Exploit of Open Source Hypervisors for Managing the Virtual Machines on Cloud

Sarfraz Nawaz Brohi
Advanced Informatics School
Universiti Teknologi Malaysia
Kuala Lumpur, Malaysia
sarfraz_brohi@hotmail.com

Mervat Adib Bamiah
Advanced Informatics School
Universiti Teknologi Malaysia
Kuala Lumpur, Malaysia
mervatbamiah@yahoo.com

Abstract— **If you need milk, would you buy a cow? Answer will be definitely no, so why to buy software, hardware or storage when you just require service. This innovative and revolutionizing mode of acquiring and utilizing IT resources is offered by cloud computing where IT resources are provided as on-demand services whether it's a software, hardware or storage capacity. In order to reduce the global warming, cloud computing is moving towards virtualization, under this technique, memory, CPU and computational power is provided to clients' virtual machines (VMs) virtually based on reality of the physical hardware. In a virtualized cloud environment, each client has a VM that is running client specific applications. As the operating system (OS) of cloud provider is running multiple VMs concurrently, it's a challenging task to manage the entire VMs. Virtual Machine Manager (VMM) so called hypervisor is the tool used by cloud providers to manage the VMs in order to eliminate downtime and to provide efficient storage, CPU as well as computational power to each VM. There are several hypervisors available in industry such as VMware, Hyper-V, Xen and Kernel Virtual Machine (KVM). In order to contribute in the field of cloud virtualization, this research paper represents the virtualization of hypervisor on x86 architecture and conducts comparison analysis on two open source hypervisors i.e. Xen and KVM to clarify the suitability of these hypervisor for managing the VMs on cloud.**

Keywords: Virtualization, Hypervisor, Xen, KVM

I. INTRODUCTION

x86 architecture is proven to be an influential platform for virtualization in enterprise computing due to its powerful features such as large scale multithreading with eight or more processing cores, high speed CPU and chipset for advanced reliability, availability and serviceability (RAS) [2]. In an operating environment, OS acts as a middleware between the users and hardware.

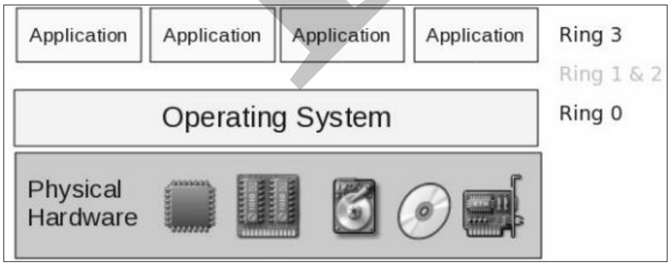

Figure 1. The x86 architecture.

The users' requests are managed by OS, where they are accessing the OS through application programs that have limited privileges to access the physical hardware [1]. In order to provide a secure operating environment, the architecture of x86 is divided into rings as shown in Fig.1 [2]. Each ring defines a privilege access level. The users' applications are isolated from the OS. These applications are placed at ring-3, that is lower privileged layer and OS is placed at Ring-0 that is higher privilege layer. Ring 1 and 2 are not used yet. The level of privilege represents the access level to the hardware, so OS have full privileges to access the underlying hardware whereas applications cannot execute a system call or instruction that is reserved by OS [2]. In a virtualized cloud computing environment several VMs running with their OSs are using the same underlying hardware concurrently as shown in Fig.2 [2]. The VMs need to access the hardware with the help of a middleware. In this case, instead of OS there is need to use hypervisor as middleware between VMs and hardware. A hypervisor manages the communication link between VMs and the physical hardware.

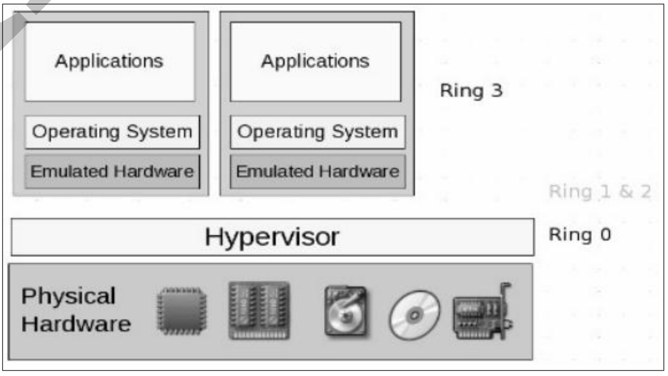

Figure 2. The x86 virtualized architecture.

