Optimization of CPU scheduling in virtual machine environments

VENKATARAMANAN VENKATESH

A thesis submitted to the Faculty of Graduate and Postdoctoral Studies
in partial fulfillment of the requirements for the degree of

MASTER OF APPLIED SCIENCE
IN ELECTRICAL AND COMPUTER ENGINEERING

uOttawa

L'Université canadienne
Canada's university

Ottawa-Carleton Institute for Electrical and Computer Engineering
School of Information Technology and Engineering
University of Ottawa
Ottawa, Canada

Abstract

Data centres and other infrastructures in the field of information technology suffer from the major issue of 'server sprawl', a term used to depict the situation wherein a number of servers consume resources inefficiently, when compared to the business value of outcome obtained from them. Consolidation of servers, rather than dedicating whole servers to individual applications, optimizes the usage of hardware resources, and virtualization achieves this by allowing multiple servers to share a single hardware platform. Server virtualization is facilitated by the usage of hypervisors, among which Xen is widely preferred because of its dual virtualization modes, virtual machine migration support and scalability. This research work involves an analysis of the CPU scheduling algorithms incorporated into Xen, on the basis of the algorithm's performance in different workload scenarios. In addition to performance evaluation, the results obtained lay emphasis on the importance of compute intensive or I/O intensive domain handling capacity of a hypervisor's CPU scheduling algorithm in virtualized server environments. Based on this knowledge, the selection of CPU scheduler in a hypervisor can be aligned with the requirements of the hosted applications. A new credit-based VCPU scheduling scheme is proposed, in which the credits remaining for each VCPU after every accounting period plays a significant role in the scheduling decision. The proposed scheduling strategy allows those VCPUs of I/O intensive domains to supersede others, in order to favour the reduction of I/O bound domain response times and the subsequent bottleneck in the CPU run queue. Though a small percentage of context switch overhead is introduced, the results indicate substantial improvement of I/O handling and fairness in resource allocation between the host and guest domains.

Acknowledgements

I would like to express my immense gratitude to my supervisor, Professor Amiya Nayak, who provided me the opportunity, encouragement and support in pursuing my master's studies. Dr.Nayak's guidance and periodic scrutinization of this research work has helped me complete this dissertation without any major impediments.

I would like to thank the Faculty of engineering, University of Ottawa and Department of systems and computer engineering, Carleton University for offering courses with updated contents about latest research trends, which have supplemented me with background knowledge related to this research.

I also thank the Xen community for their open-source virtualization solution, and in general the contributors of the open-source movement for keeping the repositories available for the academic community.

I am deeply indebted to my parents who have endowed me with spiritual and financial support throughout my academic life. This work would not have been possible without the congruence of encouragement and help from my family, the University faculty and friends alike, during the course of my graduate studies.

Venkataramanan Venkatesh

Ottawa, September 2015

Table of Contents

List of Figures

List of Tables

Acronyms

App	Application
ARINC	Aeronautical Radio Incorporated
BVT	Borrowed Virtual Time
CPU	Central Processing Unit
DMA	Direct Memory Access
Dom	Domain
Dom0	Domain 0 / Host Domain
DomU	User Domain / Guest domain
GPU	Graphics Processing Unit
Guest OS	Guest Operating System
Host OS	Host Operating System
HPC	High Performance Computing
HVM	Hardware Virtual Machine
I/O	Input / Output
IDE	Integrated Device Electronics standard
IOMMU	Input / Output Memory Management Unit
KB/s	KiloBytes per second
MAC	Media Access Control
Mb/s	Megabits per second
NWC	Non-Work Conserving
OS	Operating System
PCPU	Physical CPU / Host's processor core
PV	Paravirtualized guests
QoS	Quality of Service
R-Read	Random Read
RTOS	Real-Time Operating System

R-Write	Random Write
SCSI	Small Computer Systems Interface standard
SEDF	Simple Earliest Deadline First
SMP	Symmetric Multiprocessing
S-Read	Sequential Read
S-Write	Sequential Write
TCP	Transmission Control Protocol
VBD	Virtual Block Devices
VCPU	Virtual CPU
VFR	Virtual Firewall Routers
VIF	Virtual Network Interfaces
VM	Virtual Machine
VMM	Virtual Machine Monitor
WC	Work Conserving
NWC	Non-Work Conserving

Chapter 1 - Introduction

Virtualization essentially means the abstraction of the underlying hardware resources namely CPU, Memory and Network interfaces of a machine. While the concept suggests consolidation of multiple vendor hardware, it does not necessarily mean that the hardware is capable of supporting any arbitrary form of virtualization. In most cases, optimal post-virtualization results are achieved only when the best suited virtualization approach, in regards to the machine, is employed. Understanding of the hardware's capacity to handle virtual machines and customizing the virtual machine monitor based on the analysis, produces better performance results rather than on a naïve or a default configuration. Even in case of proprietary operating systems such as Microsoft Windows when properly virtualized, the handling of process calls, interrupt messages and other associated kernel functions are performed with ease and efficiency. Additional support hardware from the vendors such as Intel's VT-x [23] and AMD's AMD-V [24], which are specialized for virtualization are to be taken advantage of, to promote platform support. Given the required hardware support, many virtualization solutions are available in the market such as VMware's vSphere [3], Citrix's Xenserver [25] and Microsoft's Hyper-V [26] among others. Another well-known open-source virtualization solution is Xen [5], a baremetal hypervisor, which proved to be the best option for an academic research on best virtualization practices and hence, the core component of this research work. A detailed study of the Xen hypervisor's architecture and its functionality, with emphasis on the CPU scheduling schemes used, provided some valuable insights into, how customizing the hypervisor's components at the baremetal level alters its behaviour to suit the user's needs.

1.1 Motivation

According to a report [43], the overall data centre efficiency is still at 12% - 18% with most of the servers remaining idle while drawing precious energy. Proliferation of data stored in

the cloud has sprung the construction of a high number of server farms by organizations. This 'server sprawl' problem has been many years in the making, thus serving as the root cause for induction of virtualization into the server environment. Hussain et al. in [44] have addressed the 'server sprawl' problem by using virtualization-based server consolidation and storage unification through storage area network (SAN).But, the risks that come with virtualization must also be considered to avert a crisis when an unprecedented event occurs, since consolidating servers also means introducing a single point of failure for all the applications running on the hosted virtual machines. Additionally, there is the problem of virtualized legacy applications not being able to deliver near-native performance as that of a dedicated application server. Through performance based economical promotion, organizations can be convinced to make compromises on other minor issues which prevent them from accepting virtualization solutions.

While the Xen hypervisor [6] addresses some of the outstanding issues like security by providing isolation among virtual machines and provisions for adapting different resource allocation schemes to cater to the needs of the hosted applications, a near-native performance for every hosted application remains the ultimate goal of the developers in the Xen open source community. Regola and Ducom in [42] have analysed the potential of hypervisors for high performance computing by benchmarking Open VZ and KVM with focus on I/O throughput. HPC applications pose the challenge of maximizing throughput with minimal loss of CPU and I/O efficiency, which proves to be an intriguing setup for research in a hypervisor environment.

The motivation for this research originates from the reasoning that, since CPU scheduling algorithms are accounted for majority of the factors affecting the performance, this component of the hypervisor's architecture must be subjected to intensive research for arriving at sustainable solutions to performance issues. This work aims at creating perspectives about the features of these scheduling algorithms, which, otherwise, have been overlooked in the existing research material [28, 30, 32, 34, and 38]. Understanding the limits of a scheduling algorithm is of paramount importance when consolidating server hardware and this research

provides an essential insight into the CPU scheduling characteristics and their impact on VM performance.

1.2 Research problems and Objectives

While the usage of VMMs for sharing the underlying physical resources promotes appreciable resource utilization, the resource allocation strategy is orchestrated by the VMM's CPU scheduler. The efficiency with which the processors, memory, network and disks are virtualized depends entirely on the scheduling algorithm's nature. The emergence of the multi-core and multiprocessor architecture of computing devices has made CPUs more powerful, and set forth a new frontier for the existing virtual machine technologies to compete in. The VMM schedulers are expected to exhibit more sophisticated handling of VMs to be accepted for industrial deployment. Based on the multitude of hypervisors in the current market the essential benchmark qualities of a hypervisor's CPU scheduler are as follows:

- **Scheduling Fairness:** Each VM should get a fair share of the hardware – equal shares in case of non-proportional share scheduling and 'weight-based' shares in case of proportional share scheduling. Ensuring fairness among virtual machines directly influences the end-user experience in hosted VMs.
- **CPU utilization:** The efficiency of a scheduler is pre-dominantly determined by the extent to which it utilizes the hardware available. Restricting VMs to usage limits can adversely affect the utilization of the CPU, since the processor shares of idle VMs are left unused by the starving VMs.
- **Response time:** Latency-sensitivity is of paramount importance in virtual machine environments since poor response times in VMs has been the most putforth point against employing virtualization for the cloud environment. Tending to I/O intensive VMs occupies higher stance than the individual priorities of VMs in consolidated hosted servers.
- **Load balancing:** In multiprocessor and multi-core processor systems, processor cores which are lightly loaded or comparatively less heavily loaded can be forced to be yielded to reduce the work load on other cores. This automation of load balancing

among processor cores or processor pools facilitates the VMM to achieve high CPU utilization rates.

- **Domain handling:** The scheduler's ability to handle compute intensive, I/O intensive or bandwidth intensive domains with optimal efficiency, in case of homogenous VMs. In case of heterogeneous VMs, for instance, a combination of webserver and database server applications, the scheduler must be able to handle such VMs with multi-faceted workloads, within acceptable levels of degradation in processing times.

The primary objectives of this research are threefold: 1) Comprehensive study of virtualization technology and the aspects of a hypervisor's design, 2) Study of Xen hypervisor's architecture and a comparative analysis of its CPU scheduling algorithms from historical ones to the most recent, 3) Enhancement of a credit-based algorithm for handling compute-intensive applications via implementation of a novel VCPU scheduling scheme.

1.3 Research Contributions

The main contributions of this research are as follows:

1. Analysis and performance evaluation of the CPU scheduling algorithms in Xen hypervisor

   Devised a methodology for conducting a performance study and comparative analysis of the schedulers available for SMP hosts. Identification of a best choice scheduler for consolidating server workloads.

2. Optimization of VCPU scheduling strategy for handling compute and I/O intensive domains

   Improvement in the response time of I/O bound domains via implementation of a work- conserving scheduling strategy to ensure optimum CPU utilization. Evaluation of the new credit-based VCPU scheduling scheme against existing CPU schedulers using the same methodology used for analysis of the schedulers.

1.4 Thesis Organization

The remainder of this thesis is organized as follows:

Chapter 2 – Virtualization and Hypervisor Environment presents an overview of the virtualization technology and its taxonomy. This section also briefly discusses about the architecture of Xen hypervisor and its major components along with the factors affecting the scheduling mechanism of VMs.

Chapter 3 – Analysis and performance evaluation of CPU scheduling algorithms in Xen hypervisor covers the in-depth analysis and performance evaluation of the CPU scheduling algorithms in Xen hypervisor.

Chapter 4 – Optimization of CPU Scheduling for handling compute and I/O intensive Virtual Machines proposes an enhanced credit-based VCPU scheduling scheme to improve the CPU utilization and processing times of VMs with heavy computational and I/O workloads.

Chapter 5 – Conclusion & Future Work gives a summary of the thesis and concludes the experimental inferences, while outlining some visions for future research.

Chapter 2 - Virtualization and Hypervisor Environment

2.1 Background

Many organizations today in the field of Information technology prioritize reducing their future infrastructure cost requirements and optimizing the usage of their resources currently in service. Such policies taken up by huge companies can possess serious challenges to the IT engineers when it comes to designing their Enterprise Architecture Framework. During the process of working on solutions for a plethora of problems arising in this endeavour, the engineers have to bear in mind not to sacrifice any crucial elements which may adversely affect the business value of the firm. Such elements may include Quality of Service (QoS), Service rate, Service availability and Service diversity. Among a number of novel solutions which address all these issues of an enterprise, is the most proposed and highly sought-after Virtualization of resources. In [38] Jia et al. have discussed why enterprises take virtualization as the optimized choice in different stages. Their analysis shows that enterprises choose virtualization either in pioneering stage or in their developing stage since the motives vary in different growth stages. Virtualization has great potential in building fault tolerant systems with more flexible backup and disaster recovery systems for a platform. In [41], Xu et al. introduce a consistent backup mechanism for disaster recovery using container based virtualization.

2.1.1 Virtualization

The practice of virtualization has been there for many years since the development of CP-40 (the first operating system which implemented complete virtualization) by IBM Corporation in 1964, mainly for the purpose of partitioning the Mainframe computer so as to reduce maintenance costs. Since then the concept of virtualization has been applied, modified and

reinvented to suit many other needs which have developed contemporarily with the evolution of operating systems, storage devices and the Internet. Virtualization is defined as the concept of introducing a layer of abstraction between the hardware and the operating system and the applications that run on top of the operating system [1]. This layer of abstraction is nothing but a host program, known as the virtual machine monitor or the hypervisor, which enables the host computer to support multiple and identical execution environments. Figure 1 shows the comparison between the traditional and virtualized computer architecture. As mentioned earlier, the VMM or the Hypervisor enables the hardware layer to support multiple instances of the same or different operating systems.

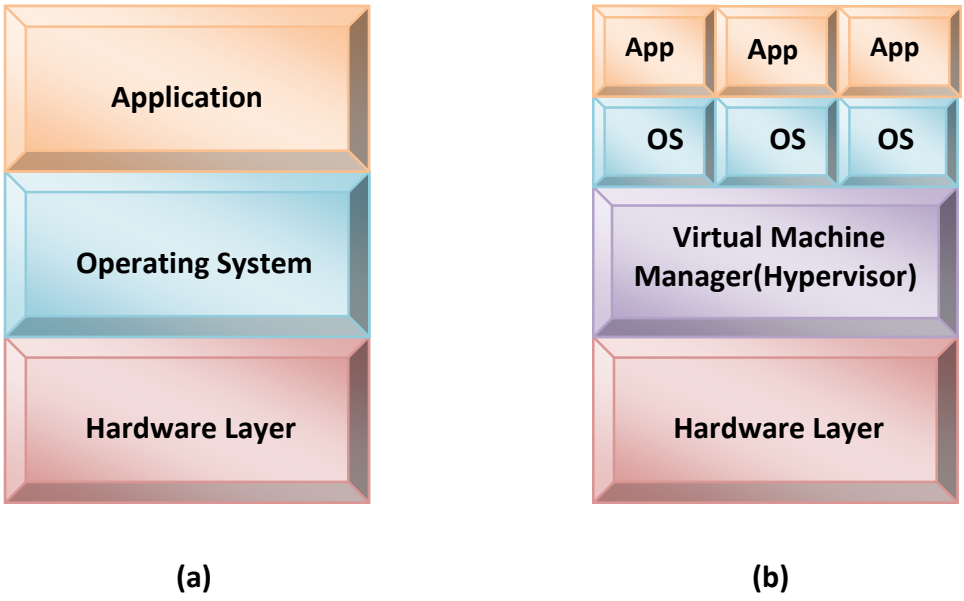


Figure 1: (a) Traditional computer architecture and (b) Virtualized computer Architecture

2.1.2 Virtual Machine

A virtual machine (VM) is a logical unit which encapsulates the guest operating system and the applications running on top of it. Virtual machines are simulations of abstract or real machines, created by transforming a physical entity into a software entity that can be run upon another machine (host). The virtualization layer decouples the virtual machines from

the hardware of the host machine by allocating the VMs only logical partitions of the underlying hardware, thus enabling migration of the VMs from one host to another. Like any other physical computing environment, virtual machines will also request for CPU, memory, hard disks and network resources. The hypervisor translates all these requests to the underlying hardware layer and allocates these resources to the requesting VMs based on the scheduling model. All the VMs are isolated from each other which prevent the processes of different VMs interfering with each other and the host operating system.

