Proceedings of the Linux Symposium

June 27th–30th, 2007
Ottawa, Ontario
Canada

Conference Organizers

Andrew J. Hutton, *Steamballoon, Inc.*
C. Craig Ross, *Linux Symposium*

Review Committee

Andrew J. Hutton, *Steamballoon, Inc.*
Dirk Hohndel, *Intel*
Martin Bligh, *Google*
Gerrit Huizenga, *IBM*
Dave Jones, *Red Hat, Inc.*
C. Craig Ross, *Linux Symposium*

Proceedings Formatting Team

John W. Lockhart, *Red Hat, Inc.*
Gurhan Ozen, *Red Hat, Inc.*
John Feeney, *Red Hat, Inc.*
Len DiMaggio, *Red Hat, Inc.*
John Poelstra, *Red Hat, Inc.*

Why Virtualization Fragmentation Sucks

Justin M. Forbes
rPath, Inc.
jmforbes@rpath.com

Abstract

Mass adoption of virtualization is upon us. A plethora of virtualization vendors have entered the market. Each has a slightly different set of features, disk formats, configuration files, and guest kernel drivers. As concepts such as Virtual Appliances become mainstream, software vendors are faced with new challenges. Previously, software vendors had to port their application to multiple operating systems: Solaris, Linux, AIX, etc. The new "port" becomes one where software vendors will be expected to produce images that drop-in to VMware, Xen, Parallels, SLES, RHEL, and even Microsoft Virtual Server.

This paper will explore the state of existing virtualization technology in meeting the goal of providing ready-to-run guest images. This includes: comparing, contrasting, and poking fun at virtual disk formats; bemoaning the assortment of kernel drivers needed to improve performance in a guest (vmware-tools, paravirt drivers...); dark muttering about incompatibilities between Xen guests and hosts; and lamenting all the different configuration files that define a guest.

Virtualization has moved into the mainstream of computing. Most businesses are no longer asking if they should deploy a virtualization solution; they are instead asking which vendors support the technologies they have already implemented. In many ways, this is a great new age for software vendors. With virtualization technology so common, it makes it possible to reduce the costs associated with supporting a product on a large assortment of operating systems. Software vendors can now bundle just the right amount of operating system required to support their software application in a software appliance model. Distributing a software appliance allows vendors to fully certify one stack, without the worry about which packages or versions a particular operating system distribution chooses to ship. The extensive QA and support models that go along with shipping a separate application are drastically simplified. Software vendors no longer need to decide which operating systems to support, the new question is "which virtualization technologies do I support?"

This question should be easy to answer. Unfortunately, it is becoming increasingly difficult. It does not have to be. The hypervisor is the new platform, and many vendors have entered the market, with more vendors on the way. Each vendor offers products to meet a similar requirement: Allow fully isolated containers, or virtual machines, to consume the resources they require, without stepping on other containers.

The advantages of virtualization are many. To the software consumer, virtualization takes away much of the concern surrounding the full stack. The fact that different applications may require conflicting library support is no longer a concern. The ability to better manage resources, increasing utilization of hardware without increasing risk of multiple applications stepping on each other is a tremendous benefit. Live migration, the ability to move virtual machines across physical hosts in real time, substantially increases availability. It is possible to maintain both performance and availability with a fraction of the hardware resources that were once required.

In the software appliance model, vendor relationships are improved as the customer can go to a single vendor for support, without playing intermediary between the application vendor, tools vendors, and operating system vendors. Software consumers need not worry about testing and patching large numbers of security and bug fixes for software which gets installed with most general purpose operating systems, but is never used or installed within their virtualized environment.

To the software producer and distributor, virtualization means simplified development, QA, and testing cycles. When the application ships with its full stack, all of the time required to ensure compatibility with any number

of supported platforms goes away. The little compromises required to make sure that applications run reliably across all platforms have a tendency to ensure that those applications do not run optimally on any platform, or increase the code complexity exponentially. This pain goes away when the platform is a part of the application, and unnecessary components which exist in a general purpose application are no longer present. Software vendors can distribute fully tested and integrated appliances, and know exactly what components are on the appliance without concern that a critical component was updated to fix a bug in some other application unrelated to what the vendor's application provides or supports.

