Virtual Machines for the Intel x86 Architecture

Sarah Lowman

sarah@lowmanio.co.uk

November 2007

A Virtual Machine (VM) is a software implementation of hardware resources, including memory, processor(s) and storage devices. It was originally defined by as *"an efficient, isolated duplicate of a real machine."*[1] A Virtual Machine Monitor (also known as a hypervisor) is the software which manages the VM by way of virtualisation. There can be many VMs running on the host computer, all with different operating systems and different intended architectures.

There are many types of VM techniques, two of which are full virtualisation and paravirtualisation. Full virtualisation requires no adjustment to the guest operating system, which allows for complete emulation of the computer. An example of this is VMware. In paravirtualisation the guest operating system is modified (and made aware that it is virtualized) so that some of the work is done in the VMM. Xen is the leading paravirtualisation VMM.

There are many other virtualisation techniques such as soft partitioning, operating system-level virtualisation and emulation. User Mode Linux uses a different technique that does not involve a VMM. It is a virtual Operating System itself, rather than a VMM, which is run in user-mode. The guest operating system talks directly to the host, as it runs as a normal application in user space. However, this can still be seen as paravirtualisation, the host operating system being the VMM.

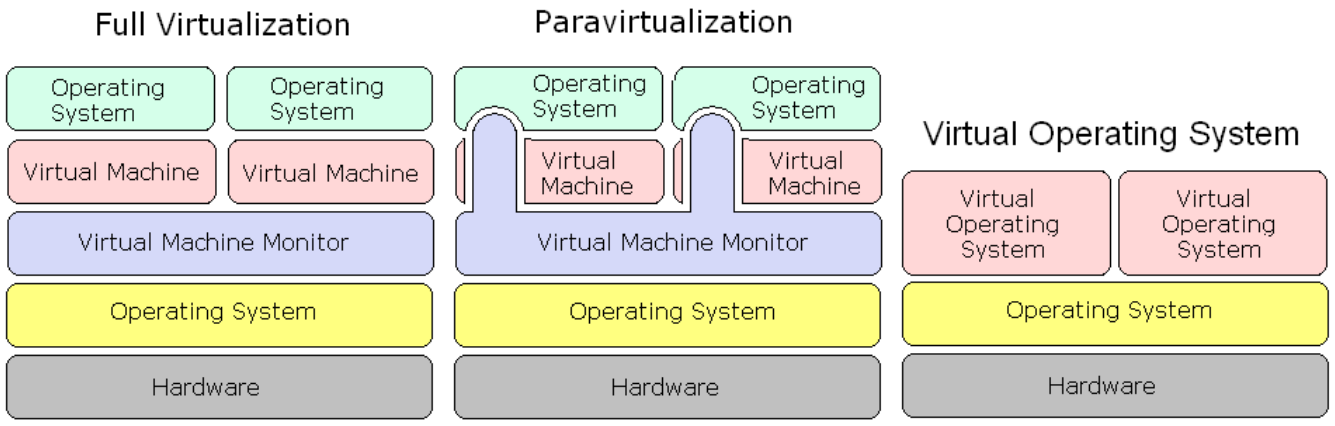

Figure 1: Different types of virtualization, representing VMware, Xen and UML respectively

[1] Gerald J. Popek and Robert P. Goldberg (1974) **Formal requirements for virtualizable third generation architectures**

In 2005 Intel introduced Intel® Virtualization Technology, a set of improvements on the processor which helps software based virtualization. AMD have also introduced enhancements called AMD-V present on all Athlon-64 and Opteron processors. Xen manages VT and AMD-V with an abstraction layer, and therefore can run unmodified host operating systems from Xen 3.0 onwards.

This paper will compare and contrast three different virtualization products; VMware, Xen and User Mode Linux. Since not every topic can be covered, they will be compared against each other relating to how the guest operating system interfaces the VMM and more importantly how the hypervisor does memory management.

User-Mode Linux

User-Mode Linux (UML) is an open source port of the Linux kernel to itself. It treats the host operating system as a platform to be ported onto, and therefore allows a version of UML inside another version of UML. There are no special patches needed on the host Linux; UML's kernel implements everything using system calls. UML runs without any instruction emulation as a user process. Processes running inside the UML have no access to the host resources or any other UMLs that may be running on the machine. All devices in the virtual machine are virtual themselves; the hardware cannot be addressed directly. UML provides virtual drives, virtual network adapters, virtual block devices, etc.