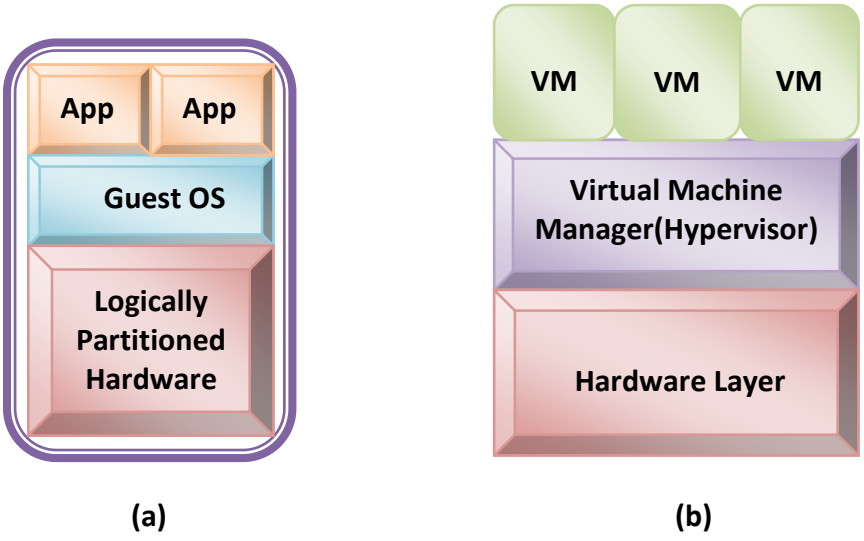


Figure 2: (a) A Virtual machine representation and (b) A Virtualized architecture running three VMs

Each VM can be made to run different operating systems so that applications which are compatible only on particular platforms can be run in parallel. VMs enable administrators to run multiple business operations at reduced infrastructure cost and are naturally used as backups for disaster recovery or in case of hardware failures to continuously support business critical applications. The number of VMs that can be mounted on a host depends on the overhead that will be added as a result of managing multiple Inter-process communications (IPC) and so, a balance has to be struck between performance and flexibility.

2.2 Taxonomy of Virtualization

Virtualization can be classified into many types based on the architecture and the functions of the virtual machine manager which forms the abstraction layer over the physical hardware of the host machine.

2.2.1 CPU Virtualization

Virtualization of the CPU in x86 architecture can be a challenging task and VMware were the first in the industry to successfully reach this important milestone. In the x86 architecture, there are four levels of privileges known as Ring 0, Ring 1, Ring 2 and Ring 3 as shown in the figure 3 [3].Since user applications require lower levels of access and minimum privileges they tend to run in Ring 3, whereas, the OS requires higher levels of access such as control over the instruction set and so, they tend to run in Ring 0.

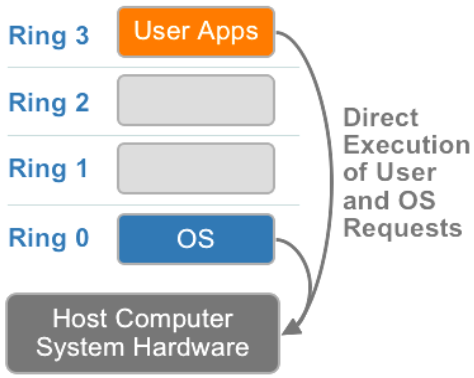


Figure 3: x86 privilege level architecture without virtualization [3]

Also, in the x86 architecture the OS runs directly on the underlying hardware thus assuming ownership over the whole hardware. Some of the instructions have different semantics outside of the Ring 0 and this complicates the process of trapping and translating these requests during runtime. VMware solved this problem by using binary translation techniques [3] which run the VMM in Ring 0 and the OS in any of the user rings. This approach grants the OS higher levels of access than the user applications in Ring 3, but lesser privileges than the VMM in Ring 0.

2.2.1.1 Full Virtualization

Full virtualization requires complete virtualization of the underlying machine, including memory access, I/O operations, interrupts and instruction set, in a way that there should be no difference in terms of resource access between the virtual machine and the host machine. In full virtualization, the virtual machine manager is run like any other application on the host operating system. Users can install the same or any other OS in the VMM application and run in parallel with the host OS. The VMs in this case are completely decoupled from the host hardware and so, there is no need for any modification in the guest OS. In full virtualization the guest OS is neither aware that it is managed by a VMM nor that the hardware allocated to it is only virtual and not physical in nature. This allows migration of the VMs from one host to another or consolidation of physical resources to provide more computing power to the host. Figure 4 shows a representation of full virtualization.

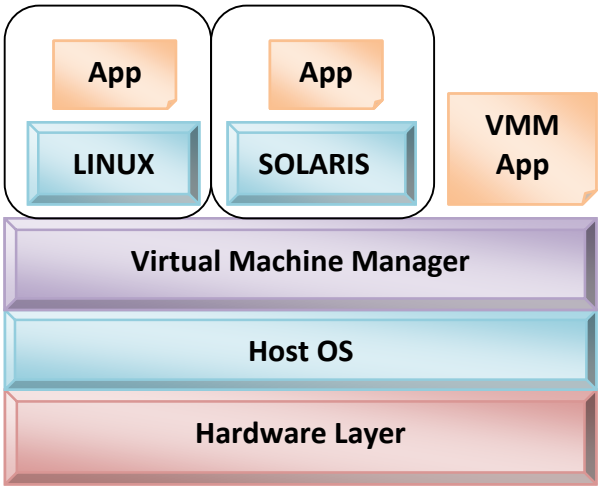


Figure 4: Architecture with Full Virtualization

VMware achieves full virtualization in x86 architecture by a combination of binary translation and direct execution. The guest OS is run in a level outside of Ring 0, but has more privileges than the user level code. The user level code is executed directly on the processor for high performance virtualization.

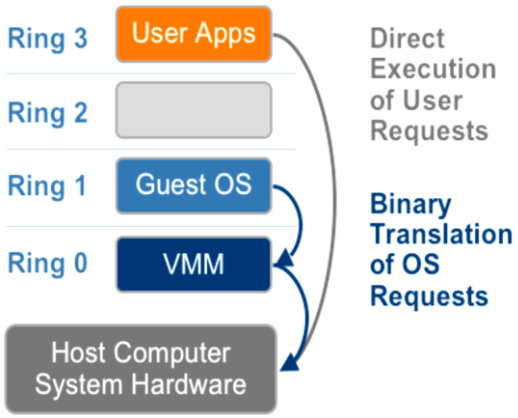


Figure 5: The binary translation approach to x86 Full virtualization [3]

The hypervisor replaces all non-virtualizable kernel code with a new set of instruction code which can run on the virtual hardware and effectively replace the original instruction set. All such OS instructions are trapped and translated by the hypervisor as shown in Figure 5 [3], whereas the user level code runs directly on the processor achieving performance levels equivalent to that of the host applications. VM security and ease of migration are the major advantages of full virtualization while performance takes a severe toll when compared to direct on hardware execution. In [40] Chen et al, propose a novel hardware assisted full virtualization technique - dynamic binary translation in DIMM (DBTIM). They aim at making the host CPU virtualizable by integrating a reconfigurable dynamic binary translation chip into a DIMM (Dual In-line Memory Module).

2.2.1.2 Paravirtualization or OS-assisted virtualization

Figure 6 [3] shows the representation of VMware's paravirtualization approach to x86 architecture wherein the virtualization layer or the hypervisor is inserted between the guest OS and the hardware. In the paravirtualization approach, the non-virtualizable instructions are sent as hypercalls to the virtualization layer for replacement. The guest OS kernel has to be modified to send these hypercalls and hence, it will be aware of its execution over a virtual hardware.

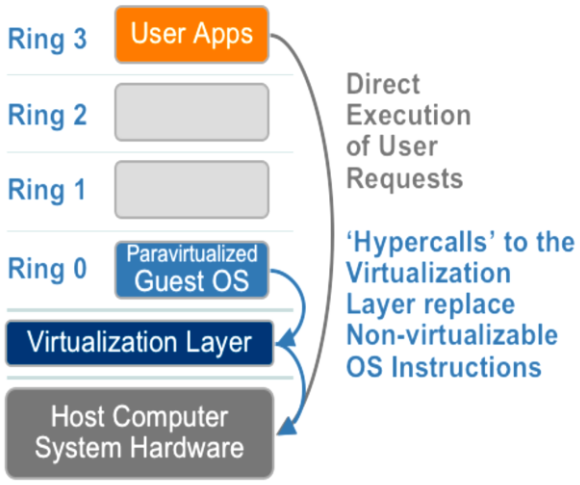


Figure 6: Paravirtualization approach to x86 architecture [3]

In paravirtualization, the compatibility of the host is sacrificed since the guest OS has to be modified consequently reducing VM portability. Since the VMM is simple, the performance levels can be brought closer to that of architectures without virtualization [3].

2.2.1.3 OS-Level Virtualization

Figure 7 shows a representation of an architecture using OS-level virtualization. In OS-level virtualization there are multiple instances of the host OS running in virtual machines over the virtualization layer.

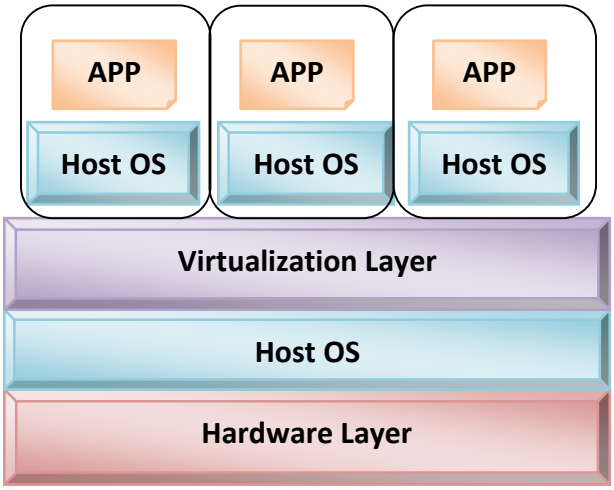


Figure 7: Architecture with OS-Level virtualization

This approach is employed when multiple users are using the same operating system and isolation is a high priority. The underlying host OS runs only a single OS kernel which is then projected as multiple virtual containers of the host OS on which users can run applications as they would on a standalone machine. This approach is commonly used solutions in server virtualization since multiple instances of the same server can lead to consolidation of resources. Major drawback is that only one type of OS(same as host OS) can be run in the virtual machines since only a single kernel is run by the host OS [1].

2.2.1.4 Bare-metal Virtualization or Hardware-Layer Virtualization

Figure 8 shows a representation of architecture with hardware- layer virtualization. In this type of virtualization the Virtual machine manager or the hypervisor runs directly on the hardware layer without the host OS as a platform. The installation of the hypervisor directly onto the hardware requires direct virtualization of I/O, memory, CPU and network interfaces.

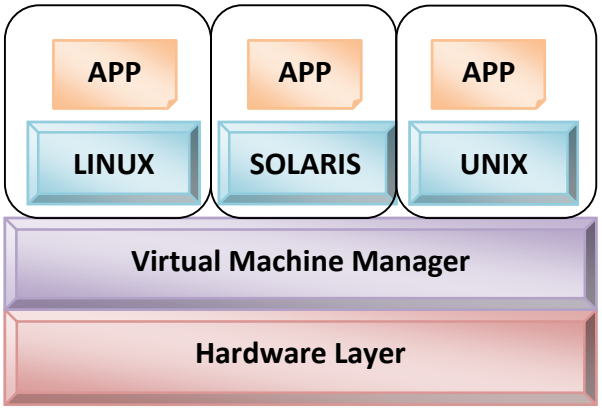


Figure 8: Architecture with Hardware-Layer Virtualization

As shown in figure 9[3], VMware achieves hardware-layer virtualization by running the VMM in the root mode with higher privilege levels than Ring 0. Intel and AMD introduced VT-x and AMD-V respectively in 2006 to provide hardware support for this approach. The OS level instructions and calls are trapped automatically by the VMM at the root level thereby removing the need for binary translation or paravirtualization.

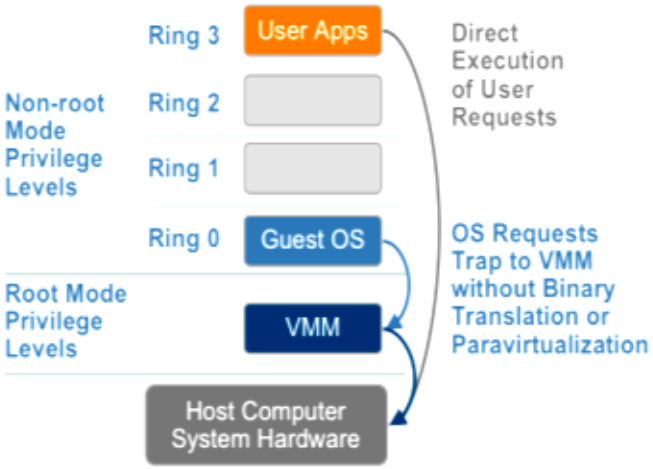


Figure 9: x86 virtualization with hardware-assist approach [3]

Since the hardware assist available for this approach is of the first generation, the programming model is rigid which lowers flexibility allowing these hardware features to be used in only limited cases [3].

2.2.2 Memory and I/O Virtualization

The resources required for a virtual machine to run are the same as those required for a standalone machine such as CPU, memory, I/O interfaces, GPU and network hardware interfaces. Like CPU all these other resources can also be virtualized using certain approaches as discussed in the following sections.