With so many benefits to virtualization, and so many options available to the consumer, what could possibly go wrong? No one ever likes the answers to that question. The proliferation of options in the virtualization market has brought a new kind of insanity for software vendors. "Which technologies do I provide ready-to-run guest images for?" The simple answer should be all of the major players. Providing choice to customers with minimal effort is a great thing. Unfortunately, the effort is not so minimal at the moment.

Each vendor has a business need to distinguish itself by providing unique features or a unique combination of features. Unfortunately for the consumer, even generic features are provided in unique ways by each virtualization provider, needlessly complicating life both for the software vendor and the end user.

This doesn't have to be so hard.

1 Disk Formats

There are several possibilities for the virtual machine disk format, none of which is universally supported. While most of the commonly-used formats offer a similar set of features, including sparse allocation and copy on write or some form of snapshot, they are not directly interchangeable. VMware's VMDK and Microsoft's VHD are among the most common formats supported. The QCOW format also offers similar functionality, though it should be noted that there are now multiple incompatible versions of the QCOW formats, making QCOW more of a format family than a format. The disk format will typically include the raw file systems or hard disk image, a bit of metadata describing the supported features, versioning, and creation method, as well as specific implementation metadata in the case of sparse allocation or copy on write. It may also contain information used to define the characteristics of the virtual machine associated with the images.

While VMDK, VHD, and QCOW are among the most commonly supported disk formats, they are far from universal. Some technologies still require raw file system images or other proprietary formats. The good news here is that conversion utilities exist for most formats available. If the desire is to have a single reference image that can be packaged for many different virtualization technologies, perhaps a raw file system or hard disk image is the best choice, as those can be directly converted to most other formats with little effort. Still, the question remains, why does this have to be so complicated? With a similar feature set among the most popular virtual disk formats, what is it that separates them? The difference lies in the metadata implementations. Perhaps in the future, common ground can be established and the disk format conversions will no longer be necessary.

2 Virtual Machine Configuration

When we look at what defines a virtual machine, there are many of the same pieces we find in stand-alone hardware. Along with the virtual disk or physical file systems, a virtual machine is allocated the basic resources required for a machine to function and be useful. This includes one or more virtual processors, a chunk of memory, and virtual or physical I/O devices. These are the core building blocks, and will differ among deployments of a given appliance based on customer requirements and available resources. In addition to these simple building blocks, there are typically a number of additional configuration options which help to define the virtual machine. These include migration and availability options, crash or debugging behavior, and console or terminal definitions. All of this configuration is specific to the actual deployment of an image, and should be defined within the hypervisor, controlling domain, or management infrastructure.

In addition to these site-specific configuration options, many virtualization vendors provide options for which kernel to boot, the initial ram disk or initrd image to be used, and other typical boot options. While there are specific circumstances where keeping such configuration options separate from the virtual machine container itself would be desirable, it is more frequently a

management headache for the consumer of virtual appliances. When the boot kernel is defined—or worse, located—outside of the central image, there is no simple way for an image to be self-maintained. While mechanisms exist for updating a virtual image similar to those for updating physical hosts, the guest does not typically have permission or ability to write to the host's file systems for such activities as updating the guest kernel or initrd. Simply put, any effective software appliance must be self-contained, and use standard tools for managing the contents of the appliance itself. Ideally the bootloader is also contained in the appliance image, but at the very minimum, a bootloader installed on the hypervisor should be able to read and interpret boot configuration information from within a guest image.

3 Paravirtualization vs. Full Virtualization

Both paravirtualized and fully virtualized machines have been around for quite some time, and each has distinct advantages. For the paravirtualized machine, the guest operating system is fully aware that it is operating under hypervisor control. The kernel has been modified to work directly with the hypervisor, and as much as possible to avoid instructions that are expensive to virtualize. The largest advantage to paravirtualization is performance. Generally speaking, a paravirtualized guest will outperform a fully virtualized guest on the same hardware, often by a substantial margin. With this being the case, why isn't every guest paravirtualized?

