Hardware Virtualization In Nested Virtualization

Muhammad Imran Iqbal
Department of Computer Science, University of Lahore, Sargodha Campus, Pakistan
&
Sobia Mehdi
Department of Computer Science, University of Lahore, Sargodha Campus, Pakistan
&
Iqra Waheed
Department of Computer Science, University of Lahore, Sargodha Campus, Pakistan

Abstract— it is a software implementation that runs on computers to run different programs like a real physical machine. There are two classifications of Virtual Machines categorized by their use i.e. System Virtual Machine and Process Virtual Machine. Virtual Machine Monitor or Hypervisor is the software layer that provides virtualization. The operating system running on virtual machine is called guest operating system.

From Operating System designs and development, Virtualization has great impact and importance. Different techniques are used in virtualization; each has its own implementations under certain circumstances. One technique is not truly efficient for other environment. Shared kernel, full kernel and hypervisor virtualization are different techniques that be used combined as well as independently. Nowadays the full virtualization with binary translation is the most effective and reliable technology used ever with great ease of use. VMware uses both full virtualization with binary translation and hardware assisted virtualization. We have discussed many types of virtualization techniques and their interrupt handling. How virtual machine works over host operating system is well explained in this review. In nested virtualization there is a lack of architecture support for multi-level virtualization. For this reason it has up to 10% of overload. It is possible to overcome the overhead i.e. 6-10% for nested virtualization and to find a better solution.

I. BACKGROUND LITERATURE REVIEW

A. Virtual Machine

Virtual Machine is a software implementation that runs on computers to run different programs like a real physical machine. There are two classifications of Virtual Machines categorized by their uses.

1.- System Virtual Machine
2.- Process Virtual Machine
3.-

If a virtual machine provides a complete set of system running complete Operating system by hardware virtualization, is called system VM [1][3][17]. It’s also called Hardware VM. It shares physical resources by different VMs, each VM have its own operation system. Virtual Machine Monitor or Hypervisor is the software layer that provides virtualization. The operating system running on virtual machine is called guest operating system.

Another type of VM that is designed to execute a single process or a program is called Process VM. The purpose of this type of VM is to run programing languages limited to the resources and abstractions provided. Like if we installed windows on our system and then install virtual process, after this install another windows then second windows will act like guest and first will act like host. The host will handle the interrupts of guest. [3] [17]

Virtualization is a layer that is added between physical hardware and the host operating system to provide services to the guest operating systems. There are two architectures defined till now i.e. Hosted virtualization and hypervisor. Hosted architecture works like an application that runs on any OS where hypervisor installs a layer that runs on bare-metal hardware. [3]
B. Virtualization and its techniques
Every operating system generally relies only on interrupts generated whenever it wants to respond to internal event or any external event. So for better response between the interrupts generated by guest OS must be optimally handled by the host OS in VMs. Before implementing any virtualization it is deadly important to understand what virtualization is and what techniques are used to implement different virtualization solutions [2]
There are various techniques used are.
✓ Guest Operating System Virtualization
✓ Shared Kernel Virtualization
✓ Kernel Level Virtualization
✓ Hypervisor Virtualization
• Para-virtualization
• Full Virtualization
• Hardware Virtualization
C. Guest OS Virtualization
In Gust OS virtualization the host machine has a single OS that may be windows OS, Linux or any other. A virtualization application is simply installed that has the responsibility to start stop or respond to each virtual machine created in that virtualization application.
D. Shared Kernel Virtualization
In shared Kernel Virtualization, it takes the advantage of the architecture of Linux, UNIX or any other open sources OS. It works by sharing the kernel of guest operating system. Each guest OS uses its own root file. Just link Linux/UNIX. Kernel can change the root file by using “chroot”. [2] Guest OS must be compatible with the kernel being shared otherwise this method is not going to work effectively.

E. Shared Kernel Virtualization
With the drawbacks of shared kernel, a new modified kernel is used with extensions to manage multiple VM, each having 1 guest OS. Each guest OS has its own kernel. [2] Each kernel must also be compiled with the host for same hardware.

II. Three level Virtualization
A. Hypervisor Virtualization or CPU Virtualization.
X.86 family processors have protected rings where instructions are executed. Ring 0 has the highest priority that can execute kernel level instructions. All other application instructions run in ring 3. In Hypervisor Virtualization, an application called VMM/Hypervisor runs directly in ring 0 providing memory or hardware allocations or other monitoring. [2][3]
If hypervisor is running in ring 0 then there will be problem for OS kernel. Host OS is explicitly designed to run instructions like memory allocation etc. In ring 0. So for that reason Guest OS must work in less privileged ring. To address this problem there are few techniques that may be applied. This problem was solved in 1998 by introducing a new technique that is called Binary Translation.

a. Para-virtualization (OS assisted Virtualization)
In Para-virtualization the guest operating system is modified to run with hypervisor. It replaces privileged instructions with hyper calls. It does not support un-modified kernels [2][3].