2.2.2.1 Memory Virtualization

Virtualization of memory for a virtual machine is performed in a way similar to that of virtual memory management by the operating systems. The applications running in the virtual machines require contiguous memory allocation which need not necessarily be physical. Direct Memory Access (DMA) for guest VMs can prove to be difficult because of the high probability of same memory addresses used by both the host OS and guest OS, resulting in corruption of memory segments. IOMMU prevents this likely memory corruption issue by re-mapping

the addresses accessed by the hardware based on the same address translation table used for mapping guest-physical address to host-physical addresses.

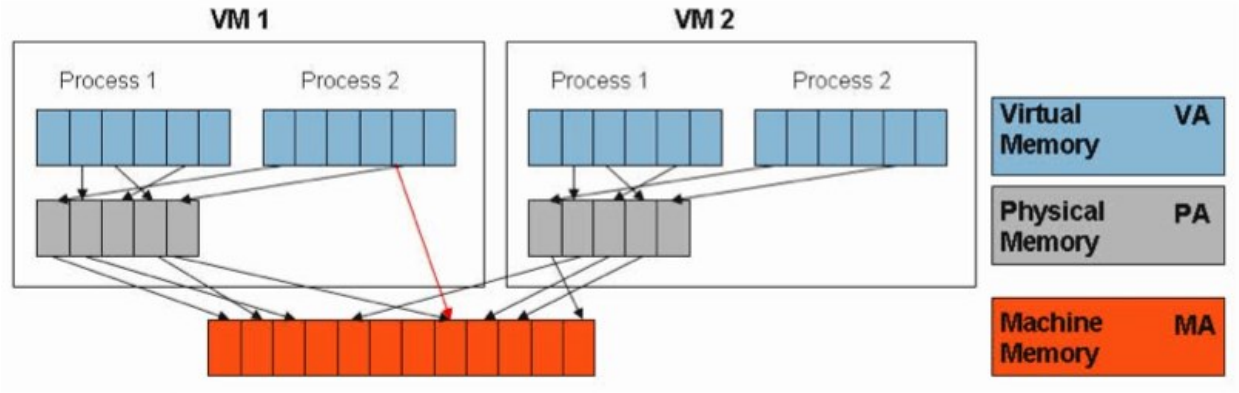


Figure 10: Memory virtualization [3]

Usually x86 architecture provides a memory management unit (MMU) and a translation lookaside buffer (TLB) for memory virtualization. The MMU has to be virtualized in order to run multiple virtual machines with guest operating systems as shown in figure 10[3]. The virtual page numbers are mapped to the physical page numbers stored in the page tables of the guest physical memory and the operating system keeps track of these mappings which are then used by the VMM. The guest operating systems do not have direct access to the memory and so the VMM has to manage the mapping between the guest physical memory and the underlying machine memory. In some cases to avoid two levels of memory translations, the VMM uses the TLB to directly map the virtual memory to the machine memory (indicated by red line). This approach of MMU virtualization in x86 based systems increases the overhead on any virtualization technique [3].

2.2.2.2 I/O and Device Virtualization

As shown in figure 11[3], the virtual machine manager or the hypervisor manages a standard set of hardware devices to be emulated and provided as virtual devices to the virtual machines. The standardized set of devices will not be subject to any change regardless of the

physical devices available with the host machine. This approach boosts portability and compatibility of virtual machines across multiple platforms [3].

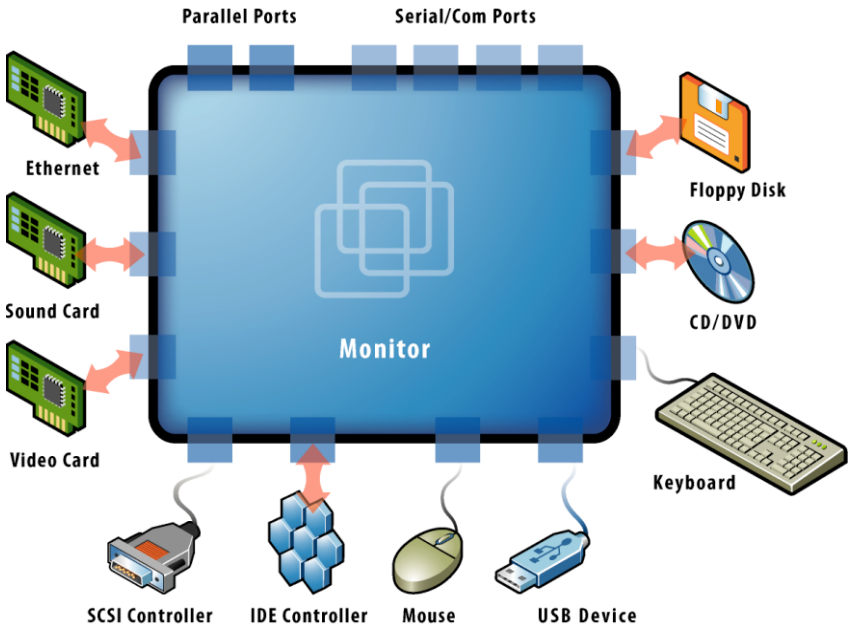


Figure 11: Device and I/O virtualization [3]

In figure 12 [29], a typical virtual switching infrastructure shows the intercommunication between a Hypervisor Virtual Switch and a Physical Switch which are managed separately. Also seen in the figure are the virtualized network interface cards communicating with their counterpart virtual ethernet ports on the switch. In Xen, pairing between a virtual network device and a PV device is made based on the similarity of MAC address and configuration. Such an arrangement enables the guest to make an uninterrupted transition from the emulated device to the PV device as soon as a host driver becomes available [30]. The hypervisor employs its built-in toolstack to assign random or fixed MAC addresses, whichever is preferred, to both virtual and physical devices. VM to VM communication is made possible in Xen by creating a software bridge in the backend domain which enables all the domains to appear on the network as individual hosts.

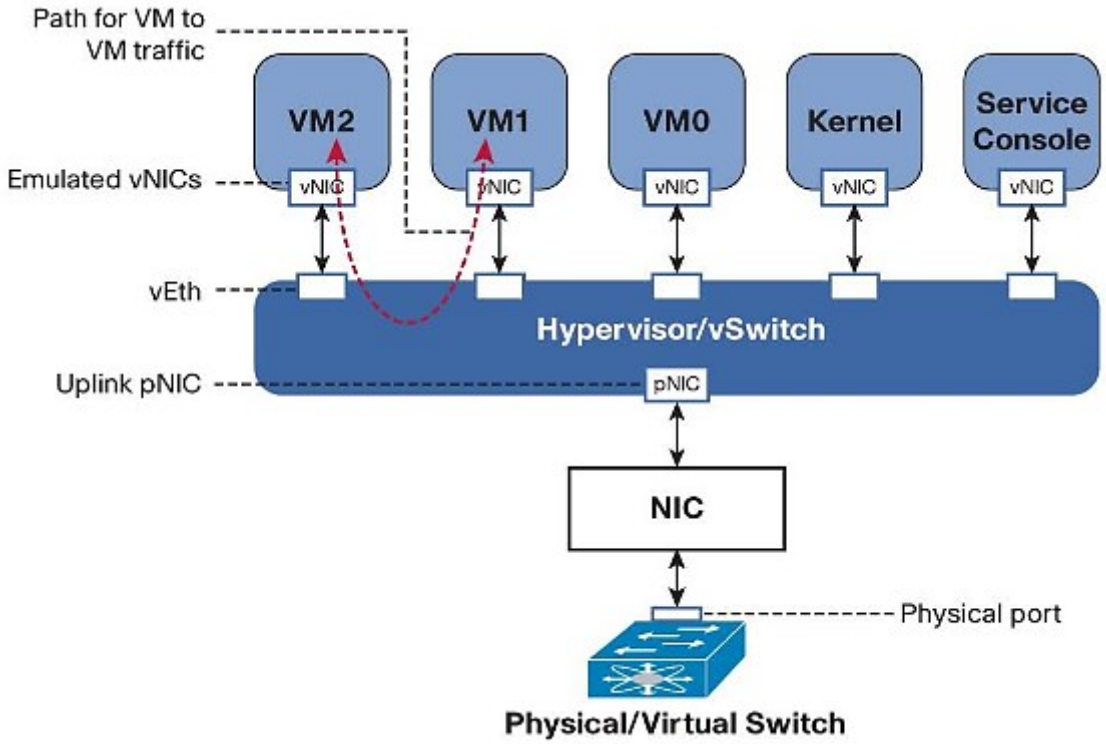


Figure 12: Network Device Virtualization [29]

In a conventional network, a high volume of virtual interrupts arise by virtue of the communication from and to the guest VMs, and the back-end drivers are incapable of efficiently using the underlying resources such as CPU. Guan et al [28] have proposed some optimization methods for enhancing the networking I/O performance by efficient interrupt coalescing and virtual receive-side scaling for leveraging multicore processors.

2.2.3 Network virtualization

A network of computing elements consists of both hardware and software resources rendering services to the users of the machines in the network. In order to virtualize a whole network, all the resources regardless of their nature have to be unified into a single software entity capable of providing the same services as that of a physical network. Typically the

functionalities of a network comprise of connection establishment, error control, flow control, congestion control, multiplexing, routing, switching, forwarding and protocol stack addressing.

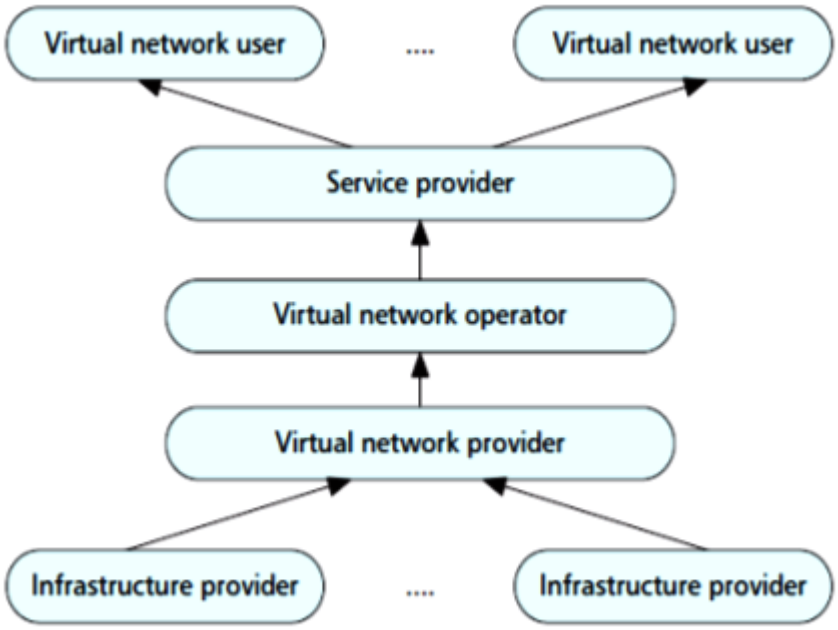


Figure 13: Network virtualization architecture [4]

Figure 13 [4] shows a general architecture for network virtualization where the resources under the virtualization layer are transparent to the virtual network operators. In network virtualization the bandwidth is split into independent channels which can be assigned to any particular server or an element of the network [4]. This enables the operators to create and maintain multiple, isolated logical networks on a physical substrate or hardware. Network virtualization offers network protection, research independence, network testing, migration and optimized usage of hardware resources.

2.2.4 Application Virtualization

In application virtualization, the application need not necessarily be installed on the host to be executed. The virtualized application can use the resources of the host such as memory, I/O device drivers, graphics drivers and hardware interfaces. This approach is used when a

single application is intended for multiple users and each user requires an isolated instance for using the application. Major drawback of this approach is that applications which require heavy OS integration are almost impossible or difficult to virtualize. The common approaches by which application virtualization is rolled out to the end users are:

- **Application streaming** : The application is transferred in stages starting with the critical segments of the code followed by data and customization settings rather than the entirety of the application package. However for the application to run on the user end a lightweight client software is needed before startup.

- **Remote Desktop Services** : One of the more familiar features from Microsoft Windows is this virtualization component, which enables a remote user to access applications and shared data hosted on a computer located in any network accessible through the Internet. Commonly used for remote troubleshooting RDS is a server-based presentation/application sharing service which runs using the Remote Desktop Protocol.

- **Desktop virtualization** : Virtual Desktop Infrastructure (VDI) is the general term used to refer to instances of multiple desktop environments made available from a server based hosting platform. Desktop virtualization has improved portability, ease of access and management of different desktop environments by severing the ties between desktop OS and dedicated hardware.

2.2.5 Server virtualization

According to market experts [5], in the current industrial scenario servers are utilized at 15% to 20% of their original capacity, which tends to be very unproductive in the long run. To tackle this acute problem, two solutions are usually employed namely server consolidation and the use of blade servers. Blade servers reduce over usage of space and power by replacing the conventional servers with multiple thin, modularly designed units mounted on a rack. But in order to reduce the usage of hardware resources, server consolidation through server virtualization is the most common approach.

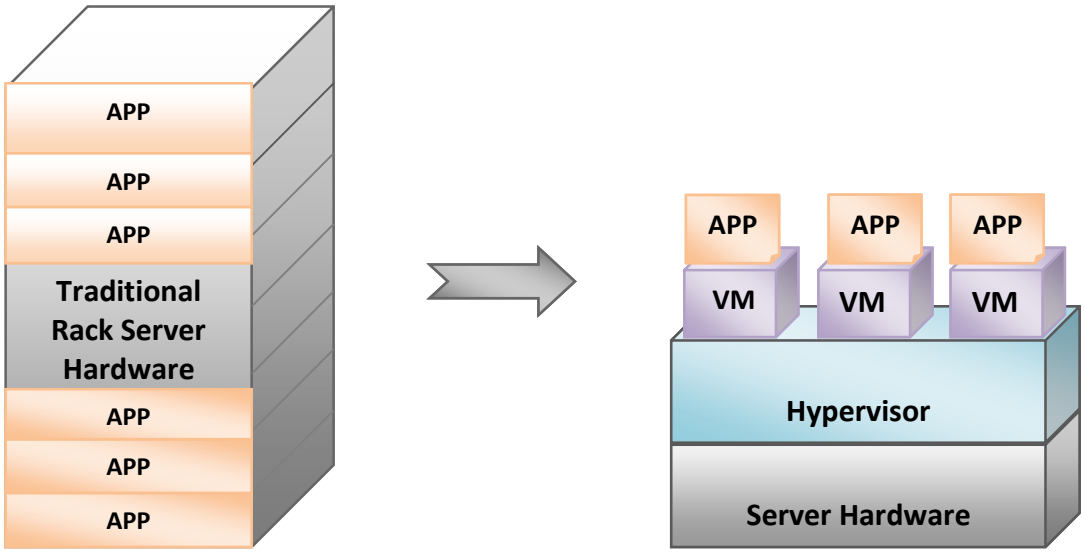


Figure 14: Server consolidation through virtualization

By virtualization of servers and consolidating the hardware into resource pools, hardware utilization can be brought up to 80%. There are a multitude of factors to be considered and decisions to be made before virtualizing a server, as discussed below.

