Multi-tier Virtual Machine Architecture: VM-on-VM (n, n-1) Implementation

Prakhar Agarwal and Vikas Saxena

Jaypee Institute of Information Technology University, Noida
E-mail : prakhar.jiit@gmail.com, vikas.saxena@jiit.ac.in

ABSTRACT

The maintenance of Virtual Machine (VM) and inter-VM communication is still a challenge in virtual machines development. This paper proposes a scheme called "VM-on-VM", which creates multiple layers of VMs hosting the operating systems (OS) and allows them to share conventional hardware in an innovative fashion. The concept of reverse-hierarchy has been devised to demonstrate the proposed scheme.

1. INTRODUCTION

Virtual Machine is a software implementation of a machine that executes programs like a real machine [1]. [2]It's an efficient, isolated duplicate of real machine with no direct correspondence to the real hardware [3].

1.1 Classification of VMs

Virtual Machine derives its original motivation from the desire of using multiple OSs simultaneously on a single machine. VM also works as an Emulator. Virtual machines are classified in two major categories based on their use and degree of correspondence to any real machine [1]:

a. A system virtual machine **(**SVM) provides support for execution of complete operating system (OS)

b. A process virtual machine (PVM) that provides support for execution of a single process

This paper presents a scheme for SVM.

1.2 Virtualization

[4]Successful partitioning of a machine to support concurrent execution of multiple operating systems poses several challenges. Firstly, one virtual machine should not adversely affect other's performance. Secondly, support to a variety of operating systems. Thirdly, performance overhead due to virtualization should be small.

Multiplexing hardware through various layers of virtual machines plays a major role in implementing the interaction between the operating systems and real hardware.

2. PRELIMINARIES

In traditional virtual machine architecture each VM emulates the underlying bare hardware into an independent virtual copy of main computer. Each operating system has its own dedicated VM.

The contemporary approach is referred to as native execution or full virtualization and has been implemented by the use of Virtual Machine Monitor (VMM), also called Hypervisor- Type I and Type II [6].

Type-I Hypervisor runs directly on the bare hardware. Type-II Hypervisor is the software that runs on OS environment [6]. Figure.1 exhibits the current architecture using Type-I hypervisor.

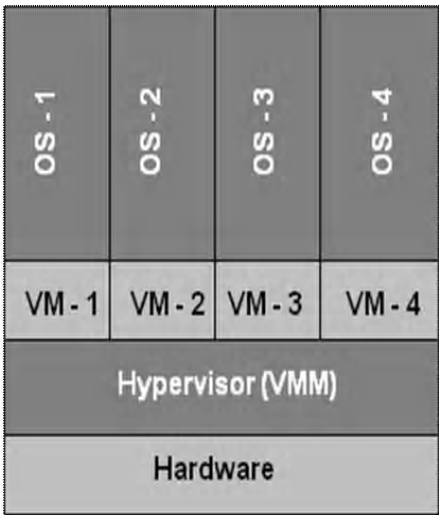

Fig. 1: Current VM Architecture

[1] Multiple VMs running private operating system are used in server consolidation. Instead of running the different processes on different machines, Virtual Machine Architecture provides a platform for running different services on different VMs. This feature has been termed as Quality-of-Service Isolation (QoS isolation) [1].

2.1 Limitations

Although the contemporary designs considerably reduce the hardware requirements. But after a strong scrutiny, following imperfections have surfaced which need to be addressed:

(1) Inter-VM communication: The QoS Isolation aims at consolidating the services to one machine. This is achieved by establishing communication among various VMs to operate as one unit. Thus a term called inter-VM communication comes under play. The benchmark used to evaluate this communication is based on Netperf [7], [8]. Each VM runs a Netperf server and Netperf client. These VMs are then connected in a ring topology [8]. We have successfully eradicated the use of such external requirement in our proposed scheme.

(2) Error Reporting mechanism: The present models of VMs fail to bring out the very crucial component of OS development, i.e. error handling. This important element has been silently ignored in all the designs and has left many loose threads in virtualization development. Our scheme addresses this issue also.

(3) Lack of Multi-purpose of Hypervisor: Hypervisors used in the virtualization of the machine are limited to work on either bare hardware (Type-I) or on an OS (Type-II). Thus we have to make an extra effort to identify which hypervisor is to be employed during the implementation of any virtualization model. This increases the requirement of manpower and related resources.

3. PROPOSED ARCHITECTURE

The limitations highlighted above motivate us to further enhance the virtualization techniques. Some critical drawbacks and their possible solutions are discussed in this section.

