Paravirtualization or Virtualization
Xen or KVM

In this paper I discuss the differences between paravirtualization and virtualization, and use them to explain the differences between xen and kvm systems.

Adrián Hernando
28/08/2012
1. An introduction to virtualization

The most intuitive way of thinking about virtualization is an emulation of an environment. This concept is similar to virtual reality, which means emulating reality on a computer. As the characters of The Matrix movie lived in a virtual reality, computer programs can run on different environments independently of the hardware or computer that supports it.

Virtualization exists mainly for two reasons:

- For sharing the computer resources and using them in a more optimal way. Computers nowadays are much more powerful than they used to be and so almost never are used at full capacity. Virtualization allows us to create different environments for multiple operating systems and have them running at the same time. This means a better use of resources, which lead to less hardware purchase. For example, if a corporation needs five different servers, without virtualization it would need five computers, each with one server. With virtualization, the five servers can run on the same machine.

- Emulating a friendly environment for a program inside another environment where it can’t run. For example, we can’t run Windows’ programs in an Ubuntu operating system. To solve this, we create using virtualization an emulation of the windows environment, so the program thinks he’s on a windows operating system and runs as if it were true. It is also what the JVM (Java Virtual Machine) uses for java’s programs, so they can run everywhere.

Although IBM already used some kind of virtualization fifty years ago, it’s becoming more and more popular these last years, thanks to cloud computing technologies and the expansion of the Internet. But how does virtualization work? It depends on the way you apply it. There are mainly two ways of doing it: hardware virtualization (also known as “full virtualization” or simply “virtualization”) and paravirtualization. In this paper I’ll explain both of them and then proceed to discuss their differences.

The programs used for virtualization are called hypervisors. A hypervisor is a layer between the guest operating system (the one which environment we want to emulate) and the host operating system (the one that’s interacting directly with the hardware) or directly over the physical layer. The most widely used hypervisors are Xen and KVM. As they represent respectively both paradigms of virtualization, paravirtualization and hardware virtualization, I’ll be discussing them too.
2. Hardware virtualization

Hardware virtualization or complete virtualization means different and isolated containers for each of the guest operating systems. Unlike operating system virtualization, applications don’t think they are running on their operating systems, but they are in fact running on it. With hardware virtualization an isolated environment is created for the guest operating system to run, and it have no connection to the rest of the system. It is best explained in Virtualization for Dummies (2008):

“Hardware emulation supports actual guest operating systems; the applications running in each guest operating system are running in truly isolated operating environments.

The guest operating system makes system calls to the emulated hardware. System calls that would actually interact with underlying hardware are intercepted by the virtualization hypervisor, which maps them onto the real underlying hardware. Hardware emulation comes in two flavors: In one, the hypervisor runs as an application on top of a host operating system; in the other, the hypervisor resides on bare-metal hardware and provides the entire operating environment for guest operating systems. The latter flavor of hardware emulation is used in production environments where the penalty overhead imposed by running a host operating system in addition to a virtualization software layer would be unacceptable.”¹

As we can see, each isolated guest OS have no connection to the rest of the machine but by communicating to the hypervisor. Every system call must be done through it. This means more security, as the systems cannot interfere with each other, but it also means a loss of resources, as everything must go through more layers.

The main problem with hardware virtualization is the fact that everything must go through the hypervisor. This means the guest operating systems can only use the resources it provides, which means they can only access the hardware through the interfaces of the hypervisor. This limits the choices and hardware updates, as they can only be as advanced as the hypervisor is. If the hypervisor doesn’t include the drivers for a new device, there’s no way the guest operating systems can use it, so the user must wait for a new version of the hypervisor to be released.