2.2.5.1 Application Performance

Performance of the applications running on the virtual servers is important since it directly impacts the user experience. Unless the performance is close to near-native levels, the virtualized servers can only be capable of running non-business-critical applications. To accommodate a wide range of applications or products, the performance loss after virtualization can be tolerated between 1% and a maximum of 60%. Efficiency of applications post-virtualization can vary, and some can be close to the efficiency of the original physical environment, while others maybe inferior to the acceptable levels expected by end-users. Usually hardware virtualization approach causes heavy performance loss, whereas the operating system virtualization approach does not have severe effects on the performance [5].

2.2.5.2 Resource Management

Different virtualization technologies follow different approaches for partitioning of the physical and allocation of the virtual resources. Some resources can be allocated dynamically during run time while some resources require rebooting the server for the allocation to be complete. The scheduling policy employed should be such that the routines for dynamically allocated resources and other resources which cannot be allocated in runtime have been clearly distinguished at the algorithmic level. While such care has to be taken for hardware virtualization, in operating system virtualization there is flexibility in resource management allowing resource allocation in real-time without any interruption in the functioning of the virtual servers.

2.2.5.3 Virtualization Management tools

In a virtual server environment, though the hardware cost and other significant establishment costs play a big part, none of them measure up to the cost of management of the virtualized server. Periodic maintenance routines such as operating system and application updates, patches, backup, installation, commissioning or decommissioning of scheduling policies add up to quite a large part of the total maintenance cost. Since all the resources are pooled together during virtualization, the tools used for managing the distribution of these resources play a vital part in optimizing overall maintenance costs. Depending on the virtualization solution the availability of management tools varies. Specific virtualization solutions provide appropriate and a good number of tools with products packaged together. In some cases the tools maybe very expensive, which leads to the buyer selecting products based on the gross value. Virtualization Software vendors like VMware include conversion tools like vCenter Converter and migration tools along with their virtualization software. During the migration process from physical server to virtual server, the resource utilization measures are considered to be of high importance when compared to any data that is being migrated.

2.2.5.4 VM Migration

Since virtualization completely decouples the virtual machine from the underlying hardware, the virtual machines or in this case the virtual servers are free to migrate among the physical servers for the purpose of load-balancing or fault tolerance. Virtualization tools such as VMware's VMotion [7], as shown in the figure 15 [7], facilitate the migration of virtual servers from one physical server to another. Major advantages of migration include moving all the VMs from a server experiencing downtime or from a server which is scheduled for an operating system update, to another server thus maintaining application availability in any case. All virtualization solutions have different software limitations, which creates different cost-benefit ratios when each of them is applied to a scenario. In order to achieve zero-downtime in a server environment, sophisticated and high-end solutions are required which will have high success rate in migrating all the VMs to another server in the network in a short period of time, so as to prevent any of the applications from entering a standby state.

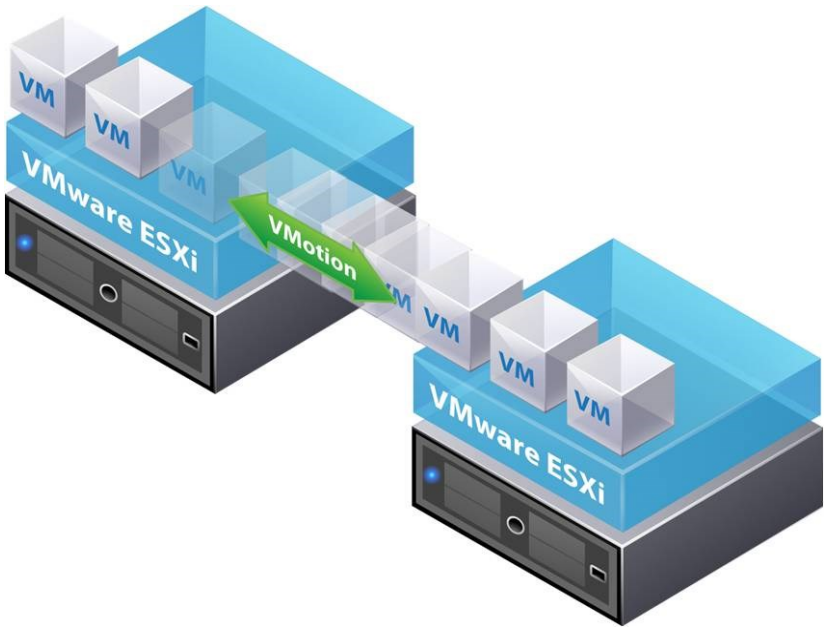


Figure 15: VMware's VMotion migrating VMs between servers [7]

In [22] Wood et al. have developed a scheme for monitoring and detecting hotspots, and remapping/reconfiguring of VMs named Sandpiper. The system chooses VMs for migration based on a volume-to-size ratio (VSR), which is calculated based on CPU, network, and memory loads. In organizational networks, especially in enterprise architectures, migration has to be paid foremost attention in order to sustain a service distribution with high availability for ensuring the continuity of business critical applications.

2.2.5.5 Security

Virtualization has a benefit in terms of multiple machines sharing the same physical substrate over a hypervisor managing the process calls and system routines of all the VMs, but if isolation of these machines is not taken care of properly, it may lead to several security loopholes in the server. If the handling of the VMs is not secure then an attacker with access to one of the VMs can gain access to any other VM using the process calls as a conduit for listening to sensitive information. Also, if the attacker is able to escape the VMs then accessing the host machine is relatively easy. Gaining access to the root level in the host machine which has the highest privilege levels can expose all the sensitive information passing through the server to the attacker. Isolation is of the highest priority when deciding the type of hypervisor to be used while virtualizing a server trusted with sensitive data. In [31] Sailer et al have proposed a new security architecture named sHype which ensures security at the VM level thus providing a strong foundation for sophisticated security measures at the higher layers.

2.3 Xen Hypervisor Architecture

Xen is an open source virtualization platform used by some of the largest cloud hosting companies such as Amazon Web Services, Rackspace Public Cloud, Verizon Cloud and a multitude of other virtualization solution manufacturers [27]. Also, the open source cloud orchestration project OpenStack has intergrated Xen into its solution portfolio. Xen is widely known as a high performance resource-managed VMM which supports applications such as server

consolidation, co-located hosting facilities, distributed web services, secure computing platforms and application mobility. Being a type 1 or bare metal hypervisor Xen is capable of hosting commodity operating systems, with modifications made to the source code in para-virtualization mode (PV guests) and non-modified operating systems in hardware-assisted virtualization mode (HVM guests).

The architecture of Xen is depicted in Figure 16 consisting of two elements, namely the hypervisor and the driver domain. The function of hypervisor layer is to provide an abstraction layer between the guest operating systems and the actual hardware by creating virtual counterparts of the physical hardware components. The driver domain or host domain or Domain 0 (“dom0”) is a privileged VM which manages other guest VMs or unprivileged domains or user domains (“domU”), depicted in the Figure 1 as Domain 1 and Domain 2.

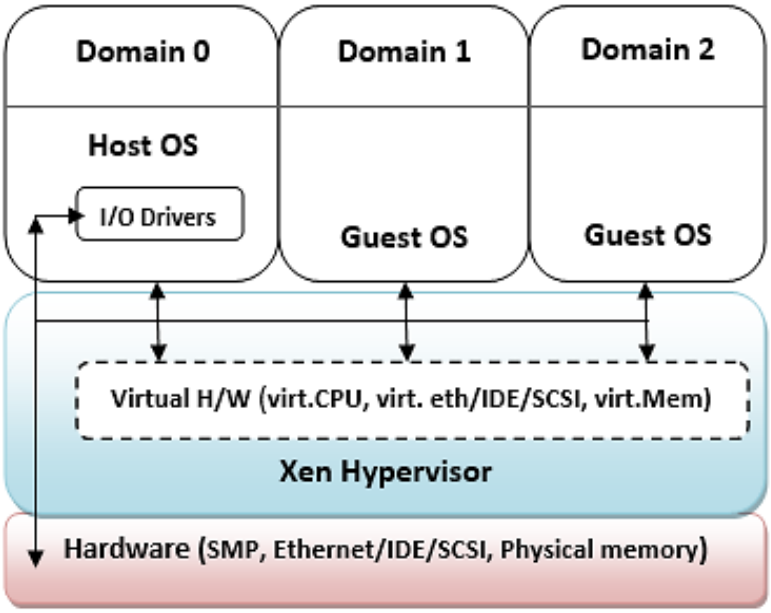


Figure 16: Xen Hypervisor Architecture

The term driver domain for dom0 is mainly due to its responsibility of coordinating the I/O operations on behalf of the user domains, since the guest domains do not have direct access to the I/O hardware. In earlier versions of Xen, the hypervisor managed the I/O operations by creating simple device abstractions for the guest domains to access. This functionality has

been transferred to the dom0 and the depiction in the Figure 1 shows both the earlier and current methodology of I/O management by Xen. This delegation of responsibility to dom0 allows the hypervisor to focus more on virtualizing other hardware components such as CPU, disk drives and network interfaces.

2.3.1 Control Plane

In Xen, the hypervisor provides only basic control operations such as scheduling the CPU among domains, access control to the data blocks and network packet filtering. The control plane management software which runs over the guest OS takes care of complex policy decisions, such as admission control. The dom0 created at boot time is permitted to use the control interface and this privileged domain is responsible for hosting the application-level management software. The administrative functionality of creating new domains and terminating existing ones lies with dom0. Additionally, the ability to control the scheduling parameters, physical memory allocations and the access to physical disks and network devices are also part of the control management software. The authority is not limited to CPU and memory resources since it also extends to the creation and deletion of virtual network interfaces (VIFs) and virtual block devices (VBDs) [6]. The access control information of these virtual I/O devices determine which domains are allowed to access them and the restrictions imposed such as read-only in case of VBDs. The development of higher-level and automation tools in this category will further ease the application of these administrative policies.

2.3.2 Memory, network and disk

When each domain is created, the memory allocation parameters are specified thus finalizing the static memory reservation for the domain. Typically a minimum and maximum memory reservation is specified during creation so that when the domain requires additional memory for certain applications, it can claim memory pages, up to the maximum limit, from Xen and when conservative memory strategy is used the domain relinquishes the memory pages to Xen. For example, XenoLinux implements a balloon driver, which manages the domain's memory usage using exactly this principle. The network components are virtualized

by Xen for providing virtual network interfaces (VIF) to the VMs and these VIFs are logically attached to the virtual firewall-router (VFR) [6]. The host domain bears the responsibility of creation, insertion and removal of rules for identifying a pattern and performing associated action, much like a trigger event.

The guest OS is not aware of this underlying structure and performs only queuing of transmit and receive buffer descriptor in the I/O descriptor rings. Based on the information in these buffer descriptors Xen determines the order and destination of packet transmission. Packet reception is facilitated by checking the VIFs in the packet header and if an appropriate match is not found, the packet is dropped. The disks (IDE and SCSI) provide direct access only to the dom0 and hence virtualization of these components forms the virtual block devices (VBDs) [6], which are created and configured by control management software running within dom0.

Similar to the network functionality, the guest OS queues the disk access requests based on dynamic priority scheduling, but access to the disk is determined by the associated control information specified for the disk. These features provide an image of an ordinary SCSI disk to the guest OS and accessing a memory location is simplified to calculating the original address from a base address and an offset. The future work performed in the virtualization of these subsystems will pave way to enhanced I/O performance and aid in the overall performance of unmodified guests.

2.4 Summary

Based on the concepts discussed in this chapter, the more commonly known definition of virtualization can be expanded to include that it is an abstraction layer between two software layers as well. The multitude of virtualization types available for improving the effective usage of every component of a computing infrastructure shows the distance the technology has come, invoking industrial research contributions to open source projects.

We have also discussed about the challenges faced in virtualizing computing resources such as CPU, memory, I/O, network, application and the solutions implemented by the vendors in

their products, thus effectively paving a path for further exploration on such issues. The discussion on server sprawl problem and consolidation as a solution shed light on the VM manageability and migration troubles which can rise in the control plane. Xen hypervisor's architecture has been looked upon briefly following the challenges in the VM environment to be expected by a VMM. Further details on the schedulers which are a major component of a hypervisor is dealt in the following chapter.

Chapter 3 - Analysis and performance evaluation of CPU scheduling algorithms in Xen hypervisor

3.1 Related work

Cherkasova et al. in [18] have compared three major weight-based schedulers in Xen. They have discussed about the workload imposed on the underlying CPU schedulers by examining the I/O model and posing challenges to that model. The paper emphasizes on the impact of the choice of scheduler and its parameters on application performance. Analysis on the CPU allocation errors has also been discussed based on a simple allocation test benchmark. Though a comprehensive review of the schedulers has been provided which can assist in performance tuning, I/O based comparative analysis of the schedulers, independent of the parameters can put the inherent properties of the scheduler to test.

Latency-sensitive VMs and responding to I/O in such VMs by inspecting I/O pending status has been discussed in [19] by Kim et al. Their work involves a new mechanism for handling I/O centric VMs, while taking into account the shortcomings of the Credit scheduler.

CPU utilization in Xen has been discussed in [20] by Varma et al. This paper contains observations made by running maximum number of homogeneous virtual machines and hosting virtual machines with different sizes of memory allocation. The emphasis has been placed on the best ways to accommodate VMs for optimal CPU utilization, and inferred that approach from the perspective of the mechanism for allocation of resources can lead to more sustainable solutions.

Babu et al. in [37] compare and evaluate the system performance and assess the framework execution of three essential hypervisors namely, Xen-PV for Para Virtualization, OpenVZ for

Container Virtualization and XenServer for Full virtualization. Performance of these virtualization platforms are assessed in view of different tests based on the benchmarking suites. Chi et al. in [38] present a comprehensive experimental study on performance interference of different combinations of benchmarks. They have observed that virtual CPU floating overhead between multiple physical CPUs, and the control transitions between the hypervisor and VMs are the major causes for performance interference. They further characterize these interference effects at both the application-level and VM-level for their interference prediction framework.

3.2 CPU Scheduling In Xen

The Xen hypervisor, since its inception, has employed a number of scheduling algorithms namely Borrowed Virtual Time [8], Atropos [9], simple Earliest Deadline First [14], ARINC 653 [7], Credit and most recently, Credit2 [10]. Scheduling parameters of individual domains can be adjusted by management software running in Domain0. After the architectural change brought by Xen 3.0, the new model of I/O operations increased the complexity of CPU resource usage. The earlier algorithms such as Borrowed Virtual Time and Atropos have been phased out due to the I/O structural change and the lack of non work-conserving mode (NWC) [11], which means that the running domain is able to exceed its CPU share in the absence of competition from other domains.