The VMs are placed at ring-3 in x86 virtualized architecture. When the OS of a VM is running at ring-3, it can issue a system call that is not allowed for a ring-3 application. If the underlying hardware detects the system call it will abolish the VM [2]. In order to handle these kind of issues and manage the VMs efficiently such as providing dynamic virtual memory and CPU scheduling, several vendors has provided solutions by introducing their own hypervisors that are being used together with existing OSs. The most popular hypervisors used for cloud computing are Xen, VMware, Hyper-V and KVM. Since Xen and KVM are open source hypervisors, these are the most demandable and competitive hypervisors being used on cloud. In this research paper we will compare these both hypervisors to determine most efficient and powerful hypervisor that fulfils the need of a cloud computing platform but before we conduct the comparison analysis, we need to discuss the use of Xen and KVM in enterprise cloud computing.

II. THE USE OF XEN AND KVM HYPERVISORS IN CLOUD COMPUTING

A. Xen Hypervisor

Xen hypervisor is a distinctive open source technology, developed collaboratively by the Xen community and engineers from more than fifty innovative datacenter solution vendors, including AMD, Citrix, Dell, Fujitsu, HP, IBM, Intel, NEC, Novell, Red Hat, Samsung, SGI, Sun, Unisys, and many other industry leaders. It is licensed under General Public License (GPL). It resides between the VMs and hardware as shown in Fig.3 [3]. Mainly there are two types of hypervisor i.e. Type 1 and 2. Xen is a Type-1 hypervisor. A Type-1 hypervisor runs directly up on the hardware with a separated layer from the host OS. Type-2 hypervisor runs together with the host OS [4].

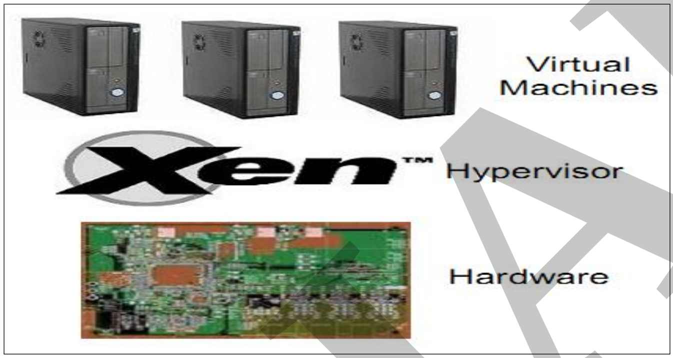

Figure 3. Xen Hypervisor.

Due to the isolation from the host OS, the security, performance and scalability features in Type-1 are more enhanced than Type-2. In a virtualized Xen cloud computing model, each client has its own VM that is running client applications. In order to create a secure operating environment, Xen hypervisor divides the VMs into two domains i.e. Domain0 (Dom0) and DomainU (DomU) due to the accessibility privileges. The Dom0 VMs have the higher privileges and they can access the hardware whereas DomU VMs have lower privileges and cannot directly access the hardware as shown in Fig.4 [5]. When the Xen hypervisor is started, for the first time it loads the Dom0 VM. Normally the user of Dom0 is a system administrator who has privilege to use the interface of hypervisor to create, delete or manage any DomU VM. Each DomU VM contains a modified Linux kernel that includes front end drivers and instead of communicating directly with hardware, it communicates with the Xen hypervisor [5].

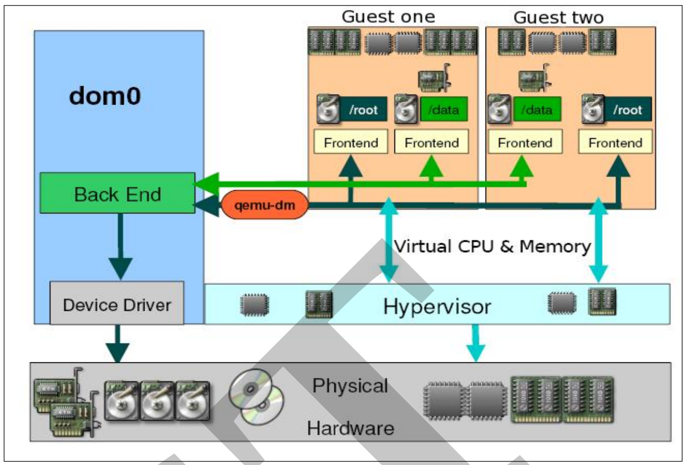

Figure 4. Xen architecture.