There are several obstacles to paravirtualization. The source level changes required to build a paravirtualized kernel can be large and invasive, in many cases tens of thousands of lines of code. These changes occur in core kernel code and can be much more complex than higher level driver code. While the nature of the changes required to support a given hypervisor can be very similar, the implementation details and ABI will vary from vendor to vendor, and even among versions of the hypervisor from the same vendor. It is not uncommon for a paravirtualization patch set to be several revisions behind the latest upstream kernel, or skip upstream revisions all together. This is unlikely to change until a given implementation has been accepted into the upstream kernel. The resources required to maintain such a large patch set outside of the upstream tree are considerable; maintaining the same code in the upstream kernel requires much fewer resources. There is also the small matter of guest operating systems which are not open source, or whose license does not allow the redistribution of changes. In these instances, paravirtualization is extremely difficult, if not impossible.

In the fully virtualized machine, the guest operating system does not need to know that it is being virtualized at all. The kernel operates exactly as it would on standard hardware. The hypervisor will trap necessary instructions and virtualize them without assistance from the guest. Standard devices such as network and block drivers are typically presented as virtual implementations of fairly well-supported physical devices to the guest so that no special drivers are needed. Modern CPUs include hardware support for virtualization, which improves performance and compatibility. While this method is an effective way to ensure compatibility with a large variety of guest operating systems, there is a high overhead in trapping all of the necessary instructions. To help with this performance problem, it is common for the hypervisor to support a number of paravirtualized device drivers. By replacing the common and well supported device drivers with new devices which are aware of the hypervisor, certain expensive instructions can be avoided and performance is improved dramatically. Typically, a virtualized guest with paravirtualized drivers will achieve performance much closer to that of a true paravirtualized guest.

This is another area of difficulty for the software appliance distributor. The vast number of virtualization vendors each have their own paravirtualized kernels, drivers, or guest tools. Some of these are open source, some are not. Regardless of source code availability, there is the question of which kernel versions these drivers will build against, or might be supported with. It is not uncommon for a vendor to have no working driver or tool set for two or three of the most recent upstream kernel versions, leaving them outside of the upstream stable support cycle all together. It is also possible that upstream kernel versions are skipped over entirely, making it difficult to find a common kernel version that can be supported by all of the desired virtualization targets that a software vendor might have. Luckily, it is quite possible to have a user space that supports a variety of kernel releases, ensuring that only the kernel version and associated virtualization drivers or tools are the only substantial changes between images. This leaves most of the QA and testing work intact, and still provides a substantial support savings over supporting entirely different general purpose operating systems.

It is hoped that some of these problems can be addressed generically. Changes to the Linux kernel are being made which make it possible to eventually build a single kernel which supports multiple virtualization solutions. Examples include `paravirt_ops` which shipped in the 2.6.20 kernel, and the VMI interface on top of `paravirt_ops` which is included in the 2.6.21 Linux kernel. While the initial groundwork for `paravirt_ops` and the VMI layer are present in mainline kernels, there is still a lot of work remaining to make them beneficial to the vast majority of users. In the short term, we have simply added another yet option for building virtualization solutions. Until stable releases of the majority players in virtualization have patches or products to support these new kernel interfaces available, and the older products are phased out, these interfaces simply represent one more option that must be supported. It really does have to get worse before it gets better.

Another proposal that has been floating around is a set of common paravirtualized drivers, which could be built as modules and provide many of the benefits associated with vendor provided tools and drivers while decreasing the number of configurations which must be built and supported. Unfortunately this proposal is in early stages and faces several obstacles. For instance, Xen provides xenbus instead of relying on the PCI specification for I/O virtualization. There is also the question of finding a common ground for block I/O, as many virtualization vendors have put considerable effort into optimizing block I/O for virtualized guests, and these implementations are not guaranteed to be compatible with one another. Still, if a agreement could be reached, the result would be basic paravirtualized drivers which could be maintained upstream, and present in the majority of Linux vendor kernels without overhead. Virtualization providers would still have the option of further optimization by using platform-specific drivers, but end users would see less basic overhead when using an image that for one reason or another could not easily deploy the platform-specific drivers.