3.2.1 Borrowed Virtual Time (BVT)

BVT is a weighted or proportional fair-share scheduler based on the concept of virtual time, dispatching the runnable VM with the earliest effective virtual time first [15]. This concept of virtual time allows latency-sensitive or I/O intensive domains to "warp" back in virtual time by temporarily promoting themselves a higher scheduling priority. The amount of virtual time equivalent which was spent for warping is compensated by depleting the domain's CPU allocation in the future. Minimum Charging Unit (MCU) [15], which is derived from the frequency of clock interrupts, is used as basic unit of accounting for a running domain. The decision for a context switch is made based on the parameter context switch allowance 'C',

a multiple of mcu, which denotes the actual time comparable with another VM competing for the CPU. The proportional fair-sharing is implemented by allocating each domain a share of the CPU based on the weights configured in the domains. Optimal fairness is ensured by BVT in allocation of CPU since the error value cannot exceed the sum of context switch allowance 'C' and one 'MCU' [15]. BVT can operate only in work conserving (WC) mode [18] which means that a domain cannot exceed its CPU share even if a portion of the CPU remains unused by other domains. This is one of the major drawbacks of BVT scheduler which severely restricts CPU utilization and hence, BVT was gradually phased out in the later versions of Xen development.

3.2.2 Atropos

The Atropos scheduler allocates shares on the basis of an application dependent period. The share of the processor received by each domain is determined by the slice s and period p which together represent the processor bandwidth of that particular domain. Each domain is guaranteed to receive a minimum of 's' ticks of CPU time and if available, several slices of CPU time in each period of length 'p'. A domain is allocated a 'slack' CPU time, only if it is configured to receive that extra time as indicated by the Boolean parameter 'x'. To ensure timely servicing of I/O from domains, a latency hint 'l' is specified as one of the parameters for differentiating I/O intensive or latency-sensitive domains. The Atropos scheduler provides an additional degree of freedom by enabling the user to control both the absolute share of the domains and the frequency at which they are to be scheduled. A domain, much like a process in operating systems, tends to be in any of the five states of a process cycle. Say if the domain's present state is on a scheduler queue, then based on the deadline 'd', which denotes the end of current period 'p' for the domain, a value 'r' is calculated in real-time and passed to the scheduler as the time remaining for the domain in that current period. Queues 'Qr' and 'Qw' [16], representing the list of ready-to-run and waiting domains respectively, are sorted according to deadline 'd', in addition to another queue of blocked domains. The values dx and px denote the deadline and period of the domains at the head

of the respective queues. The Atropos scheduler due to its significant overhead and inability to balance loads in a SMP has been replaced by more sophisticated algorithms.

3.2.3 ARINC 653

Avionics Application Standard Software Interface [14] is a software specification which partitions time and space to allow multi-level application hosting on the hardware. ARINC 653 is based on the standard Real Time Operating System (RTOS) interface for partitioning of computer resources in the time and space domains. The resulting software environment is capable of handling isolated domains of applications, executing independently of each other. The partitioning of memory into sub-ranges for creating multiple application levels also leads to a spatial and temporal isolation of the domains, thus maintaining them under the impression of complete possession of the underlying hardware.

This methodology of domain isolation, wherein each independent block is called a 'partition' [14], provides ARINC 653 with the capacity to deterministically schedule the Xen domains. The CPU time is allocated in the form of 'credits' in frames which are consumed by the domains. When the domains have reached their credit limit for a given frame they are considered as 'out of credits' or 'expired' and correspondingly placed in a wait queue called 'expiry list'. When the credits are available via the next frame the domains can resume executing again. In case, a domain is made to yield the CPU or if a domain is blocked, prior to consumption of the allocated credits in a given major frame, execution for these domains is resumed later in the major frame.

On the event of completion of a major frame, the domains in the expired list are restored to their preset credit allocations. These preset values and other parameters of the ARINC 653 are configurable from within the dom0 or the driver domain. The parameters to be set include run time for each domain and the major frame for the schedule. Calculation of the length of the major frame depends on the condition that all the domains must be able to expend all of their configured credits. Current implementation of the ARINC 653 lacks support for multi-processor environments and hence, preferred mostly for special case usage.

Table 1: Uniprocessor Scheduling algorithms in Xen and their features

Features	BVT	Atropos	ARINC 653
CPU sharing scheme	Proportional fair share	Absolute shares with option for allocating slack time share	Each domain receives a fixed , dedicated time slot or frame
Parameters and their Description	**ctx allow [global] -** the context switch allowance is the minimum time that a scheduled domain will be allowed to run before being preempted **Domain –** the domain for which the scheduler parameters are set or retrieved **mcuadv** - the MCU (Minimum Charging Unit) advance determines the proportional share of the CPU that a domain receives. Inversely proportional to a domain's sharing weight. **warp** - the amount of `virtual time' the domain is allowed to warp backwards **warpl** - the warp limit is the maximum time a domain can run warped for **warpu** - the unwarp requirement is the minimum time a domain must run unwarped for before it can warp again	**Domain –** the domain for which the scheduler parameters are set or retrieved **Period -** The regular time interval during which a domain is guaranteed to receive its allocation of CPU time. **Slice** - The length of time per period that a domain is guaranteed to run for **Latency** - The latency hint is used to control how soon after waking up a domain it should be scheduled. **xtratime** - boolean flag that specifies whether a domain should be allowed a share of the system slack time	Set **maxcpus=1** on Xen boot command line along with sched=arinc653 Update domain configurations to use only 1 PCPU for all domains and only 1 VCPU per domain Create domains as usual and call the arinc653_schedule_set API with an ARINC653 scheduler structure **Major Frame:** Peridiocally repeating scheduling frame of fixed duration **Minor Frame:** Divisions within the major frame **Timeslice**: The timeslice a VCPU receives before being preempted to run another

Features	BVT	Atropos	ARINC 653
Work- conserving/Non work-conserving modes	WC	WC/NWC	WC
Handling of I/O intensive or latency-sensitive domains	Warping (borrows future CPU time and gains scheduling priority)	Latency hint	Minor frame are selected to meet all the periods, runtimes and deadline constraints of the domains
Real-time scheduling decisions	No	Yes	No
Parameters passed in real-time	N/A	Slack time, Domain deadline, remaining time	N/A
Preferable for	Non -SMP hosts, non-latency sensitive applications	Non- SMP hosts, latency sensitive applications	Non- SMP hosts, latency sensitive applications
Special features	Pre-emption only for latency sensitive domains, low overhead	Distribution of CPU slack time	Temporal isolation of domains
Limitations	No NWC mode, no automated load balancing	No automated load balancing, overhead, non-optimal fairness	No support for multi-core processors; cpu usage and execution windows are fixed regardless of whether the domain is idle or not
First usage in Xen Version	2.0,3.0	2.0	4.1

3.2.4 Simple Earliest Deadline First (SEDF)

The SEDF scheduler is a proportional or weighted sharing algorithm which employs real-time information feedback to guarantee CPU time and deal with latency-sensitive domains. The SEDF scheduler may be considered to inherit its structure from the Atropos schedule, from the perspective of parameters and specifications. For example, a combination of the CPU bandwidth parameters denoted by a tuple {Slice (s_i), Period (p_i), Extra-time (x_i)} is used to specify the amount of CPU time requested by the domain. Like Atropos, there is a minimum guarantee of s_i time slices in every period of length p_i and a domain is allocated extra-time if the Boolean flag x_i is set. This tendency of allowing domains to exceed their CPU limits forms the non-work conserving property of SEDF. More importantly, this slack time which is re-distributed as extra-time on demand, follows a best-effort and fair approach. The slack time is a by-product of under-utilized timeslices and is not borrowed or encroached from other domains' CPU time. The optimization of fair allocation is impacted to a great extent by the granularity used in defining the tuple of a domain. Deadline d_i and remaining time r_i of a domain in the current period p_i are passed to the scheduler in real-time [14], thereby, facilitating the scheduler to make real-time decisions based on the status of the domains. The feature where SEDF differs from Atropos is that the SEDF scheduler chooses the domain with the earliest d_i, from the waiting queue, to be scheduled next. Though the response time to latency-sensitive domains is better in SEDF, the lack of load balancing on SMPs proves to be a major drawback, which has forced the Xen developers' community to gradually phase it out in the future versions.

3.2.5 Credit

Credit is a proportional share scheduler in which the automation of global load balancing of virtual CPUs among the physical cores was introduced to the Xen Hypervisor. When a CPU completes execution of the domains assigned to it, the queues of other CPUs are surveyed for any ready-to-run virtual CPUs (VCPUs). This load balancing, to an extent, prevents CPUs from being overwhelmed and promotes optimized CPU utilization. The queue maintained by

each physical CPU is sorted based on the priority of the VCPU and the priority may either be OVER or UNDER [17].

OVER ***:*** the initial allocation of the VCPU, based on proportional share scheduling, has been exceeded;

UNDER : the initial allocation for the VCPU is yet to be consumed completely.

Every 10ms, or for every scheduler tick, the credits of the domain are decremented and when a VCPU has consumed all of its allocated credits, the value of the credit crosses the zero mark and the state of its priority is converted from UNDER to OVER, and prevented from being scheduled. A VCPU or VM has 30ms before it is pre-empted by another VM, unless it has enough credits to stay UNDER. At the end of every 30ms, the credits for each domain are replenished and the domains which were in OVER state are allowed to execute again. Two parameters govern the credit allocation of a domain,

Weight – proportion or percentage of CPU to be allocated;

Cap – limit of extra credit which cannot be exceeded.

Value of Cap set at '0', means there is no limit to the amount of CPU time that a domain can exceed. Non-zero value for Cap limits the amount of CPU a domain receives by restricting the extra credits to that particular percentage. The availability of both work conserving and non-work conserving modes in Credit scheduler makes it compatible with many applications and continues to be the highly used scheduler (also the default scheduler in Xen 4.x series) in virtual servers running on Xen.

3.2.6 Credit2

The revised version of the Credit scheduler is Credit2, which accounts for some of the short-comings of its predecessor. The Credit scheduler, besides its favourable characteristics, has a few problems like less than average performance in handling I/O intensive domains. Lack of fairness in scheduling latency-sensitive or interactive domains (network-based applications) and hyper-thread dependency issues have raised the cause for the Xen open-source community to develop another version which tries to solve these problems. The Reset condition has been integrated into the core of the algorithm. Some features of the Credit2 [17] are:

- Initial Credits for all domains start at a fixed value
- Consumption of credits at different rates, proportional to the weight of the domain
- VCPUs are inserted into runqueue based on credits
- The Reset condition (back to initial credit value) is invoked when the credits of the VCPU at the head of the runqueue reaches 0 or goes negative

The Credit2 scheduler is still in the experimental stage but our performance tests show improvements in scheduling fairness and I/O handling over the Credit scheduler.

Table 2: Multiprocessor scheduling algorithms in Xen and their features

Features	SEDF	Credit	Credit2
CPU sharing scheme	Weighted sharing with real-time support	Proportional fair share	Proportional fair share
Parameters and their Description	**Domain –** the domain for which the scheduler parameters are set or retrieved **period/slice -** EDF scheduling parameters, same as in Atropos **latency-hint -** the scaled period in case the domain is doing heavy I/O **extra** - a flag which controls whether the domain can run in extra-time **weight** - mutually exclusive with period/slice and specifies another way of	**Domain –** the domain for which the scheduler parameters are set or retrieved **Weight** – determines the CPU share the domain will be allowed to use **Cap** – optional limit on the amount of CPU time which a domain should not exceed **CPUpool** – restrict output to domains in particular CPU pool **ratelimit** – limits the number of schedules per second	**Domain –** the domain for which the scheduler parameters are set or retrieved **Weight** – determines the CPU share the domain will be allowed to use **Cap** – optional limit on the amount of CPU time which a domain should not exceed **CPUpool** – restrict output to domains in particular CPU pool **ratelimit** – limits the number of schedules per second

	setting a domain's cpu slice **CPUpool** – restrict output to domains in particular CPU pool	**tslice –** determines the time to allow a domain to run without pre-empting. Should be higher than the ratelimit**.**	**tslice –** determines the time to allow a domain to run without pre-empting. Should be higher than the ratelimit**.**
Work- conserving/Non work-conserving modes	WC/NWC	WC/NWC	WC/NWC
Handling of I/O intensive or latency-sensitive domains	Latency hint	Ratelimit	Ratelimit
Real-time scheduling decisions	Yes	No	No
Parameters passed in real-time	Deadline of a domain, remaining time	N/A	N/A
Preferable for	SMP hosts, real-time applications	SMP hosts, latency sensitive applications	SMP hosts, latency sensitive applications
Special features	Time guarantee (real-time priority pre-emption)	Automatic global load balancing of VCPUs on multiple physical CPUs	Automatic global load balancing of VCPUs on multiple physical CPUs
Limitations	No automated load balancing, overhead, non-optimal fairness	Inferior to Credit2 scheduler in handling latency sensitive workloads	Needs improvements on handling I/O intensive domains
First usage in Xen Version	3.0	4.1	4.2

3.3 Performance Benchmarking and Analysis

In this section we describe the performance benchmarking environment and parameter setting done for comparison between the scheduling algorithms in Xen. The version of Xen hypervisor under study is Xen 4.4.1 which employs Credit, SEDF and Credit2 as the main scheduling algorithms. The hardware specifications of the system, used for conducting the experiments, are as follows: Intel Core i3-370M processor with two cores of 2.4 GHz each, 64-bit architecture, 3GB RAM and 320GB hard disk.

The driver domain ran Ubuntu 13.04 with linux kernel 3.8.0-19 and the user domain ran Ubuntu 12.04 with linux kernel 3.5.0-23 on Xen 4.4.1 stable release. The test suites used to simulate VM workloads in this experiment are as follows:

IBS - the Isolation benchmark suite [9] which incorporates a CPU intensive test capable of creating simulated workloads by performing a series of integer and floating point calculations. The intense computational operations performed on the CPU resemble compute-intensive applications such as HD video playback or 3D graphics rendering.

IOzone - a file system benchmark [10] which allows performing of multiple Disk-I/O operations and monitor the rate of read/write operations. The sequential and random read/write operations collectively induce stress on the file system.

Netperf - a network monitoring tool [11], aids in measuring low-level network performance by exchanging bulk files with a remote host.

Sysbench - a performance benchmark tool [35] which includes an OLTP test for simulating database server transactions from a Mysql database.

In Xen architecture, the driver domain or dom0 performs the I/O operations on behalf of the guest domain or domU, which creates a situation where in, the dom0 can come under high stress due to many concurrent I/O operations arriving to be serviced. Also, scheduling fairness implies that the domUs should be able to obtain a near-native performance from the underlying hardware.