For each DomU VM, CPU and memory access operations are handled directly by the Xen hypervisor but I/O is directed to Dom0 because Xen hypervisor is not able to perform any I/O operation. There is a communication channel between Dom0 and DomU, through which DomU VM send I/O requests to Dom0 by using front end drivers as shown in Fig.5 [6].

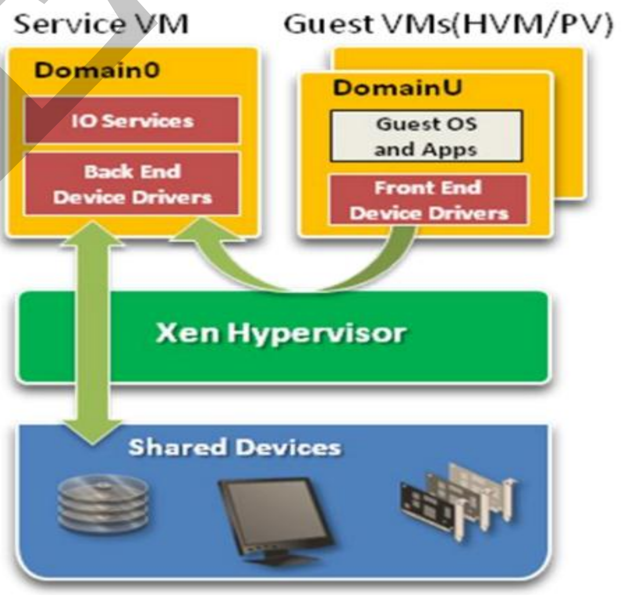

Figure 5. Xen service model.

The above stated model is called Xen service model. There are few limitations identified in this model. For-example when number of DomU VMs increase and each VM request Dom0 for I/O, the data processing efficiency of Dom0 will decrease, so it can affect the performance of overall service. If any VM contains virus, when it communicates with Dom0, it can affect

Dom0 and if it is affected, the entire service provided will fail because Dom0 is the only resource that manages all DomU VMs [6]. Due to these risks of failure Xen enhanced their hypervisor by introducing pass-through model. In this model each DomU VM have direct access to hardware, but only limited to some specific hardware devices due to security concerns. The pass-through model is shown in Fig.6 [6].

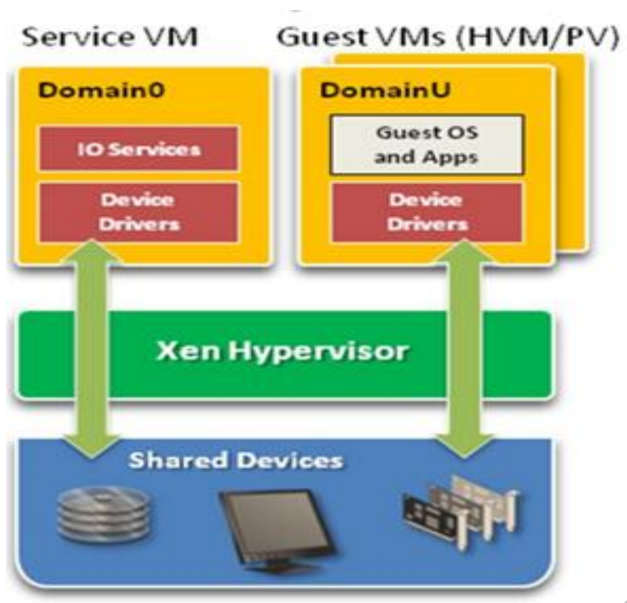

Figure 6. Xen pass-through model.

The use of pass-through model reduces the load on the Dom0 as well as reducing data transfer from the hardware to the DomU. The DomU is not required to have front end drivers in order to send I/O requests to Dom0. Xen Hypervisor version 3.0 supports pass through model to manage clients' VMs on cloud.

