy their state into memory. Finally, the kernel shuts down its own activity except for one thread to copy memory to a *hibernate image*. Then, it unfreezes necessary subsystems to write the image to disk. The hibernate image contains the entire system state, exactly what is needed for running on a virtual machine.

During resume from hibernation, a *boot* kernel, which must be the same kernel that created the hibernate image, boots the machine and loads the image from disk. The boot kernel then suspends the devices it was using, and

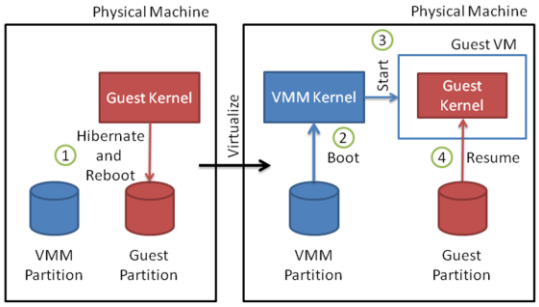

Figure 1: Overview of system operations

overwrites itself with the memory pages from the hibernate image to restore the *resume* kernel. The resume kernel then restores device states and resumes all process.

As shown in Figure 1, to use hibernation for on-demand virtualization, the system loads a VMM after restart, and starts the boot kernel inside a virtual machine.

4.2. Connecting with VM Devices

Hibernation can only resume an OS on a machine with a matching hardware profile: the resume kernel does not rescan hardware to see if anything changed. Here we present several methods for dealing with this problem.

Hotplug to attach new devices. Several classes of device support hotplug, meaning that they can be inserted or removed from a running system, and Linux will discover and configure the new device. The system can therefore transition hotplug-capable devices from physical to virtual hardware by virtually unplugging the physical devices and plugging in new virtual devices. Thus, we modify the resume kernel to rescan the PCI bus to discover virtual devices and attach them to the OS during the resume process. However, this process makes the device available, but does not connect them to any kernel structures or applications. For example, an open file system will not automatically attach to a new disk.

Logical devices to retain device state. Linux provides several logical devices that act as an interposition layer between the kernel and device drivers. These devices are created to provide aggregation or high availability. For example, the network *bonding driver* can switch a network stack between different devices, and the block *device mapper* can do the same for file systems and disks. Thus, these logical devices allow existing kernel structures to transfer their state from one device to another, and can therefore disconnect from the physical device and connect to the virtual ones.

Upon resume, the system can update each logical device to point to the new virtual hardware. For example, it updates the bonding driver to switch from the physical network device to the virtual network, and the device mapper to switch from a physical disk to a virtual disk. At this point, higher levels of the kernel and applications can continue to use these devices as if nothing happened.

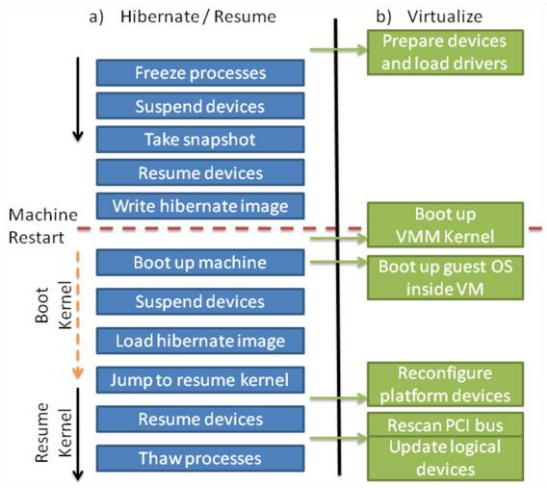

Figure 2: Comparison between the hibernate/resume process and the virtualize process.

Information passing from the boot kernel. In many cases, there are no logical devices to transfer state. This is particularly true for low-level platform devices, such as the ACPI controller that configures I/O buses and interrupts. Normally, Linux configures this device during boot to establish device address mappings and route interrupts. After boot, the code is discarded from memory to save space.