3. Paravirtualization

Instead of separating guest operating systems in isolated containers, paravirtualization acts more like an administrator, distributing the resources of the system and managing system calls for the operating systems. It derives all the communication through a dummy device driver and then redirects it to a privileged guest. Again, this is best explained in Virtualization for Dummies (2008):

“One distinguishing feature of paravirtualization is that it doesn’t provide its own device drivers. It inserts a dummy device driver – called a front-end driver – in guest virtual machines. That front-end driver communicates with the back-end driver that resides in the privileged guest (known as DomainO in Xen and the Root Partition in Windows Server Virtualization). The privileged guest contains the actual device drivers for the underlying hardware, and it coordinates communication between the back-end drivers and the actual device drivers.”

Although this may seem more complicated, in fact it represents an advantage, as any driver updated in the privileged guest can be used by all of the other operating systems. This means all the operating systems can use all the hardware the privileged guest is connected to. This also means that guest operating systems must be modified in order to run over the hypervisor.

The next image represents how paravirtualization works in Xen systems:

As we can see in the image, guest operating systems must communicate to the privileged operating system in order to make system calls or interact with the hardware. The front-end device drivers of the guest OS communicate with the back-end of the Dom0 (in Xen systems), who then uses the native device driver to communicate with the hardware. The VMM (virtual machine monitor) conducts this communication between the guest operating system and the hardware.

---

4. Contrasting virtualization and paravirtualization

Most authors agree that choosing between virtualization and paravirtualization is not a matter of which one is better, but of which one solves your problem in the most efficient way. Each of them have some advantages and some disadvantages which I’ll proceed to discuss:

First of all, about speed and performance. When speaking about performance in computers, one can’t usually guarantee anything, as it depends on too many factors like which program are you trying to run or the computer in which you are doing the tests. However, most articles usually give this point to paravirtualization. As we can see in the “Short IT recipes” blog, full and paravirtualization were tested and the latter resulted faster. These were the results of the test:

**nbench results**

<table>
<thead>
<tr>
<th>Index (higher is better)</th>
<th>Full virtualization</th>
<th>Paravirtualization</th>
<th>dom0</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEMORY INDEX 10.892</td>
<td>18.991</td>
<td>21.045</td>
<td></td>
</tr>
<tr>
<td>INTEGER INDEX 16.921</td>
<td>17.962</td>
<td>17.963</td>
<td></td>
</tr>
<tr>
<td>FLOATING-POINT INDEX 28.499</td>
<td>28.781</td>
<td>29.917</td>
<td></td>
</tr>
</tbody>
</table>


**Primes results**

<table>
<thead>
<tr>
<th>Time [s] (lower is better)</th>
<th>Full virtualization</th>
<th>Paravirtualization</th>
<th>dom0*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time 116.423</td>
<td>112.432</td>
<td>104.57</td>
<td></td>
</tr>
</tbody>
</table>

* - four CPUs were used.

Update 17/03/09: I tested fully virtualized ubuntu-server 6.10 using primes, and got 113.237 s.

**bench results**

<table>
<thead>
<tr>
<th>Index (higher is better)</th>
<th>Full virtualization</th>
<th>Paravirtualization</th>
<th>dom0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arithmetic Test (type = double) 409.6</td>
<td>411.6</td>
<td>461.3</td>
<td></td>
</tr>
<tr>
<td>Dhrystone 2 without register variables 484.4</td>
<td>489.8</td>
<td>539.2</td>
<td></td>
</tr>
<tr>
<td>Process Creation Test [ips] 614.0</td>
<td>3176.7</td>
<td>2855.6</td>
<td></td>
</tr>
<tr>
<td>Shell scripts (4 concurrent) [om] 190.6</td>
<td>1250.0</td>
<td>1261.3</td>
<td></td>
</tr>
</tbody>
</table>

---

As we can see, paravirtualization performance was slightly better than full virtualization. Wpeckham seems to agree on that on his post in linuxquestions.org\(^5\), pointing out that in paravirtualization the guest OS is modified to help the host layer. But as I said, it always depends on many factors, so different tests may throw different results.

- Paravirtualization also seems better for drivers and updates. Any update or driver installed in the privileged guest OS affects the whole system, which make maintenance much easier.