UML has two modes, the old mode called Tracing Thread (tt) mode, and a new mode developed in 2002 called the Separate Kernel Address Space (skas). In tt mode each UML process gets a process on the host operating system. A special thread called the tracing thread traces (with ptrace) all the system calls the UML processes do. The tracing thread gets notified when the system call is entered or exited and nullifies the original call. It then notifies the UML kernel to execute the system call. This had a number of problems such as securities and performance issues. The UML kernel exists in the address space of each of its processes and is writable by default; therefore a process could potentially escape to the host. UML has a jail mode which makes UML data read only when a process is running, but this degrades the performance. Another performance issue arises because signals need to be sent to force control to the UML kernel during every system call, of which the delivery and return are relatively slow.

Skas mode attempts to address these problems by changing where the UML kernel runs. The UML kernel now runs in a different host address space to its processes and so is wholly unattainable to them. This speeds up the performance due to the removal of signals for every system call and makes the system much more secure.

UML creates a file the size of UMLs physical memory when it is started up. Kernel and process virtual memory is done by mapping pages from this file to virtual address spaces. In tt mode, the kernel and processes share address space, so there are some unusable holes in the process virtual memory area. This system has some problems which lead to the UML using up a lot of the hosts memory. Like all Linux kernels, data is kept in cache until space is needed. This is perfectly normal for Linux running on hardware since nothing else would require use of memory. However, UML would be taking up memory that the host might possibly want to use. When UML does release some cache, this is still seen as dirty by the host, which will preserve it.

Another issue arises when there are multiple UMLs on a host machine. Unless the UMLs are booted from the same Copy on Write (COW) file, there will be duplication of data in the host's memory. For each UML (booted from similar images) the data will be present in the host's cache and the UMLs cache. UML does have some mechanisms to deal with these problems, such as booting multiple UMLs from the same COW file (only 1 copy in host cache) and setting `ubd=mmap,` which eliminates the multiple copies of UML cache. This still does not get rid of the memory consumption problem, which can be reduced by using `/dev/anon`. By applying a `devanon` patch to the host kernel and making sure the version of UML supports this; the cached pages will be freed to the host when it is unmapped in UML.

Xen

Xen is an open source virtual machine monitor where the host operating system can be Linux or BSD. Xen has very close to native performance results with little over 3% overhead. Prior to Xen 3.0, the host OS had to be patched since Xen did not offer full virtualization, however Xen 3.0 utilizes the hardware virtualization provided by Intel VT and AMD-V and does not require patching.

The x86 Intel architecture provides four levels of privilege which are called rings. Figure 2 shows an abstract example. The lowest is 0, which is what the operating system uses, giving unobstructed access to the hardware. The highest level, Ring 3, is used by applications. To give the illusion that the virtual OS is running in Ring 0, the VMM also has to run in Ring 0. Xen does not allow the guest operating systems to run in Ring 0 however, it runs them in Ring 1. This is called *ring deprivileging*. This gives rise to a lot of problems for the VMM, as some instructions will cause a general protection exception when run in ring 1. Prior to Xen 3.0, the way to solve this was for the guest OS's instruction to be replaced by a hypercall. A hypercall is similar to a Linux system call which passes control into Xen on ring 0. Xen does the instruction and emulates the results back to the guest OS. Similar to UMLs problem, this means Xen has to constantly monitor the guest OS and trap the instructions that will fail.

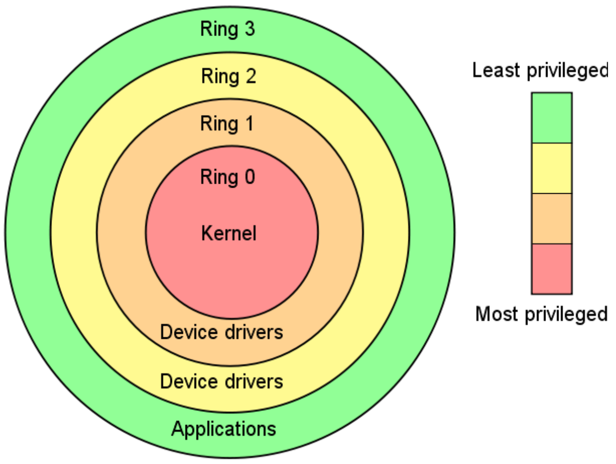

Figure 2: Privilege rings for the x86, along with their common uses[2]

Intel VT (and AMD-V) solve this problem for Xen by having two different classes of rings. In Intel VT there is now a privileged root ring and deprivileged non-root ring, called ring 0P and 0D respectively. Xen uses 0P, and the guest OS's use 0D. This means that the guest OS's are essentially running in ring 0, but are in the layer above Xen so are unaware they are virtualised.