We therefore rely on the boot kernel to correctly configure the hardware, and then pass the configuration to the resume kernel, which can update its ACPI state.

Use of legacy devices. We can leverage virtual machine support for legacy interfaces for devices that do not support hotplug or logical devices. These interfaces, such as VGA mode, are left over from earlier PC architectures, and there is no variation between different models of devices. Hence, a generic driver can operate these devices, and the kernel can statically configure them. For example, all PC display adapters support VGA display mode. By switching to this mode prior to hibernate, the display works seamlessly across the conversion.

Preparation for virtualization. The operating system must make minimal preparations for virtualization. First, it must avoid using hardware that is unavailable in a virtual machine. If the capabilities of the virtual hardware, such as support for certain instructions or platform devices (*e.g.*, an IOMMU) are not available then the native system must take care not to use these features if it wishes to virtualize at some point. However, if the OS can operate without using devices like GPUs, they can be disabled just prior to conversion. Second, the OS must reserve resources, such as a disk partition, to contain the VMM. Third, the OS must run with logical devices described above at all times. Compared to full virtualization, these add only minimal overhead.

5. Implementation

We have implemented a prototype version of on-demand virtualization for the Linux 2.6.35 kernel using KVM as a virtual machine monitor. A machine must have at least two disk partitions: a VMM partition and a guest partition. The guest partition must have all the necessary drivers to run on both the physical machine and the virtual machine. Figure 1 shows the overall process of the conversion.

We modify the TuxOnIce hibernation code [3] as shown in Figure 2(b). Before hibernation, the kernel performs extra tasks to prepare for running inside a VM, including loading drivers for virtual devices. After the resume kernel is restored inside a VM, the kernel reconfigures platform devices, rescans for PCI devices and updates logical devices.

5.1. Dealing with Devices

Each class of devices requires a slightly different solution for transitioning from physical to virtual hardware.

Block devices. The device mapper provides a logical block device that can redirect requests to other block devices based on a mapping table. In order to reconnect file systems to the correct partition, the kernel records file system UUIDs for each partition before hibernation. When the resume kernel discovers VM disk controllers, new block devices will be added to the kernel device tree. By comparing each block device's UUID with the previously recorded one, the kernel can find virtual devices corresponding to the old physical partitions and updates mapping tables accordingly.

Network interfaces. We use the Linux bonding driver to create a logical network interface. Before hibernate, the bonding driver attaches only to the physical network card. After resume, the kernel discovers a virtual network card and attaches it to the bonding driver. The bonding driver finds that the physical device is unavailable and sends all packets to the virtual NIC instead. The use of bonding driver is inspired by work on VM live migration [18].

Platform devices. We modify the boot kernel, used during resume, to pass on the IOAPIC and ACPI timer configuration data to the resume kernel. However, we cannot reprogram PCI interrupt routing in the resume kernel, so instead we update the kernel's routing tables with the boot kernel's interrupt configuration.

5.2. Preliminary Results

Our implementation is preliminary and only supports one-way conversion from native to virtual execution. The conversion takes about 90 seconds, and active ssh/scp connections remain open. The majority of time is spent in hibernate/resume process and rebooting the machine.

We plan to improve the conversion speed by storing

the hibernate image in RAM and using kexec [16] to reboot into VMM partition since it can preserve RAM content. In addition, we also need to add support for *devirtualizing* by transferring a VM back to run on its original physical machine. We can rely on the fact that we only detach original physical devices from their drivers to make them inactive prior to conversion. This assumption will simplify devirtualization since we can enable the devices by attaching back their drivers.

6. Related Work

There are many approaches which try to bring in the benefits of virtualization without its cost. Here we list several ideas which try to achieve this goal.

The first step toward on-demand virtualization was demonstrated by microvisors on an Alpha processor system [14]. However, their goal is to support online maintenance instead of enabling other benefits of virtualization. As a result, they did not virtualize devices or memory, and relied instead on having extra hardware available.