3.1 Multi-tier VM Interface

Starting from the current designs, we propose to design a hierarchy of virtual machines through multi-tier model. To achieve this, first of all we will rectify the demerit of hypervisor discussed above. We propose to build a Hybrid Virtual Machine Monitor (HVMM) which can virtually run on Hardware, OS and even on VM. A more simple term to define such a component would be: a multi-purpose VMM. It can eliminate the requirement of different types of hypervisor and help in reducing the inter-operatibility issues.

3.2 Reverse Hierarchy Model

In our model, bottom-most layer is occupied by real hardware. Just above the hardware layer we have placed a Hybrid Virtual Machine Monitor (HVMM-1). Above that, the base Virtual Machine of our model (VM-1) is present. Then presiding over VM-1 is another HVMM-2 using which VM-1 hosts another VM-2 and an operating system OS-1. Further, VM-2 and OS-1 have HVMM-3 and HVMM-4 over them respectively. The first virtual machine, VM-1, will

not be capable of hosting any operating system as it is the base machine. At any stage of this hierarchy we'll find that total number of OSs is one less than total number of VMs. Thus, the scheme gets its name as (n, n-1) implementation, where,

n = number of VMs at a specific layer

n-1 = number of OSs at the corresponding layer.

Figure 2 will give a clear understanding of the concept.

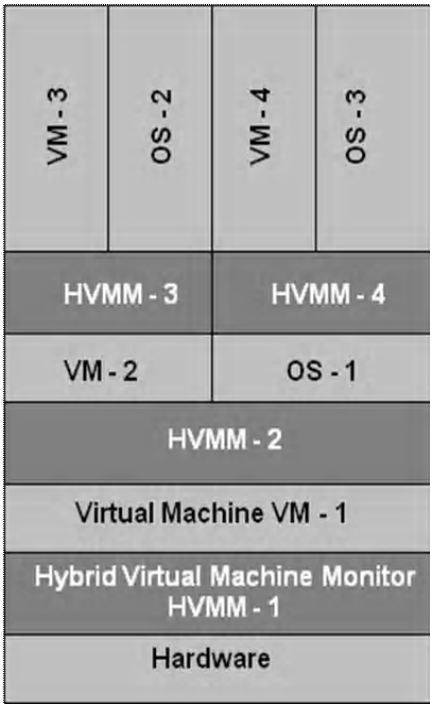

Fig. 2: Multi-tier VM architecture

As it is clear from the above figure, we have proposed an idea of hosting a VM on a VM. HVMM plays an integral role in successfully supporting this scheme. All major virtualization tools like VMware[4] and XEN[9] do not support this feature. The present designs exhibit compulsory requirements of different types of hypervisors and therefore are less flexible. The major disadvantage of these contemporary models is that if an OS doesn't require all the resources provided by its host VM then these resources are wasted unknowingly. This wastage takes place at every OS-VM communication channel. The hard fact is that we can not share these resources with any other VM. So, to prevent this situation and in return maximize the use of any VM, we are exploiting all such free resources to host another VM. The whole process continues in an upward direction or we can say it represents a reverse-hierarchy. According to the need of the OS, it is placed appropriately in this hierarchy. The OS with minimum requirements sits at the top-most level and vice-versa providing a very streamlined process for resource utilization.

3.3 Inter-VM communication

As discussed previously, Netperf is the external requirement to establish interaction among all the VMs on a single hardware. The use of Hybrid Virtual Machine Monitor eliminates this flaw from the contemporary designs. An added feature of HVMM is that it enables communication between its *Host* and its *Guest*. When we are hosting a VM-on-VM, the communicating interface between them is HVMM. There is no need of any other element. The interaction takes place in the form "packets". These "packets" at each layer take care of the status of each operating system.

Whenever, new OS is loaded a performance check is done and the whole tree is optimized for best performance. All messages and signals are sent via this method only. This is what we refer to as tree topology. This unique design enhances the use of virtual machines. Using virtual machines in combination increases the productivity of each virtual machine. This approach scores over previous implementations by providing *Direct* sharing of resources among the various operating systems and VMs.

Numerous systems have been designed which use virtualization to subdivide the ample resources of a modern computer. Some require special hardware while some fail to support a range of operating systems. Our model exhibits the super functionality of the hypervisor to perform the inter-layer communication. There is no requirement of any other scheme to achieve this task. Secondly, the networking in the whole system is present *by default* through virtual communications network.

This model also minimizes the maintenance effort to an enormous extent. The present models demand maintenance of each and every virtual machine if ever a real hardware failure occurs. Each VM will need to be re-configured with accordance to the new hardware to achieve the "*best stable condition*". This unwanted process has been very easily removed in our model. The base virtual machine, VM-1 or the hypervisor is the only VM directly interacting with the real hardware. All other virtual machines are interacting with this base VM with connections traversing through the hierarchy. So, if ever there is any kind of hardware failure, then only VM-1 specifications need to be modified. This saves a lot of possible computations and time.