Table 3: Performance Benchmarking Parameters

	Domain 0	**Domain U**
Kernel	Ubuntu 13.04 - 3.8.0-19	Ubuntu 12.04 - 3.5.0-23
Memory	1.49 GB	1.49 GB
Credit sched Weight	256	256
IOZone	File size : 512 MB Record length : 1 MB	File size: 512 MB Record length : 1 MB
Netperf socket size	Send : 16384 bytes Receive : 87380 bytes	Send : 16384 bytes Receive : 87380 bytes
Sysbench	Table size : 1000 to 1000000 rows	Table size : 1000 to 1000000 rows
	Num. of threads : 4	Num. of threads : 4

To ensure that a scheduling algorithm is capable of providing near-native I/O support and scheduling fairness, we organized a series of experiments coupling I/O operations with CPU intensive operations and high bandwidth networking to simulate a virtualized server environment and measure the performance delivered by the schedulers under these conditions.

Throughout the experiments the schedulers are retained in their default parameter settings such as 1:1 domain weight ratio and Cap = 0 for the Credit schedulers.

3.3.1 Disk I/O benchmarking using IOzone

We employ IOzone test suite, which performs a series [10] of sequential read/write (S-Read/S-Write) and random read/write (R-Read/R-Write) operations, followed by the measurement of the read/write rate in KBytes per second. We choose a combination of 512Mb file size with 1 Mb record length for avoiding a system freeze, when hosting multiple guests while also keeping in mind, the cumulative operation time of the experiment.

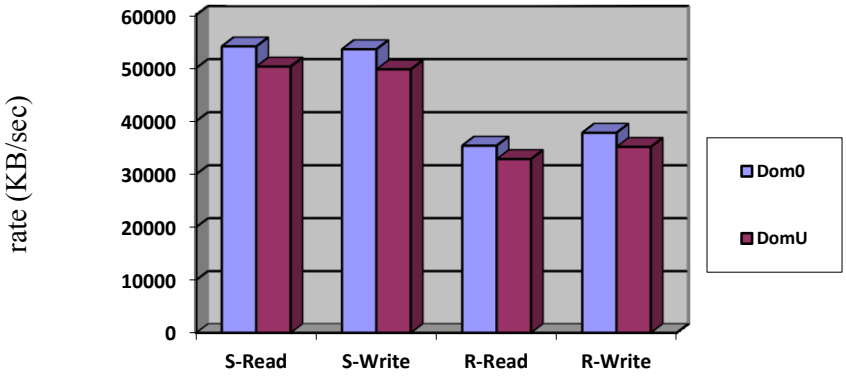
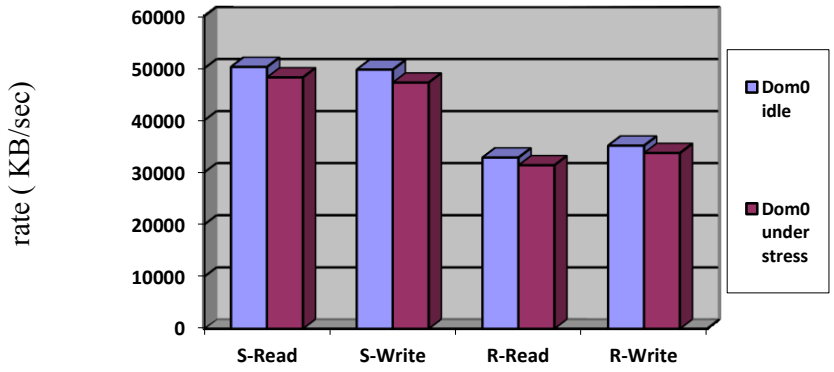


Figure 17: Credit Scheduler Disk I/O performance

In order to avoid files being written or read from the cache memory or RAM, IOzone is tuned



to perform only Direct I/O operations, which ensures that only the physical disk is used in the file operations. Figure 17 shows credit scheduler's dom0 disk I/O performance when

domU is idle and domU's disk I/O performance when dom0 is idle. In Figure 18, the performance by a credit scheduler domU when dom0 is performing heavy CPU operations facilitated by IBS's CPU intensive test. In the first case, the domU is able to achieve 93% of dom0's I/O performance and when dom0 is under stress, domU's I/O execution rate degrades by 4%-5% of its earlier performance when dom0 was idle.

In Figure 19, the disk I/O performance of SEDF scheduler's dom0 (domU idle) and domU (dom0 idle) is shown. Under the SEDF's scheduling the domU was able to run I/O operations at 97% of dom0's I/O execution rate. This might seem better than the Credit scheduler's fairness but there is a tradeoff in overall I/O performance, where the Credit scheduler has an edge and the SEDF scheduler lags due to the overhead induced by its real-time processes.

Figure 18: Credit scheduler domU disk I/O performance

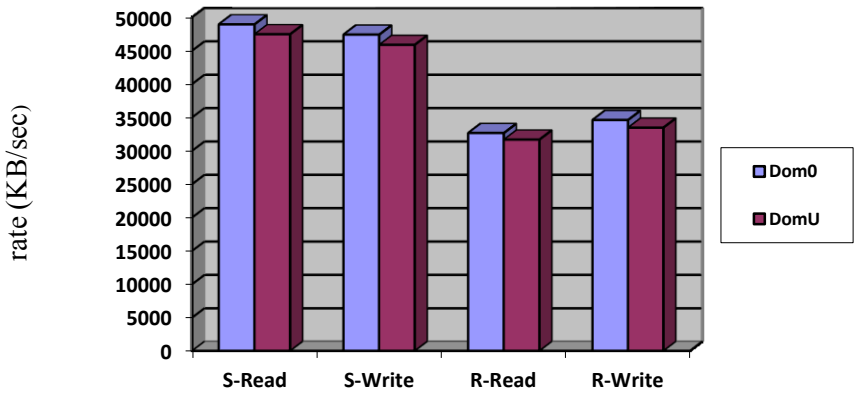


Figure 19: SEDF scheduler disk I/O performance

When the dom0 is subjected to heavy CPU operations in the SEDF scheduler, the domU's disk I/O performance is found to be drastically affected. As illustrated in Figure 20, the disk I/O performance of domU, when dom0 is under stress, degrades by 41% - 45% of its performance when dom0 was idle.

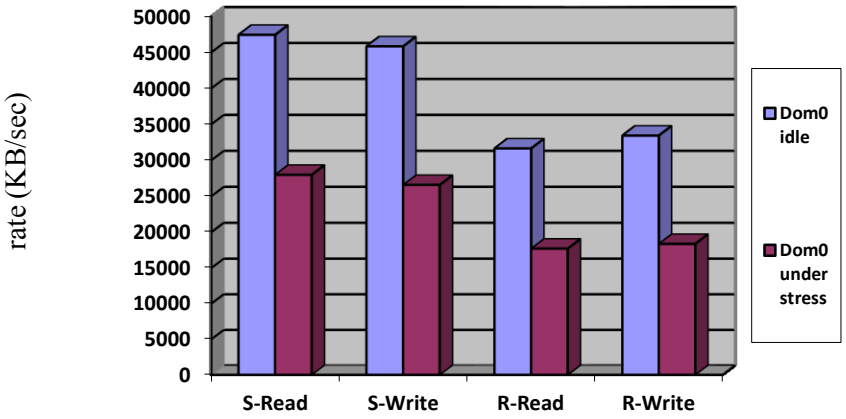


Figure 20: SEDF scheduler domU disk I/O performance

Figure 21 shows the disk I/O performance of the Credit2 scheduler; dom0 or domU is idle, when the other domain is executing I/O operations. With Credit2 scheduler the domU was able to achieve from 94% up to 99% of dom0's I/O execution rate. These readings show a marked improvement in handling guest domain I/O requests from Credit scheduler and the overall I/O execution rate is also justified.

The handling of I/O from domU by dom0 when under stress in Credit2 scheduler is better when compared to Credit and SEDF. As shown by Figure 22, the disk I/O performance by domU when dom0 is performing CPU intensive operations produces results which show that Credit2 scheduler is capable of providing near-native I/O support in a compute intensive environment also. The results show that the domU is capable of achieving a service degradation of only 4%-6% and given the superior overall I/O execution rate, this fall in read/write rate is only of negligible magnitude.

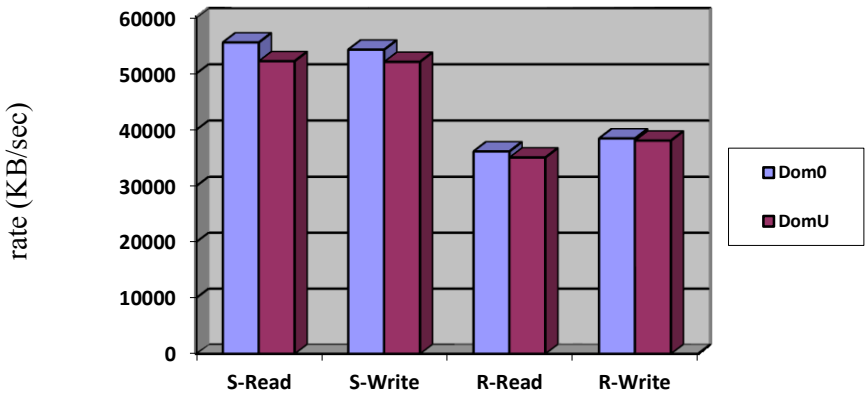


Figure 21: Credit2 scheduler disk I/O performance

Disk read/write access operations are not the only source of I/O in a server. Network communication creates one of the major influxes of I/O operations in a server environment like data center. Incoming communication requires CPU to process the packets and decide on an acknowledgement scheme based on the information received. Web application servers and database servers form the majority of the servers in the cloud infrastructure.

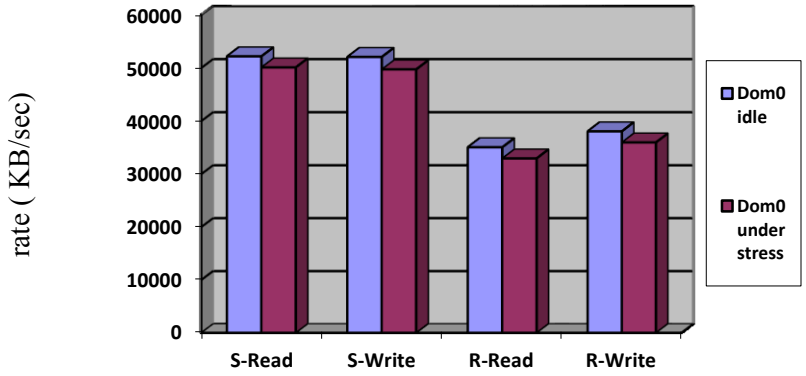


Figure 22: Credit2 scheduler domU disk I/O performance

VM environments involving a combination of disk I/O operations and network I/O processing are commonly found in a typical server. These operations are not mutually exclusive when it

comes to a virtual server as heterogeneous consolidation is beneficial to the infrastructure provider and hence, heavily preferred.

Table 4: Disk I/O – Credit, SEDF and Credit2 performance comparison

IOzone	Credit	SEDF	Credit2
Near Native performance (% of Dom0)	93%	97%	94% - 99%
DomU Disk Throughput depreciation when Dom0 under stress	4% - 5%	41% - 45%	4% - 6%

3.3.2 Network performance benchmarking using Netperf

The following experiments are conducted by allowing Netperf to run a client on the domU and perform TCP stream tests via bulk file transfers with a remote host connected by a 100Mbps ethernet link. The file transfer is done in increments of message size from 1 MB up to 4MB and the throughput is measured in Mbits per second. The send socket size is 16384 bytes and receive socket size is 87380 bytes (Netperf default socket setting).

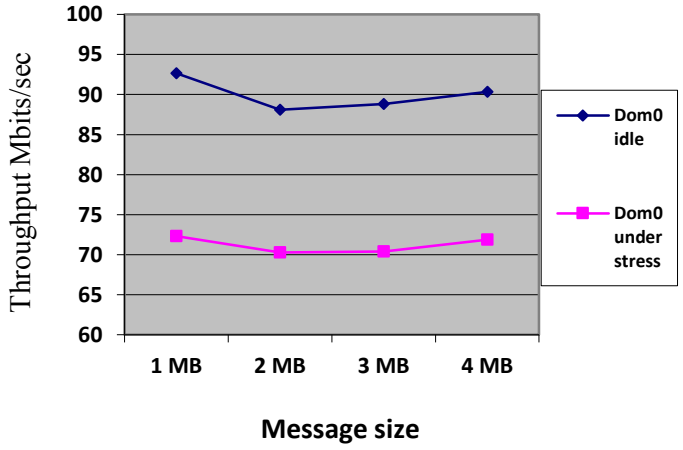


Figure 23: Credit scheduler network performance

Figure 23 shows the network performance of domU when dom0 is kept idle and when dom0 is made to perform CPU intensive operations facilitated by IBS CPU stress tests.

The results exhibit a clear drop in throughput from an average of 87.85 Mb/s to 72.71 Mb/s. The Credit scheduler has an average performance when the CPU is under contention and an above average performance when CPU is not performing any compute intensive tasks.

When SEDF is put under the same test, the results show that the real-time scheduler fails to handle I/O tasks promptly when the CPU is placed under stress. Figure 24 shows the performance of SEDF's bandwidth intensive test and we can observe a fall in throughput from an average of 89.30 Mb/s to 63.55 Mb/s when CPU is under contention.

The Credit2 scheduler exhibits significant improvement from Credit scheduler in network performance tests. As shown in Figure 25, the average throughput of 91.05 Mb/s, when dom0 was idle, dropped to an average of 79.15 Mb/s, when CPU is under stress, which is better than the Credit scheduler's 72.71 Mb/s.

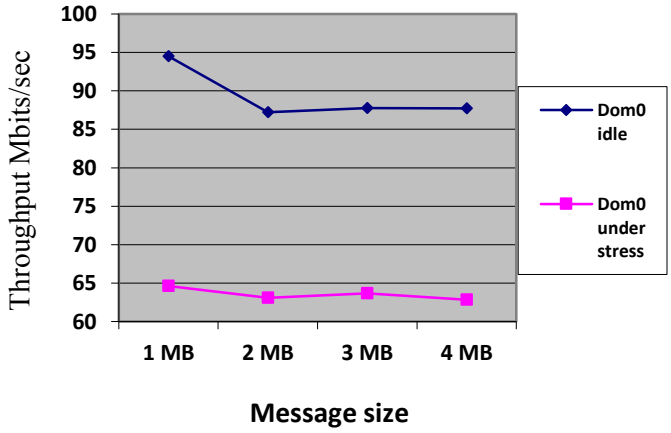


Figure 24: SEDF scheduler network performance

The consolidation of I/O intensive tasks and network bandwidth intensive tasks by running IOzone in dom0 and Netperf in domU is created. The IOzone suite reads/writes files of size 512 MB on the disk and Netperf performs file transfers of sizes ranging from 1MB to 4MB.