B. KVM Hypervisor

In a multitasking environment, when multiple applications are running on a single OS, the OS scheduler maintains a schedule for the applications to run concurrently without any process interference. Similarly in a cloud computing environment, when VMs of several users are running on the provider's OS, a hypervisor is used to manage each VM to ensure safe and efficient workflow, so a hypervisor requires the similar features and components as of OS. Since Linux is an open source OS and it has several OS components such as memory manager, process scheduler, I/O stack device drivers, security manager that are actually required for the implementation of a hypervisors, KVM is developed by implementing Linux kernel module with enhanced hypervisor functionalities rather than reinventing the wheel i.e. redeveloping the developed components from beginning [2]. Each Linux process has two modes of execution user and kernel mode. The user mode is considered as unprivileged and kernel mode is considered as privileged process. The default mode for a process is user mode. It changes to the kernel mode when it requires some sort of services from kernel such as request for writing to hard disk. While implementing the KVM, the developers added a third mode for process, called as guest mode as shown in Fig.7 [7].

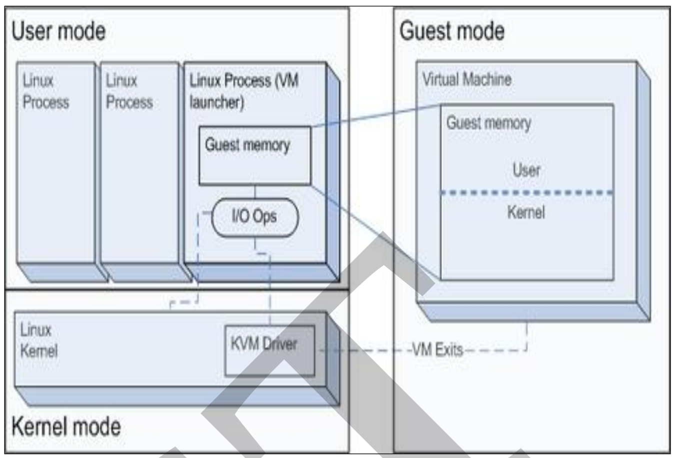

Figure 7. KVM guest mode.

The guest mode itself has two normal modes user and kernel, can be called as guest-user and guest-kernel mode. When a guest process is executing non-I/O guest code, it will run in guest-user mode. In guest-kernel mode, the process handles exits from guest-user mode due to I/O or other special instructions. In user mode, the Linux process performs I/O on behalf of a guest. The model of KVM is shown in Fig.8 [8].

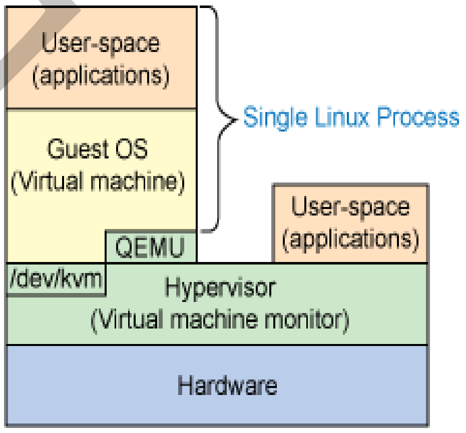
User-space (applications)
Guest OS (Virtual machine)
QEMU
Single Linux Process
User-space (applications)
/dev/kvm
Hypervisor (Virtual machine monitor)
Hardware

Figure 8. KVM architecture.

In the KVM model each guest VM is implemented as a simple Linux process and that process itself is able to run multiple applications concurrently because it is acting as a virtual OS. Each VM is scheduled by standard Linux scheduler. Using KVM, the I/O requests of a guest VM are handled through Quick Emulator (QEMU), it is a platform virtualization solution that allows virtualization of an entire PC environment (including disks, graphic adapters, BIOS, PCI bus, USB and network devices) [2]. Any I/O requests a guest OS makes are intercepted and routed to the user mode to be emulated by the QEMU process. The memory virtualization is provided to guest VMs by dev/kvm device, each VM has its own memory space that provides isolation of the VMs from each other. Physical memory of guests is actually the virtual memory provided to their OSs by KVM hypervisor [7].

III. Xen or KVM: Hypervisors Comparison Analysis

In the previous section we described the use of Xen and KVM. In this section we will compare both hypervisors to decide which one of them is the most suitable for managing VMs. Considering a cloud computing environment, the features of a good hypervisor should include appropriate support of security, memory management, performance and scheduling policy. Our comparison of Xen and KVM is based on the aforesaid features.