b. Full Virtualization using Binary Translation
Full virtualization supports unmodified kernel. The hypervisor protects the CPU operations and provides an emulation to run both in ring 0. Using binary translation in which instructions from guest OS is translated into new sequence of instructions that directly runs on Hardware. This technique is costly for time and resources. [2][3]

c. Hardware virtualization
Intel and AMD processors introduced virtualization in their hardware so that the virtualization runs without having overhead like full virtualization. They include an additional mode above ring 0 for virtualization. An administrative console runs over hypervisor to manage resources.[3]
B. MEMORY VIRTUALIZATION
After CPU virtualization the next step is memory virtualization that shares physical resources among all virtual machines. The operating system keeps mappings of virtual page numbers to physical page numbers stored in page tables. All modern x86 CPUs include a memory management unit (MMU) and a translation look aside buffer (TLB) to optimize virtual memory performance. The VMM is responsible for mapping all guest memories to physical addresses. [3]

C. Device and I/O Virtualization
The last one is Device and I/O Virtualization. Software based Device and I/O Virtualization is more featured as compared to the direct hardware. By using virtual NIC and switches user can enjoy the features of a virtual network.[3]

III. NESTED VIRTUALIZATION
A hypervisor can run multiple operating systems concurrently, but now we want to make virtualization more capable by utilizing the concept of nested virtualization, in which a hypervisor can run multiple other hypervisors with their associated virtual machines.

The authors of [4] represented the design, implementation, analysis and evaluation of nested virtualization. Although there is lack of support or nested virtualization, author uses multiple level of virtualization on single level of architectural support by CPU (e.g., instruction-set) virtualization, memory (MMU) virtualization, and I/O virtualization. The base of x.86 architecture is based on trap and emulate model [6][7]. Whenever a trap is occurred in guest hypervisor it results a trap in lower level.

The implementation of nested virtualization is based on Intel x.86 architecture. There was three level of virtualization in turtle’s project [4].

A. CPU: Nested VMX Virtualization
Lower hypervisor inspects the trap and then forwards to upper level for emulation. For this purpose there were some optimizations were made for efficiency [6] [7]. There is also special in-memory virtual machine control structures (VMCS) [11] added that has three groups. VMEntry is used to enter in to guest mode and VMExit is used to switch back to hypervisor and Control Data: used by hypervisor in case an interrupt occur into virtual machine and specify where it exit from the virtual mode to root mode.

B. MMU: Multi-Dimensional Paging
Multi-dimensional paging is used for efficient memory virtualization that combines different memory tables into one or two tables provided by MMU [8]. Address translation is required to translate virtual addresses into physical address and vice versa, for this purpose we need a thin layer of address translation by software means or hardware assisted. Current hardware supports only two level of translation so new technique Multi-Dimensional Paging is required for multiplexing the three needed translation tables onto the two available in hardware. For this reason guest creates a guest page table (GPT), hypervisor use this GPT to translate the guest virtual addresses into the host physical addresses.
that called **shadow page table (SPT)**. Then hypervisor then run the SPT instead of GPT. For good performance x 86 recently added two-dimensional pages **table** in which we firstly translate the guest virtual address into virtual physical addresses that is called EPT, and then use this EPT to convert virtual physical addresses to host physical addresses.

![Figure 2 MMU alternatives for nested virtualization [4]](image)

Hypervisor decide it on the processor capabilities whether to use the SPT or EPT to virtualize the MMU. [4]

C. **I/O: Multi-Level Device Assignment**

Direct device assignment is adopted by bypassing multiple levels hypervisor I/O stack to provide nested guests with direct device assignment. [9] [10] There are three methods i.e. emulation, Para-virtualized driver and direct assignment [12] [13]. Among all of these three methods, direct device assignment [9] [10] is considered for providing best performance [14] [15] because we don’t need to switch between hypervisor and VMs. By multi-level device assignment L2 guest can directly access the underlying hardware by dealing with DMA, interrupts, MMIO (Memory-mapped I/O) and PIOs (Port I/O) [16]. For emulation and injection method it slows down the performance. For modified guests, para-virtualized IOMMU is used which maps the mapping of L2 to L1 is replaced by a hyper call of L0. L0 changes the mapping table resulting mapping from L2 to L0. The slowdown factor can be optimized by optimized exit handling at each level.

IV. **VIC: INTERRUPT COALESAGING FOR VIRTUAL MACHINE STORAGE DEVICE IO**

CPU overheads for handling all the interrupts can be very high for high IO rate and it can cause lack of CPU resources for application itself. In existing techniques, CPU overhead is a big problem in virtualization. [19, 20] Existing techniques are not practical for virtual devices because they are high resolution timer based.

As compared to other existing techniques, [21] presents a technique which is not based on high resolution interrupt delay timers and it provides an effective implementation in a hypervisor by increasing a work throughput and CPU overheads. This is a design and implementation of a virtual interrupt coalescing (vIC) scheme for virtual SCSI hardware controllers in a hypervisor.

**A. Challenges for vIC:**

1. How to control the rate of interrupt delivery from a VMM to a guest without loss of throughput?
2. How and when to delay the IPIs without inducing high IO latencies?