Xen manages memory by abstraction. Because physical memory is used and freed up when it is needed, it is difficult for the guest OS's to have an unbroken stretch of memory for themselves. Most operating systems do not work well in fragmented memory address space, so Xen gives what appears to be a continuous address space to each guest OS. Xen has a machine-to-physical table which stores the mapping from actual machine page frames to "pseudo-physical" page frames, while each guest OS has the inverse of that table. To each guest OS, this looks like they have the whole of memory in one continuous block all starting from frame 0. In reality, each frame this is mapped to may be scattered about anywhere in machine memory in no particular order.

The guest OS may read their own page tables without intervention from Xen. However, anything other than a read but be explicitly requested. Once Xen has validated the request and considered it safe, it applies the request. This prevents the guest OS's to add any random mappings onto their tables and accessing wrong memory.

Xen can do memory management in a different way, where the guest OS does not need to request permission. This gives an illusion that the guest's page tables are writeable, but does not provide full virtualization of the Memory Management Unit (MMU). Xen traps any write attempt to a page table and disconnects that page from the table whilst it is being written. This means the MMU cannot access the new change until Xen has, as before, validated and safety checked it.

[2] Image from http://en.wikipedia.org/wiki/Ring_(computer_security)

VMware

VMware is a company that produces several different x86 VMMs. They are based in two categories; for desktops and servers. Even though VMware VMMs are proprietary, there are some small free versions (for example VMware Player) and some exist in open source format.

Because VMware is a full virtualization technique, the guest OS's do not communicate with the VMM. The guests do not know they are virtualized, so no special hypercalls are needed. VMware virtualises the guest OS "using a combination of binary translation and direct execution techniques."[3] Binary translation is the technique of replacing kernel instructions that are non-virtualizable with new instructions that emulate them. User level instructions are directly executed on the hosts processor to maximize performance.

Like Xen, VMware Server ESX (which shall be called VMware, for short) provides an abstraction for memory management. Each guest OS sees what it thinks is machine memory. This is done in a very similar fashion: VMware holds a PPN-to-MPN (physical page number to machine page number) table.

VMware does not separate each of the guest OS's memory like Xen. Instead, the guest OS's can share memory which enables a few extra features such as memory overcommitment. Pages can be shared across different guests on the same host or within the same guest. VMware does this by scanning the pages in use and makes a 64bit hash of them and adds them to a hash table. If the hash key matches another, then a full comparison of the pages is done. A single copy of this is then held in memory until one of the guest OS's wishes to write to it, in which case another copy is made for it to write to.

Overcommitment of memory means that the total size allocated for all the guests is more than the actual amount of machine memory. VMware manages the allocation of memory based on system load and parameters set. However, most operating systems do not support varying memory during run time, so the size of memory given to the guest stays constant after being started.

VMware has several techniques it uses to manage memory, such as 'ballooning'. This reclaims pages considered to be the least important to the guest OS. The guest OS requires a special balloon module, which can "inflate", forcing the guest OS to page out to virtual disk when memory is scarce and "deflate" freeing up some guest memory (see figure 3). When the balloon is inflated and the guest writes out to virtual disk, then VMware can regain the matching machine page. The balloon module polls VMware every second asking whether it needs to take any inflation/deflation action.

[3] http://www.vmware.com/files/pdf/VMware_paravirtualization.pdf, VMware Whitepaper: Understanding Full Virtualization, Paravirtualization and Hardware Assist

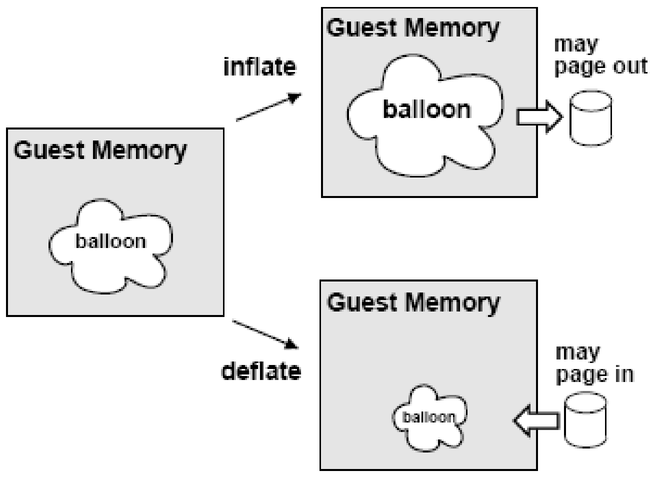

Figure 3: VMware ServerESX ballooning technique[4]