4 Architectural incompatibility

Even when dealing with a single virtualization vendor, there are a few architectural anomalies to keep in mind. One particularly painful current example is PAE support. When dealing with 32-bit systems, both guest and host, there is a question of exactly how much memory is supported. In order for a 32-bit system to address more than 4GB of memory, PAE is supported on most modern x86 processors. In Linux, PAE is support is determined at kernel build time. Unfortunately a PAE-enabled kernel will not boot on physical or virtual hardware which does not actually support PAE mode. This is because PAE mode causes fairly significant changes to the page table structure regardless of the amount of actual memory in a system. This is important to know because several mainstream virtualization solutions take different approaches to PAE support. In the VMware case, PAE is supported on the host in modern versions, meaning the hypervisor can address more than 4GB of memory, but the guest does not support PAE mode even in instances where the host has more than 4GB available. While it is not a horrible limitation to say that a guest can only support less than 4GB of memory, it also means that a guest kernel cannot be built with PAE support and still work on all VMware deployments. (Whether PAE is supported in a VMware guest depends on the host kernel and on image-specific configuration settings.)

In the Xen case, the rules are less clear-cut. Xen has essentially three parts: the hypervisor, the domain 0 kernel, and the guest or unprivileged domain kernel. The hypervisor and the domain 0 kernel must always have matching PAE support, meaning if the domain 0 kernel is built with PAE support, the xen hypervisor must be built with PAE support as well. For guest domains, the situation is split between paravirtualized guests and hardware virtual machines using the hardware virtualization features of modern CPUs from Intel and AMD. A hardware virtualized machine can run with PAE either enabled or disabled, regardless of the domain 0 and hypervisor. For paravirtualized guest domains, the kernel must be built with the same PAE features of the hypervisor and domain 0. It is not possible to mix and match PAE between paravirtualized guests and the hypervisor with current releases of Xen. While it would be simple enough to say that PAE support should always be enabled, there are a few obstacles to this. Some hardware does not support PAE mode, particularly a large number of laptops with Intel Pentium M CPUs. Additionally, there are existing Xen hosts which do not support PAE for one reason or another. It is believed that over time non PAE implementations of 32-bit Xen will fall out of use, but the current issue is real and still somewhat common.

Guest architecture support will also vary according to

the hypervisor used. While many hypervisors currently available offer support for both 64-bit and 32-bit guests under a 64-bit hypervisor, several do not. Although the hardware transition is nearly complete (it is difficult to find mainstream desktops or servers which do not support x86_64 these days), it will still be some time before older 32-bit hardware is retired from use, and even longer before 32-bit applications are no longer supported by many vendors. This means that it may be necessary for software appliance vendors to offer both 32-bit and 64-bit guest images if they wish to ensure compatibility with the largest number of virtualization technologies. For applications which are only available in 32-bit flavors, it means that guests will have to run a 64-bit kernel in some circumstances, though a 32-bit user space is generally supported.

Conclusion

With well over a dozen virtualization solutions in use today, and more on the way, there is a lot of choice available to the consumer. Choice can be a double-edged sword. Competition drives innovation, we are seeing results from this at a rather fast pace today. Competition in the virtualization space also has the potential of driving support overhead to painful levels. Differing approaches to virtualization can ensure that the correct tool is available for any given job, but if the tool is too difficult to use, it is (more often than not) simply ignored in favor of the easier option.

Software vendors can leverage the benefits of virtual appliances now. While there are certainly obstacles to be overcome, they are not insurmountable. The advantages to a software appliance model are great, and the pains associated with this growth in virtualization technologies have to be addressed.

As developers, providers, and integrators of virtualization technology, we have to address these issues without allowing things to get out of hand. We need to look beyond the the technology itself, and see how it will be used. We need to make sure that the technology is consumable without a massive amount of effort from the consumers.