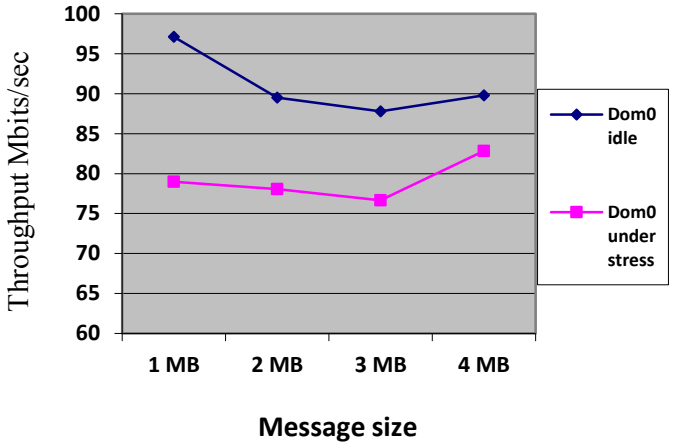


Figure 25: Credit2 scheduler network performance

When the three schedulers were tested in this scenario as shown in Figure 26 Credit2 performed the best of the three with an average throughput of 84.57 Mb/s, while Credit gave an average throughput of 71.63Mb/s and SEDF yielded an average throughput of 69.65 Mb/s.

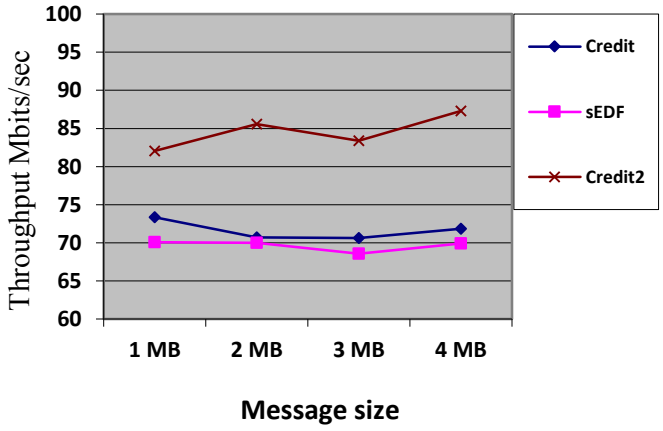


Figure 26: Network performance of domU when dom0 performs heavy disk I/O operations

Table 5: Network I/O – Credit, SEDF and Credit2 performance comparison

Netperf	Credit	SEDF	Credit2
DomU network Throughput when Dom0 idle	87.85 Mbps	89.30 Mbps	91.05 Mbps
DomU network Throughput when Dom0 under CPU stress	72.71 Mbps	63.55 Mbps	79.15 Mbps
DomU network Throughput when Dom0 performs heavy Disk I/O	71.63 Mbps	69.65 Mbps	84.57 Mbps

3.3.3 OLTP database server performance benchmarking using Sysbench

The performance of the schedulers when hosting database server applications can be measured using Sysbench. As a prelude to simulating the database server we create a Mysql test table with rows of data starting 1000 rows and continue scaling up to 1000000 rows.

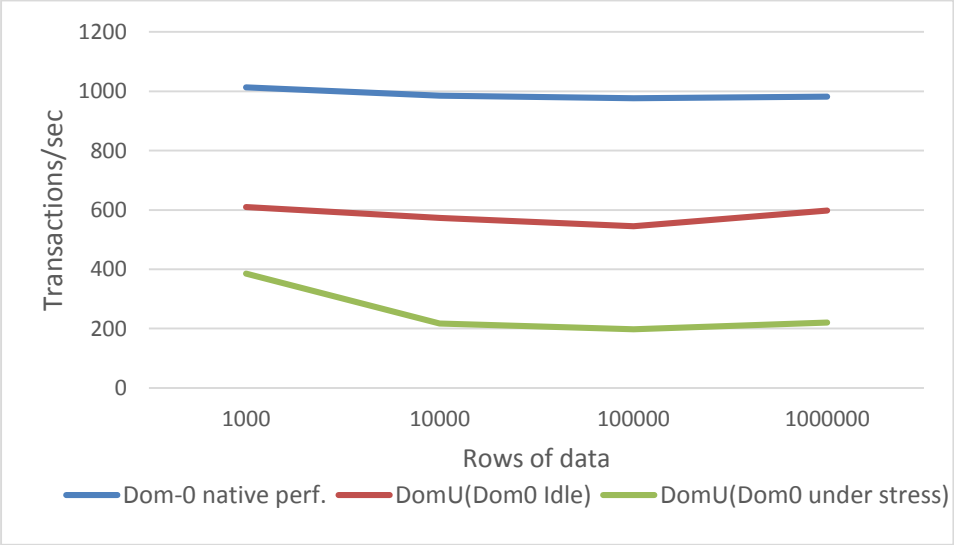


Figure 27 Credit Scheduler Database server performance

As illustrated in Figure 27, the Credit scheduler's number of transactions/second, which is an important merit for seamless database querying, suffers a performance drop when a user Domain is hosting the database server. And when the host domain is placed under computing stress the OLTP transaction rate further degrades due to the unavailability of the CPU for the server's I/O requests.

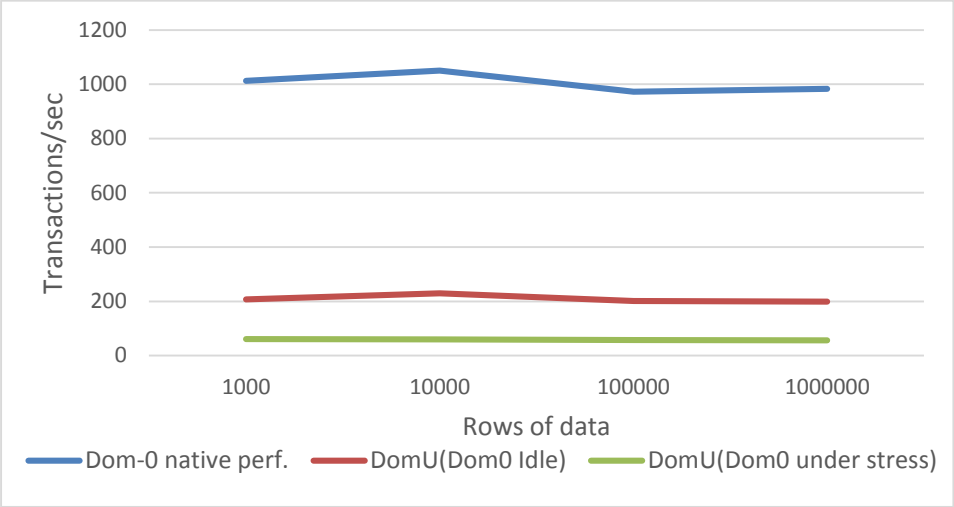


Figure 28 SEDF scheduler database server performance

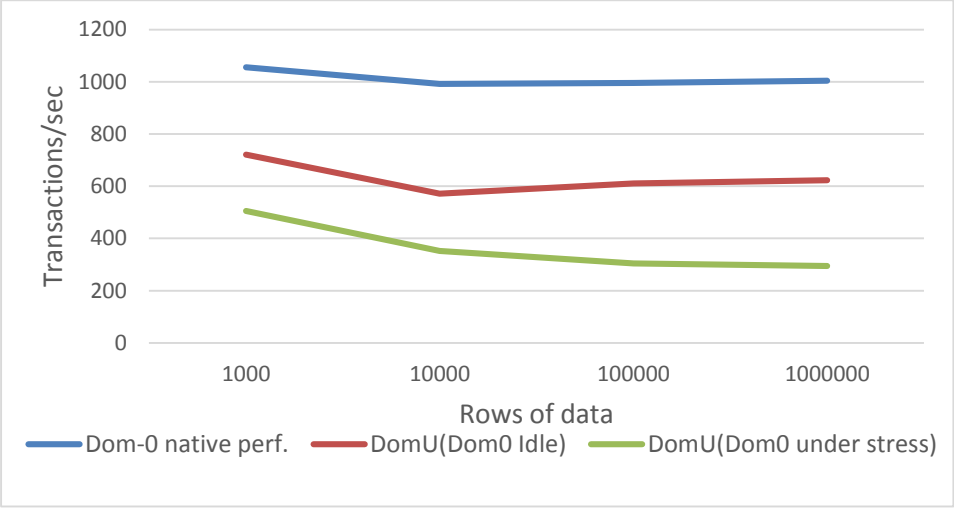


Figure 29 Credit2 scheduler database server performance

Figure 28 and 29 exhibit the database server performance of SEDF and Credit2 schedulers respectively. The SEDF scheduler's domU reduces to almost one-fifth of its native domain's

performance and when CPU is in contention an average of 75% drop in domU's performance is inferred from the benchmarking results. On the other hand the Credit2 scheduler once again comes on top among the three schedulers, by showcasing only minimal loss in the number of transactions executed per second when hosted by the user domain.

3.4 Consolidated server environment:

One of the solutions for server sprawl in a data center is the consolidation [46] of hardware resources promoting optimization, while masking the server architecture from the users by ensuring negligible or unnoticeable deterioration in QoS. Any hypervisor which is claimed to be a candidate for server virtualization has to be qualified to host multiple and multi-faceted server workloads.

Table 6: Consolidated server environment simulation parameters

	Domain 0	**Domain 1**	**Domain 2**
Kernel	Ubuntu 13.04 - 3.8.0-19	Ubuntu 12.04 - 3.5.0-23	Ubuntu 12.04 - 3.5.0-23
Memory	1 GB	1GB	1 GB
Credit sched Weight	256	256	256
IOZone S-Write	File size : 512 MB Record length : 1 MB		
Netperf socket size		Send : 16384 bytes Receive : 87380 bytes	
Sysbench			Table size : 1000 to 1000000 rows
			Num. of threads : 4

While the scale of number of VMs hosted is tied to the hardware's capacity, other aspects such as I/O latency and fairness towards domains can be optimized by choosing the apt scheduler for a given combination of applications.

The consolidated server environment is simulated by hosting two user domains and each of the above discussed performance benchmarks in each of the domains. The experiment is conducted by running IOzone on dom0, Netperf on dom1 and Sysbench Mysql benchmark on dom2. The performance results are observed in the following table.

Table 7: Performance of Xen Schedulers in a consolidated workload environment

	IOzone S-write	**Netperf TCP stream**	**Sysbench MySql**
	KBytes/sec	Mbits/sec	Transactions/sec
Credit	49916	70.48	388.25
SEDF	27011	60.77	262.29
Credit2	51044	75.92	478.05

The extent to which the virtual counterpart of a dedicated server has performed can be gauged by using the Domain0 or host's native performance. As shown in figure 30, the performance indexing based on domain 0 gives a comprehensive picture of the schedulers in a typical server environment.

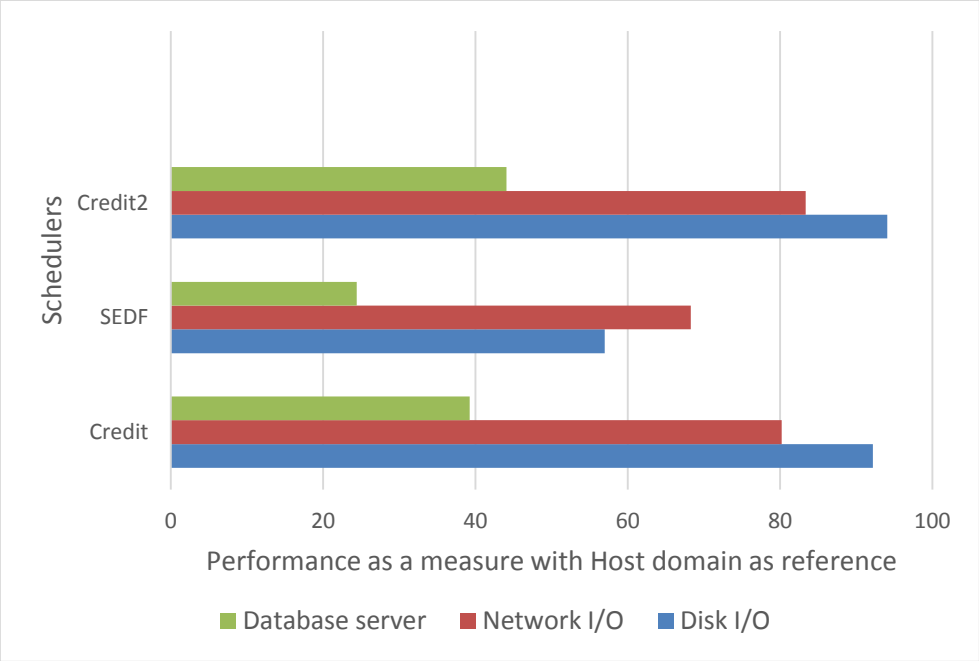


Figure 30 Consolidated workloads – Credit, SEDF and Credit2

The consolidated workloads generate I/O requests on a frequent basis thus causing an I/O bottleneck, which has evidently caused the setback in the schedulers. The results obtained are not far from the observations made, when the CPU as a resource was under contention. This shows that servicing of I/O requests or interrupts is just as important as CPU allocation. The servicing of Disk I/O has not been interfered by the I/O bottleneck and this suggests the consolidation of disk intensive domains with I/O heavy workload servers such as a web-server.

The experiments provide a glimpse of the Credit2 scheduler's exceptional performance in a high frequency I/O environment and the adept I/O handling capacity which was lacking in Credit scheduler. The results obtained collectively form a substantial evidence to the assertion that choice of CPU scheduling algorithm used by Xen hypervisor impacts the performance of the applications hosted on the server.

3.5 Summary

The choice of CPU scheduling algorithm used by Xen hypervisor evidently impacts the performance of the applications hosted on the machine. We analysed the CPU scheduling algorithms incorporated into Xen so far, since its earliest version and conducted exhaustive experiments using simulated workloads on the latest batch of schedulers released by the Xen developers' community.

The most significant inference from the experiments is that servicing of I/O by the driver domain depends heavily on Xen's scheduling capacity. By focussing on the I/O handling tendency of each scheduler provided an insight into the right usage of the hypervisor for workload specific applications. Though varying the scheduler parameters could even out some odds in the scheduler's functionality, true scheduling fairness is achieved by achieving optimal performance on all domains without disproportionate usage of resources.

This chapter mainly dealt with the behaviour of the hypervisor when faced with heavy workloads which challenge the resource usage by the scheduler in charge. Credit and Credit2 have proven to be robust in their handling of heavy I/O workloads such as Disk, bandwidth intensive network and database operations. Credit2, though in experimental phase has been observed to exhibit and deliver the promised improvement in I/O handling over the Credit scheduler. The older SEDF scheduler has been observed to fall behind in contention with both the Credit scheduler versions. ARINC 653 is a special case scheduler built for uniprocessor architectures and hence cannot be compared with the other schedulers in this version of Xen.