A. Security

Since we already discussed, in KVM a VM is implemented as a Linux process, hence the security of VM is based on the standard Linux security policies. The Linux kernel includes Security Enhanced Linux (SELinux) a project developed by the US National Security Agency to add mandatory access controls, multi-level and multi-category security as well as policy enforcement. SELinux provides strict resource isolation and internment for processes running in the Linux kernel. The security policies to secure VMs are defined by using sVirt that is built on SELinux to implement security control features. System administrator uses sVirt to define security policies such as a VM cannot access another VM, but they can share some resources according to the requirements of an organization. In a virtualized cloud environment, hackers can use the hypervisor as main source of attack on overall system. If the hypervisor is compromised, hackers can attack all the VMs on cloud. In order to secure the hypervisor, SELinux and sVirt provide powerful security techniques to secure the hypervisor [2].

On the other hand Xen provides a mechanism of isolating the guests from each other. A DomU guest cannot access other DomU guests, hence if any VM is malicious or affected with virus will not affect other guests. Secondly, Xen supports access privileges for-example, only Dom0 guest is allowed to communicate with hardware. Thirdly, the Xen hypervisor contains a tiny code footprint which limits the areas of attack. Xen is also in collaboration with *"The Invisible Things Lab"*, they focus on identifying security issues in computing infrastructures. Xen works with this group to overcome any security loop holes identified in Xen hypervisor [9].

B. Memory Management

KVM inherits the powerful memory management mechanism of Linux. The memory for a VM is stored as memory for any other Linux process and can be swapped, backed by large pages for better performance. Memory management in Linux is supported by Non-Uniform Memory Access (NUMA) technology. It allows VMs to efficiently access large amounts of memory. KVM supports the latest memory virtualization features from CPU vendors with support for Intel's Extended Page Table (EPT) and AMD's Rapid Virtualization Indexing (RVI) to deliver reduced CPU utilization and higher throughput. Memory page sharing is supported through a kernel feature called Kernel Same-page Merging (KSM). KSM scans the memory of each VM and where VMs have identical memory pages, KSM merges these into a single page that it shared between the VMs, storing only a single copy. If a guest attempts to change this shared page it will be given its own private copy. When consolidating many VMs on the hosts there are many situations in which memory pages may be shared for example unused memory within a windows VM, common DLLs, libraries, kernels or other objects common between VMs. With KSM more VMs can be consolidated on each host, reducing hardware costs and improving server utilization [2].

Xen uses the memory overcommit strategy to provide virtual memory to VMs. Memory overcommit is the technique that provides an illusion of having virtual memory more than the physical memory, in other words the sum of total memory assigned to VMs can be greater than actual physical memory. For example, if a machine has 5GB of RAM and we want to run as many as possible 1GB VMs, we can run maximum four VMs each of 1GB because dom0 requires some physical memory. With the new memory overcommit feature in Xen 3.3, we can run six, ten or even more VMs only on 5GB RAM. The concept of overcommit sounds magical, but actually it's technical. Suppose a domain that is idle or nearly so, is probably not using much memory, this memory can be made available to use in another domain or for a newly created domain. The complicated part is to determine how much memory can be taken away from domains without causing problems for them and even more importantly, how to give the memory back if a domain suddenly needs it again. This careful memory balancing ideally should be done in a management tool that can monitor memory needs of all domains and add or subtract memory from each domain as needed. In certain cases this technique is not useful enough such as environments where all domains are heavily utilizing the memory and none of them are having idle memory. But for environments those require a ratio of high virtual-domains-to-physical-machines and that are willing to make some tradeoffs, memory overcommit can substantially increase VM density and saves cost. Memory is taken from one domain and given to another using the existing Xen ballooning mechanism that has recently improved to be more robust [10].

C. Performance

KVM inherits the performance and scalability of Linux supporting VMs with up to 16 virtual CPUs, 256GB of RAM and host systems with 256 cores and over 1TB of RAM. KVM also supports live migration which provides the ability to move a running VM between physical hosts with no interruption to service. Live migration is transparent to the end user, the VM remains powered on, network connections remain active and user applications continues to run while the VM is relocated to a new physical host. In addition to live migration KVM supports saving a VM's current state to disk to allow it to be stored and resumed at a later time [2].