In the proposed technique, the authors of [21] present their virtual interrupt coalescing mechanisms to efficiently resolve both of these challenges.

**B. Implementation**

In this design, a parameter is used known as interrupt delivery ratio R, it represent the ratio of interrupt delivered to the guest and the actual number of interrupts received from device for that guest. Is the ratio is small it indicates higher degree of coalescing. Dynamically, interrupt delivery ratio R, is set up in a way it provide coalescing benefits to CPU efficiency. We took command in flight (CIF) as main parameter and IOPS (IO per second) as secondary.
Three main parameters used in this technique are:

1. \textbf{iopsThreshold},
2. \textbf{cifThreshold} and
3. \textbf{epochPeriod}

In iopsThreshold, the specific IOPS value below which no interrupts coalescing can be done.
cifThreshold: the specific CIF value below which no interrupts coalescing can be done.
epochPeriod: a specific time after which we reevaluate the ratio.

The algorithm operates on following conditions:

1. **Low-IOPS (R = 1):** if achieved throughput of a workload is below the configurable iopsThreshold then vIC will turn off.
2. **Low-CIF (R = 1):** if achieved CIF is below a configurable cifThreshold then VLC will turn off.
3. **Variable R based on CIF:** ratio R is challenging because CPU efficiency achieved by coalescing may be low down the throughput.

The additions of steps are: vIC adds a new “shared area object tracking” the last time that the virtual machine monitor interrupt.

![Figure 3 Virtual Interrupt Delivery Steps [21]](image)

In addition to System model of ESX server architecture, vIC adds a new shared area object tracking the last time VMM interrupt. In VMM, interrupt coalescing scheme is introduced before sending the IPI, vSCSI ensure the time last VMM interrupt is less than a configurable threshold. If not then IPI is fired. If yes then IPI is deferred.

**V. RECURSIVE VIRTUALIZATION**

Recursive virtualization gaining popularity nowadays. Many projects support some form of recursive virtualization. Modern operating system start it to embed the visualization functionality in it and also making hypervisors more capable to run at the same time on one host machine because it save significant hardware cost. One new concept of hardware prototype introduce that consider it could be work efficient like real time system and In real time system resources are in recursive by nature because of local decision[18], in event handling we could use the same process for recursive virtualization. Two algorithms are designed in recursive virtualizations that are discussed below.

**A. Forward Propagation:**

When an event is received by a hypervisor, it may be consume by itself or forward this event to its actual guest in the hierarchy. It’s known as forward propagation. According to the figure 1, and suppose that key board is assigned to App1 while the processor is assigned to App2. Now, external keyboard interrupts should be delivered to App1 while at the same external timer interrupts should be forwarded to OS2.
B. Reverse Propagation

Internal events should be delivered in top-down manners. Without the hardware support this event delivered in bottommost hypervisor first so we simulate it in top-down manners. In figure 2, exception generated in App1 should be delivered to OS1 and same as with the OS3 to deliver the exception to hypervisor 3 and after exception handling hypervisor forward exception to its correct level. It’s important that each hypervisor make decision on its base not on others so we need an algorithm for correct and efficient top-down delivery. We call it reverse propagation.[17]

VI. CONCLUSION AND FUTURE WORK

Nowadays the full virtualization with binary translation is the most effective and reliable technology used ever with great ease of use. VMware uses both full virtualization with binary translation and hardware assisted virtualization. We have discussed much type of virtual machines and their interrupt handling. How virtual machine works over host operating system is well explained in this review.

In nested virtualization there were few methods to be used for efficiency. First of all we learnt that there is architectural support problem. x-86 architecture does not support multi-level virtualization, that's nested virtualization was introduced. In nested three area are virtualized namely CPU, Memory and I/O but they cost in overhead. There is a lack of architecture support for multi-level virtualization. For this reason it has up to 10% of overload. It is possible to overcome the overhead i.e. 6-10% for nested virtualization by introducing multi-level architecture support because X.86 architecture supports only level 1 hypervisor. Same hardware support problem with recursive virtualization. Second thing today’s modern operating systems are embedding virtualization support into its kernel that has great performance factor. If multiple level hypervisor, as discussed in recursive and nested virtualization, is introduced in operating systems instead of separate hypervisor for each virtual machine. Windows 7 has introduced hypervisor in it for win XP virtual machine, performance should boost up and the overall overhead of X.86 architecture can be reduced. This is implementation of virtual interrupt coalescing (vIC) in the VMware ESX hypervisor showed it is able to improve throughput (IOPS) up to 19% and improve CPU efficiency up to 17%.

Architecture modifications are time taking and a long process so experts suggest software modifications instead of hardware architecture level modifications. And also there is a big scope in Multi-level paging and device assignment.
VII. ACKNOWLEDGMENT
The authors would like to thank our Supervisor Miss Nida Yasir for her continuous support and guideline throughout the research. We would also like to thank Mr. Afzaal Ahmad for his innovative ideas and insightful comments and discussion.

VIII. REFERENCES