VMware Converter [4] is capable of cloning a physical machine into a VM. An agent copies data from a physical machine into a VM image. However, it does not include running state of the physical machine, such as open applications. In addition, the VM must be rebooted first to discover new hardware.

Another line of work is to provide some benefits of virtualization without relying on a VMM. OS live migration [12] adds whole-system migration to an OS. In order to achieve this, each class of device must provide an import/export interface to facilitate the transfer of abstract device state during the migration. In contrast, we rely on power management and logical devices because they require no modification to existing drivers.

NoHype [10] uses hardware features to partition resources such as processors, memory and devices between each guest OS. This approach allows guest OS to execute natively without VMM intervention most of the time. However, NoHype requires additional hardware support and does not address hardware differences in the case of live migration. In contrast, on-demand virtualization works with existing hardware, and relies on virtual hardware to mask device differences.

7. Conclusion

We propose on-demand virtualization as a mechanism for reaping the benefits of virtualization in environments where its merits do not justify for paying for its cost all the time. Thus, it can be useful in many cases: datacenters providing machines with native performance and PCs requiring devices that are not emulated by a VMM.

We showed that on-demand virtualization is possible by implementing a prototype in Linux that can virtualize while retaining network connectivity. We found that platform devices impose a major obstacle because they were not designed to be reconfigured after the boot phase. However, retaining OS and device driver states across the conversion is not a major issue as hibernation and logical devices can be used to solve this problem.

8. Acknowledgements

We would like to thank people in our Sonar systems group and anonymous reviewers for valuable feedback. Swift has a significant financial interest in Microsoft.

References

[1] ACPI Specification `http://www.acpi.info/DOWNLOADS/ACPIspec40a.pdf`.

[2] AMD-V Nested Paging `http://developer.amd.com/assets/NPT-WP-1%201-final-TM.pdf`.

[3] Linux software suspend `http://www.tuxonice.net/`.

[4] VMware vCenter Converter `http://www.vmware.com/products/converter/`.

[5] D. Abramson, J. Jackson, S. Muthrasanallur, et al. Intel Virtualization Technology for Directed I/O, *Intel Technology Journal*, 2006.

[6] R. Bhargava, B. Serebrin, F. Spadini, et al. Accelerating two-dimensional page walks for virtualized systems, *ASPLOS*, 2008.

[7] J. Chow, T. Garfinkel, and P.M. Chen. Decoupling dynamic program analysis from execution in virtual environments, *USENIX*, 2008.

[8] M. Dowty and J. Sugerman. GPU virtualization on VMware's hosted I/O architecture, *WIOV*, 2008.

[9] A. Kadav and M. Swift. Live migration of direct-access devices, *SIGOPS Oper. Syst. Rev.*, 2009.

[10] E. Keller, J. Szefer, J. Rexford, et al. NoHype: virtualized cloud infrastructure without the virtualization, *ISCA*, 2010.

[11] A. Kivity, Y. Kamay, D. Laor, et al. kvm: the Linux virtual machine monitor, *OLS*, 2007.

[12] M. Kozuch, M. Kaminsky, and M.P. Ryan. Migration without Virtualization, *HotOS*, 2009.

[13] M. Kozuch and M. Satyanarayanan. Internet suspend/resume, *HotMobile*, 2002.

[14] D.E. Lowell, Y. Saito, and E.J. Samberg. Devirtualizable virtual machines enabling general, single-node, online maintenance, *ASPLOS*, 2004.

[15] G. Neiger, A. Santoni, F. Leung, et al. Intel Virtualization Technology: Hardware Support for Efficient Processor Virtualization, *Intel Technology Journal*, 2006.

[16] H. Nellitheertha. Reboot Linux faster using kexec `http://www.ibm.com/developerworks/linux/library/l-kexec.html`.

[17] J. Sugerman, G. Venkitachalam, and B.-H. Lim. Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor, *USENIX*, 2001.

[18] E. Zhai, G.D. Cummings, and Y. Dong. Live migration with pass-through device for Linux VM, *OLS*, 2008.