Xen is very efficient in terms of performance due to the implementation of techniques such as para-virtualization, pass-through model and isolation of hypervisor from underlying OS. Firstly, para-virtualization allows the guest OS to co-operate with the hypervisor to improve overall performance for I/O, CPU and memory virtualization. By being aware that OS is running in a virtualized platform, modified OS is able to assist the hypervisor in a variety of tasks. Secondly, the use of pass-through technology allows a guest domain to communicate with a specific piece of hardware directly without having to send communication to and from the Dom0. Allowing a guest domain direct access to hardware significantly improves time to response for a guest, lowers processing time by eliminating the Dom0 middleman and reduces load on the Dom0 queue but security is still maintained as the guest is restricted in what hardware it can access thereby preventing guest interaction. Thirdly, the isolation of hypervisor from OS ensures maximum performance. Any OS will have a series of tasks that must be scheduled and processed during normal operation. The majority of these tasks are not related to processing the virtualized guests thus can have potential impact on overall performance. The Xen hypervisor is able to process the virtualized guests without any OS overhead and can even be tuned specifically to maximize guest processing based on user demands and requirements for a given guest. The scheduler within Xen is also customized for a virtualized environment thereby ensuring that a Xen infrastructure is capable of meeting the highest user expectations [9].

D. Scheduling Policy

In the KVM model, a VM is scheduled and managed by the standard Linux kernel. The current version of the Red Hat Enterprise Linux kernel supports setting relative priorities for any process including VMs. This priority is for an aggregate measure of CPU, memory, network and disk I/O for a given VM and provides the first level of Quality of Service (QoS) infrastructure for VMs. The modern Linux scheduler accrues some further enhancements that will allow a much finer-grain control of the resources allocated to a Linux process and will allow guaranteeing a QoS for a particular process. Since in the KVM model, a VM is a Linux process, these kernel advancements naturally accrue to VMs operating under the KVM architectural paradigm. Specifically, enhancements including Completely Fair Scheduler (CFS), Control Groups (CGroups), network name spaces and real-time extensions will form the core kernel level infrastructure for QoS, service levels and accounting for VMs. The Linux kernel includes CFS to provide advanced process scheduling facilities based on experience gained from large system deployments. The CFS scheduler has been extended to include the CGroups resource manager that allows processes and in the case of KVM, VMs to be given shares of the system resources such as memory, CPU and I/O. Unlike other VM schedulers that give proportions of resources to a VM based on weights, CGroups allow minimums to be set not just maximums, allowing guaranteed and more resources to a VM if available[2].

Xen provides variety of algorithms for CPU scheduling. Xen API includes Borrowed Virtual Time (BVT), Atropos and Round Robin schedulers that are provided at the booting time for the selection. The BVT, Atropos and Round Robin schedulers are part of the normal Xen distribution but users are allowed to add more algorithms for scheduling. BVT provides proportional fair shares of the CPU to the running domains. Atropos can be used to reserve absolute shares of the CPU for each domain. Round-robin is provided as an example of Xen's internal scheduler API. Domains are statically assigned to CPUs, either at creation time or when manually pinning to a particular CPU. The current schedulers then run locally on each CPU to decide which of the assigned domains should run there. Domains are preemptively scheduled by Xen according to the parameters installed by Dom0. However, a domain may choose to explicitly control certain behavior with the use of a hyper call such as *sched op(unsigned long op)*. This hyper call will request a scheduling operation from hypervisor. The options that can be passed as parameters are yield, block, and shutdown. Yield keeps the calling domain runnable but may cause a reschedule if other domains are runnable, block removes the calling domain from the run queue and cause is to sleep until an event is delivered to it. Shutdown is used to end the domain's execution, the caller can additionally specify whether the domain should reboot, halt or suspend [11].

IV. CRITICAL ANALYSIS: XEN AND KVM

In section-III, we compared the features of Xen and KVM, it is identified that KVM is based on Linux features because memory management, scheduling policy and security of KVM are inherited from standard Linux kernel. In this regard, Xen developers claim that KVM is not a true hypervisor instead it is the conversion of Linux kernel as a hypervisor [12]. On the other hand, Xen developers have added new techniques of memory management, scheduling policy and security to implement the Xen hypervisor. In this regard, KVM claims that it is the process of reinventing the wheel, there is no need to develop these features as they are already available in any OS [2]. This seems to be a kind of war between hypervisors. For an enterprise it is a challenging task to select a hypervisor. But still these two hypervisor are mostly being used and adopted by several enterprises such as Xen hypervisor is currently available in solutions from Avaya, Cisco, Citrix, Fujitsu, Lenovo, Novell, Oracle, Samsung, VALinux, and cloud providers including Amazon, Cloud.com, GoGrid and Rackspace are amongst the many cloud solutions using Xen as their virtualization foundation [12]. In addition to the broad Linux community KVM is supported by some of the leading vendors in the software industry including Red Hat, AMD, HP, IBM, Intel, Novell, Siemens, SGI and others [2]. In order to clarify the efficiency of KVM and Xen for managing a virtualized cloud platform we presented the comparison analysis among them by considering the features that are required for an efficient hypervisor. The comparison analysis clearly represents the efficiency of Xen and KVM hypervisor for managing the VMs on cloud.