3.4 Error- Reporting Mechanism

Lastly, every virtual machine in this model maintains an Error Log Retention File (ERL File) which keeps a record of all kind of associated errors and keeps the record in the log until proper remedial action is taken. The main VM, that is, VM-1 maintains a Master Failure Log (MFL) which records all the notifications received from child VMs. Thus, if the MFL is empty, that means all ERL Files are also empty. Logically, the binary digit '0' represents zero errors while '1' indicates one or more errors. Figure.3 explains this concept.

Failure of any of the operating systems in the complete hierarchy is notified in an easy and systematic manner. The VM hosting this failed operating system records the error in its ERL File. This ERL File then sends this information to the parent VM which further notifies its super-parent. This process is executed in the entire structure with an entry being made at each level until it reaches the MFL. In simple terms, MFL is dependent on ERL Files present at each and every layer of virtual machines. When an error is resolved the status bit of that error will turn '0' which will be replicated in corresponding and root ERL as well as MFL.

All these features equip our model to be in a healthy and protected state round the clock, thus promoting system development and maintenance on

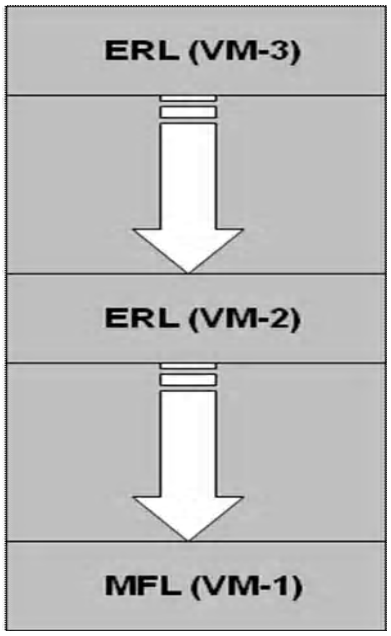

Fig. 3: Error Reporting Diagram

the run. This model decreases the amount of human labor required in keeping the system up to date. Every kind of system maintenance task is performed by the system itself and hence this a better approach as compared the existing models.

4. FUTURE WORK

There is a great scope in this area of computing. One of the few future developments can be implementing multiple Base Virtual Machines (VM-1) on a single hardware. We could then have multiple virtual machines trees residing on a single hardware. This will take the concept of virtual machines to next level where VMs will no more be a choice but rather a necessity.

5. CONCLUSION

We have proposed a methodology of implementing Multi-tier reverse hierarchy of virtual machines. The model presented here shows how a simple concept of trees is exploited in building the next generation virtual machines. Our results prove that existing models have many shortcomings which have been very carefully eradicated in our model. This kind of future of virtual

machines provides an excellent area of study for great advancement.

REFERENCES

1. http://en.wikipedia.org/wiki/Virtual_machine
2. Gerald J. Popek and Robert P. Goldberg (1974). "Formal Requirements for Virtualizable Third Generation Architectures". *Communications of the ACM* 17 (7): 412 –421.
3. Smith, Daniel E.; Nair, Ravi. "The Architecture of Virtual Machines". *Computer* 38 (5): 32–38. IEEE Computer Society. doi:10.1109/MC.2005.173.
4. The Book Of VMware - The Complete Guide To VMware Workstation (2002)
5. Operating System Concepts, Sixth Edition, Abraham Silberschatz, Peter Baer Galvin, Greg Gagne
6. http://en.wikipedia.org/wiki/Hypervisor
7. Benjamin Qu´etier, Vincent Neri, Franck Cappello. Selecting a Virtualization System For Grid/P2P Large Scale Emulation.
8. http://www.netperf.org/netperf/
9. Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177. ACM Press, 2003
10. Open Virtual Machine Format Specification (OVF) v0.9- White Paper
11. Running Multiple Operating systems concurrently on an IA32 PC using virtualization techniques, Kevin Lawton, 1999
12. Amit Singh. An introduction to virtualization. In *http://www.kernelthread.com/publications/virtualization/*, March 2004
13. S. Devine, E. Bugnion, and M. Rosenblum. Virtualization system including a virtual machine monitor for a computer with a segmented architecture. *US Patent*, 6397242, Oct. 1998.
14. K. Govil, D. Teodosiu, Y. Huang, and M. Rosenblum. Cellular Disco: Resource management using virtual clusters on shared-memory multiprocessors. In *Proceedings of the 17th ACM SIGOPS Symposium on Operating Systems Principles*, volume 33(5) of *ACM Operating Systems Review*, pages 154–169, Dec. 1999
15. J. Bruno, J. Brustoloni, E. Gabber, B. Ozden, and A. Silberschatz. Retrofitting quality of service into a time-sharing operating system. In *Proceedings of the USENIX 1999 Annual Technical Conference*, June, 1999.