VMware also uses an 'idle memory tax' which fines a guest OS for idle pages. The idea is to stop virtual machines hoarding memory for idle tasks and give some of it to guests who are struggling with little memory. This works in conjunction with ballooning, as the machines that have more idle pages will be more likely to have inflated balloons. The default tax rate is 75%, which admits most idle memory to be freed up but also allows for some robustness such as if that guest soon after has a lot of active pages.

Conclusions

In both memory management and running modes, the default settings do not make UML very efficient to run on the host machine. Add-ons and patches are needed to free up memory and cache, and reduce system calls. Xen and VMware both run much more sophisticated memory management techniques, but are also much more sophisticated virtual machines.

Xen's memory management is very efficient as the guests do not share any virtual memory. There is a slight penalty for allowing memory sharing, as the VMM has to constantly scan through all the pages to check for duplication. The advantages of memory sharing are dependent on the system: if the guest OS's are different and do not use much memory, there will be hardly any sharing. However, it is conceivable that sharing is effective when multiple copies of the same operating system with the same setup are virtualised.

VMware overcommits memory and then uses ballooning and idle memory tax to make sure no problems arise because of this. Overcommitment is a good idea since the guest OS's can have more RAM than is available, but it also means VMware has to work hard at making sure that the host and guest OS's do not

[4] Image from http://www.VMware.com/pdf/usenix_resource_mgmt.pdf

run out memory. Whether this trade-off is better than Xen's performance would have to be tested, but Xen's simple partitioning of memory and memory management may be just as efficient as VMware's more complicated approach. Also, as today's servers have a lot of memory, VMware's memory management techniques may not give much more of a performance increase.

Choosing which VM technology to use would be difficult for high performance applications, such as a hosting service, as it depends on the type and complexity of service being provided. If the system was large and complex, then VMware or Xen would be suitable choices. If the server(s) were older machines, VMware would have an advantage, as Xen could not make use of hardware virtualization technology and VMware would be able to cope with small amounts of RAM. However, in a newer machine with lots of memory and Intel VT/AMD-V, Xen and VMware's performance would probably be matched. UML would be a great choice if the hosting service was not too demanding and efficiency was not an issue.

There are many other issues to consider as well, such as if Windows as hosting operating system was needed (VMware is the only one which supports this) or if the VMM had to be open source and free. Overall if a choice had to be made, VMware would probably come out best.

References & Resources

People

Jeff Dike, creator of User Mode Linux

Discussion on UML on IRC channel with various UML developers

Online Articles and Papers

Formal requirements for virtualizable third generation architectures, Gerald J. Popek and Robert P. Goldberg http://portal.acm.org/citation.cfm?doid=361011.361073

Introduction to the Xen virtual machine, Rami Rosen http://www.linuxjournal.com/article/8540

Virtualization in Xen3.0, Rami Rosen http://www.linuxjournal.com/article/8909

User-Mode Linux, Jeff Dike http://www.kernel.org/doc/ols/2001/uml.pdf

User-mode Linux. Jeff Dike http://www.redhat.com/magazine/001nov04/features/usermode/#uml-design

Intel ® Virtualization Technology: A Primer, Andrew Binstock http://www.intel.com/cd/ids/developer/asmo-na/eng/221874.htm?page=4

Interface manual Xen v3.0 for x86, The Xen Team University of Cambridge, UK http://www.linuxtopia.org/online_books/linux_virtualization/xen_3.0_interface_guide/index.html

Server Virtualization Memory Sharing - Vendors Divided, Chris Wolf http://dcsblog.burtongroup.com/data_center_strategies/2007/06/virtualization_.html

Memory Resource Management in VMware ESX Server, Carl A. Waldspurger http://www.VMware.com/pdf/usenix_resource_mgmt.pdf

The Role of Memory in VMware ESX Server 3, Author unknown http://www.VMware.com/pdf/esx3_memory.pdf

Understanding Full Virtualization, Paravirtualization and Hardware Assist, Author unknown http://www.vmware.com/files/pdf/VMware_paravirtualization.pdf