V. Conclusion and Future Work

This research paper described the use of hypervisor on x86 virtualized architecture for cloud virtualization. The most demanding open source hypervisors Xen and KVM are compared according to their techniques of supporting security, memory management, performance and scheduling policy. It is concluded that both hypervisors are efficient enough to manage the VMs on cloud but still it depends on the adopting enterprise to use either KVM or Xen. The comparison of their technique will enable an enterprise to judge on the selection of suitable hypervisor for managing their virtualized cloud platform. The discussion and comparison analysis in this research paper was carried out on technical, conceptual and logical basis based on the research conducted from the sources of Xen and KVM. By continuing the future work of this research, we will conduct experimental sessions to evaluate the performance of on Xen and KVM hypervisors.

Acknowledgment

The credit of accomplishing this research paper goes to several entities. First, we would like to thank our parents. They always supported us throughout our complete study life. Secondly, we are thankful to our supervisor for giving us an encouragement to write this research journal. Finally, we are thankful to IJAEST for always giving us the right response at the right time that enables us to publish high quality research papers.

References

[1] Wei Chen, Hongyi Lu, Li Shen, Zhiying Wang, Nong Xiao, and Dan Chen, "A Novel Hardware Assisted Full Virtualization Technique," in *Young Computer Scientists, 2008. ICYCS 2008. The 9th International Conference for*, 2008, pp. 1292-1297.

[2] Redhat, "Kernel Based Virtual Machine [online]", 2009, Available from http://www.redhat.com/f/pdf/rhev/DOC-KVM.pdf , Accessed [4th Jul 2011].

[3] Xen, "Why Xen Hypervisor [online]", 2009, Available from http://www.xen.org/files/Marketing/XenBrochure_Q12009.pdf, Accessed[6th Jul 2011].

[4] T. Naughton, G. Vallee, S. L. Scott, and F. Aderholdt, "Loadable Hypervisor Modules," in System Sciences (HICSS), 2010 43rd Hawaii International Conference on, 2010, pp. 1-8.

[5] Redhat, "Xen Full Virtualization Architecture [online]", 2009, Available from http://docs.redhat.com/docs/enUS/Red_Hat_Enterprise_Linux/5/html/Virtualization/glos.html, Accessed [7th Jul 2011].

[6] Amit Aneja, "Xen Hypervisor: Designig Embeded Virtualized Intel Architecture [online]", 2011, Available from http://download.intel.com/design/intarch/PAPERS/325258.pdf, Accessed [11 Jul 2011].

[7] Duilio Javier, "Linux KVM as a Learning Tool [online]", 2009, Available from http://www.linuxjournal.com/magazine/linux-kvm-learning-tool, Accessed [12th Jul 2011].

[8] M. Tim Jones, "Discover the Linux Kernel Virtual Machine [online]", 2007, Available from http://www.ibm.com/developerworks/linux/library/l-linux-kvm/, Accessed [15th Jul 2011].

[9] Xen Org, "Why Xen [online]", 2008, Available from http://www.xen.org/files/Marketing/WhyXen.pdf, Accessed [17th Jul 2011].

[10] Dan Magenheimer, "Memory overcommit [online]", 2008, Available from http://xen.org/index.php/2008/08/27/xen-33-feature-memory-overcommit/, Accessed [19th Jul 2011].

[11] Xen Team, "Interface manual [online]", 2004, Available from http://www.xen.org/files/xen_interface.pdf, Accessed [21st Jul 2011].

[12] Paula Raune, "KVM and Xen cofounders engage in war of words", 2008, Avaialble from http://www.zdnet.com/virtualization/kvm-and-xen-cofounders-engage-in-war-of-words/415, Accessed[21st Jul 2011].