- Security is a factor in which full virtualization is better than paravirtualization. As it creates isolated containers, no operating system can hurt other. This means if one operating system crashes, the rest can go on working as if nothing happened. On the other hand, in paravirtualization the systems share some common things, as the privileged guest operating system. This means a failure can extend to the rest of the system.

- Full virtualization is also cleaner, meaning its diagram is much easier. Each isolated OS communicates with the hypervisor, which communicates with the hardware layer. In paravirtualization, guest operating systems must be modified to connect to the privileged guest operating system dummy drivers, and then it must communicate to the real drivers. Then it can communicate with the hypervisor which finally gets to the hardware.

5. Contrasting xen and kvm

As I said in the introduction, Xen and KVM are two of the main hypervisors nowadays. They also represent the two types of virtualization: Xen works with paravirtualization and KVM with full virtualization. This means all the comparisons and contrasts I made between virtualization and paravirtualization are generally valid as a comparison between Xen and KVM.

KVM is a more or less recent hypervisor developed by Sun Microsystems (now owned by Oracle). It’s main advantage lies in being included in the Linux kernel code. This means it’s updated with linux and any distro can automatically use it without further modifications. This has proved to be a great advantage, and have helped spreading KVM faster through companies.

On the other hand, Xen was one of the first hypervisors, and because of that many of the virtualization technology is based on it. It is widely spread through the world and supported by many companies. Unlike KVM, Xen is not included in the Linux kernel, but recent news say it will be supported in Linux 3.x releases⁶. As cdubuque says⁷, this is not the same as KVM inclusion in the kernel, but it will help working with Xen in linux systems, which is a great step forward for both Xen and Linux.

As happened with virtualization and paravirtualization, KVM seems more reliable and easier to use, while Xen seems faster and easier to maintain. Andrea Chierichi and Ricardo Veraldi explain in their article⁸ using qualitative and quantitative tests the differences between Xen and KVM. They proved KVM was easy to install and reliable, pointing out that it never crashed, and found out that KVM performance was comparable to Xen’s, and sometimes even better. However, they recognized some problems with network performance and disk I/O, and finally came to the conclusion that though KVM was promising, Xen is still a better choice.

---

⁷ cdubuque, Simon (2012) KVM is Linux. Xen is not., chucknology.com, posted February 2 2012 and retrieved august 28 from (http://chucknology.com/2012/02/02/kvm-is-linux-xen-is-not/)
6. Conclusion

After comparing paravirtualization and full virtualization, Xen and KVM, one cannot avoid the obvious “it depends on the context”. Both paravirtualization and hardware virtualization seem to be reliable solutions to the virtualization problem, and both Xen and KVM have done an excellent work implementing these systems.

Then, what made Amazon chose Xen over KVM? First of all it should be mentioned that Amazon uses a heavily modified version of Xen, and also that Xen is older than KVM. This means when Amazon had to choose a hypervisor, they chose Xen over KVM because KVM didn’t exist or wasn’t well-known back then. But why haven’t they changed to KVM, which by being supported by Red Hat have started a great competition with Xen? Drue Reeves asks the same question in his blog⁹, coming to the conclusion that Amazon will be forced in the next few years to consider that option. Right now they have a development team and enough support for staying with Xen, and while the cloud computing competition doesn’t get dirty, they can continue that way. But eventually, they may be forced to change to the well supported KVM.

Virtualization is a relatively new technology whose expansion thanks to cloud computing have no precedents. New technologies and solutions are developed every day, and usually quantitative differences between systems are marginal, as happen with paravirtualization and virtualization. At this point, the only way of knowing the best solution is to wait for the winners, assuming there will be any, to write history.

---

7. Bibliography

- cdubuque, Simon (2012) KVM is Linux. Xen is not., chucknology.com, posted February 2 2012 and retrieved august 28 from http://chucknology.com/2012/02/02/kvm-is-linux-xen-is-not/