Chapter 4 - Optimization of CPU Scheduling for Handling Compute and I/O Intensive Virtual Machines

4.1 Compute-intensive and I/O intensive applications

Every data centre environment has a multi-faceted collection of server varieties such as mail, web, application, FTP, database, compute, media and game servers. In case of virtualized high definition video streaming and compute servers, the CPU-intensive domains will occupy the CPU for a longer time while starving I/O bound domains, since they require more computational time on the processor. During this CPU wait time, the domains yielding the CPU will complete their I/O and move into the ready or runnable queue, waiting for the next CPU cycle. While the domains hold up in the ready queue, the I/O devices are waiting in idle state. And when the CPU-intensive domains complete the CPU burst they are dispatched to an I/O device's queue which has lesser wait times than the CPU queue. The cycle continues when all the I/O-bound domains, which have short CPU bursts, execute rapidly and move back to the I/O queues. And currently, the CPU sits idle and hence the compute-intensive domain is yielded the idle CPU. This creates an I/O bottleneck in the virtual server leading to poor response times which directly affects the quality of service. To understand how the domains behave in such bottle neck conditions, we need to first study the life cycle of a VM and the phases a VM goes through in Xen to observe how they are analogous to process states in operating systems.

A VM or User Domain in Xen can be present in any of the following states:

- Paused – The domain's execution is paused and it still consumes allocated resources such as memory, but will not be eligible for scheduling. Domain enters this state on 'xm pause' and returns to scheduling queue on 'xm unpause'.

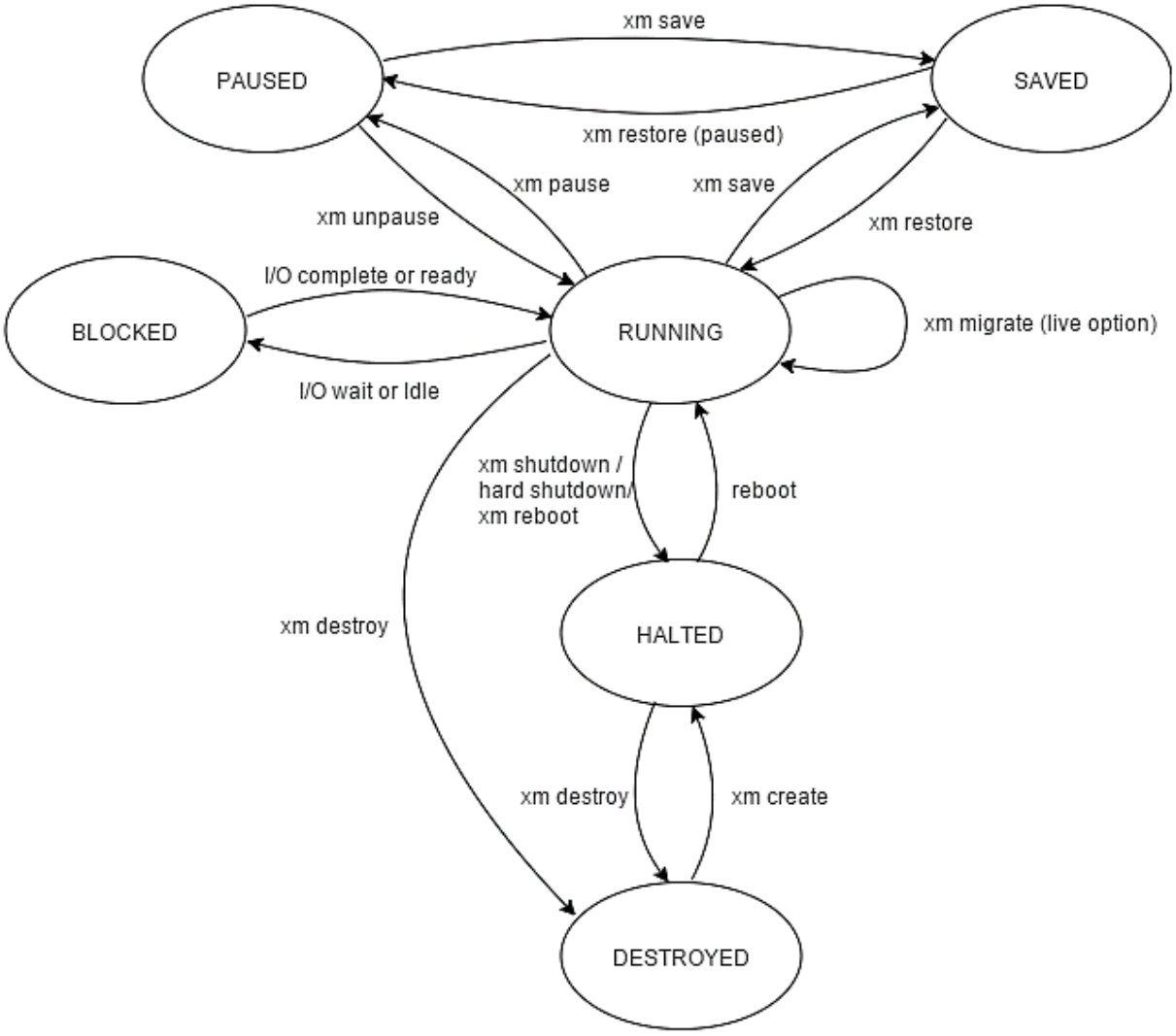


Figure 31: Domain states or VM states in Xen Hypervisor

- Saved – 'xm save' command saves a running domain to a state file for it to be restored later as it was, but TCP timeouts could have severed network conenctions. Saved state domain will no longer be running on the system and the memory allocated for the domain will be released for other domains to use. 'xm restore' restores the domain from the state file.
- Running – Domain is running or runnable on the CPU.
- Blocked – Domain is waiting on an I/O or in idle mode. It is currently not running or runnable.

- Halted / Shutdown – Domain enters this state if ‘xm shutdown’ command is used or on a hard shutdown, terminating all the processes of the OS. ’xm reboot’ command can enter this state momentarily if the domain is configured as such.
- Destroyed – The domain’s processes are killed instantly and the domain id is deleted. ‘xm create’ is required to construct the domain again from the configuration file.

We can see that the VM states from Figure 31 are similar to process states in operating systems and they can be promised better response times by optimizing the scheduler to ensure fair allocation of CPU time.

4.2 Related Work

Xen hypervisor’s VCPU scheduling characteristics have been discussed by Tian et al. in [32] through analysis and extraction of scheduling state changes of VCPUs. They have analyzed some basic properties of the Credit scheduler and the interference caused in I/O performance due to VCPU scheduling. They have observed that the Credit scheduler does not provide VCPU co-scheduling for VMs hosted on SMP chipsets. Furthermore, their inference is that synchronizing VCPUs frequently causes severe degradation of application performance, especially when dealing with high number of VMs and VCPUs per VM.

In [33] Zhong et al. have dealt with lockholder preemption in Xen virtualized environments. Their work encompasses analysis into the effects of lockholder pre-emption on scheduling performance, and a proposal for lock-aware scheduling mechanism through modifications to the Xen platform and Linux. The scheduling algorithm implemented by them shows improvements in way of reduced interference from synchronization and ensures fairness among VCPUs. We empirically evaluate our optimized system, and experimental results demonstrate that the system gain better performance. Though their proposed lockaware scheduling mechanism was aimed for Xen platform, the implementation is limited to paravirtualized systems.

The authors in [34] focus on expanding the current scheduling framework's functionality to support soft real-time workloads in virtualization systems. In this work, they present an enhanced scheduler SRT-Xen which promotes the improvement in soft real-time domain's performance compared to the existing schedulers in Xen. Furthermore their work includes integration of a new real-time friendly scheduling framework and corresponding strategies in Xen, while enhancing the efficiency of management of the VCPU queues. The resulting implementation demonstrates a fair scheduling mechanism for both real-time and non-real-time tasks. They have employed the PESQ (Perceptual Evaluation of Speech Quality) and associated performance benchmarks for the evaluation and comparison of SRT-Xen with existing Xen scheduling schemes.

Scheduling of parallel applications and accompanying challenges has been explored by Ding et al. in [36]. Their work involves performance based enhancement in handling VMs with parallel computational workloads. A parallel schedule algorithm (PSA), which is built upon the structure of the Credit scheduler, has been developed and implemented by them. Their version of the scheduler supports scheduling of the synchronous parallel VCPUs which has been observed to reduce the effect of asynchrony in parallel VCPUs caused by the interrupts generated by I/O bounded workloads. Their evaluation of the parallel algorithm shows improvement in scheduling of the parallel VCPUs with heterogeneous workloads when compared to the original Credit scheduler, co-scheduler and UVF: a demand-based coordinated scheduler.

4.3 VCPU Scheduling Scheme

Our VCPU scheduling policy will be implemented on the credit based scheduler's structure, since credit sharing based on weight is the most efficient method of ensuring fairness. The credit based scheduling scheme has run queue for every PCPU which is populated by the VCPUs depending on priority. The working of the credit scheduler is shown in Figure 32 [45] which includes a scenario involving 5 domains.

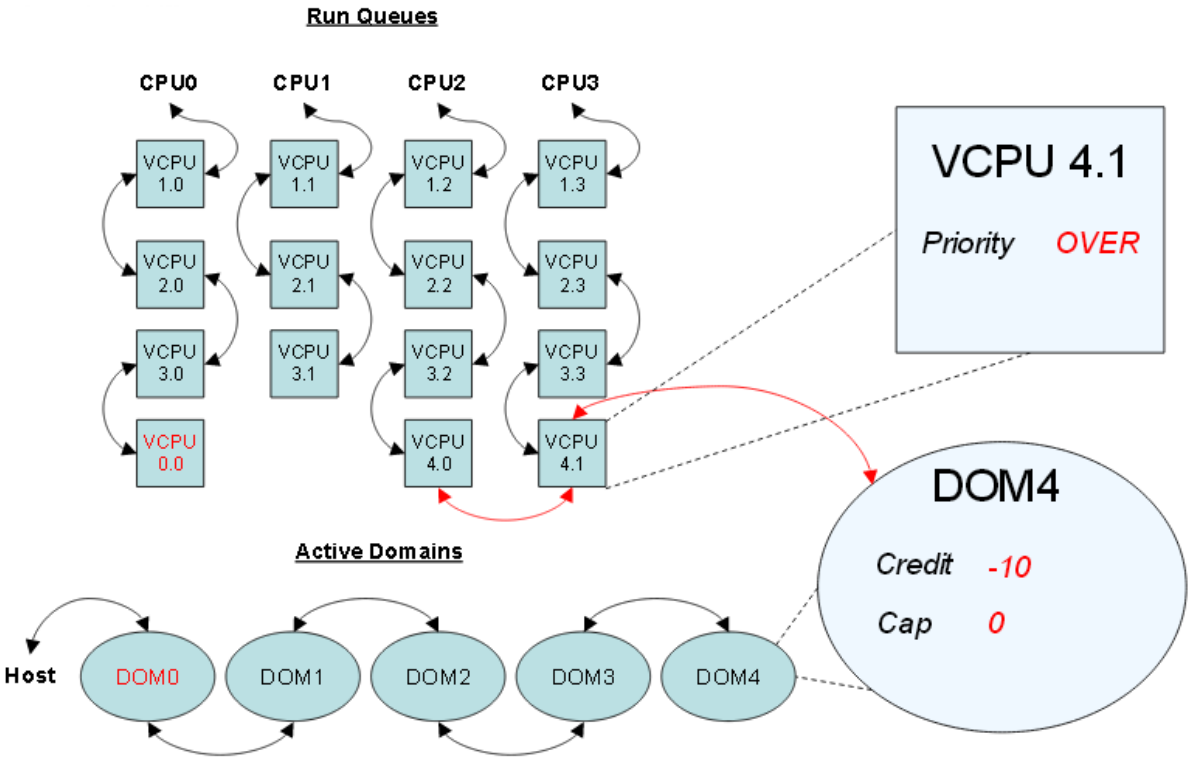


Figure 32 Credit based VCPU scheduling scheme [45]

The credit based VCPU scheduling allocates 30ms as time slice for the VCPUs to burn their credits when the CPU is allocated. But our experiments from the previous chapter indicate this length of time favours compute-intensive workloads rather than I/O intensive ones. While the I/O bound VCPUs wait for the allocation of CPU the next accounting period replenishes the credits, thus further enriching the CPU bound VCPUs. The option to reduce the time quanta to address this problem requires manual intervention at every instance a different workload is received.

We propose a new credit-based VCPU scheduling scheme where in the credit remaining for each VCPU after every accounting plays an important role in making the scheduling decision. The starved VCPUs with the most remaining credits are allowed to supersede others to favour the reduction of the I/O bottleneck in the run queue. Since most I/O bound VCPUs need only 1ms to 10ms to complete their short CPU burst cycles, there will be no adverse effects in CPU-bound domain performance.

Algorithm :

```
enhanced_schedule (current time);
Input parameter        : current time
Output parameter       : next VCPU to run
BEGIN
Descheduling a running VCPU:
IF
 Running VCPU is runnable
THEN
 Insert at end of PCPU run queue;
ELSE
 VCPU is idle and no other VCPUs on PCPU run queue;
END IF

Perform VCPU selection from run queue:
IF
 VCPU's priority is OVER or VCPU_credit less than max_credit;
THEN
 Check other PCPU run queue;
          IF
           Other PCPU run queue empty;
          THEN
           Dequeue from current PCPU run queue;
           Insert VCPU into PCPU with empty run queue;
          ELSE
           Yield PCPU and insert at end of current PCPU run queue;
           Perform VCPU selection from run queue;
ELSE
Update max_credit (VCPU_credit);
Return (VCPU to run next);
END IF

Update max_credit:
IF
 Received VCPU_credit greater than zero;
THEN
max_credit = VCPU_credit;
END IF;
```

The VCPU scheduling scheme proposed here has the potential to introduce context switch overhead because of additional validations in the scheduling decision path of code. Though the context switch overhead will exist the benefits of a higher response time scheduling for I/O bound domains will outweigh the negative aspects of the Enhanced scheduler.

4.4 Scheduler Interface

The Schedulers in Xen are integrated into the microkernel through an interface [27], where mapping of the scheduling functions is facilitated. It is accessible in the Xen's source code which includes a structure with pointers to functions, essential in implementing the functionalities of the scheduler. This interface provides the developer with an abstraction layer to implement and integrate new schedulers with features limited to or superimposing the existing interface.

The development of a new scheduler involves the implementation of an important structure pointing to the new scheduler's functions, thus adding it to a list of existing schedulers. Xen refers to this static array, while invoking the scheduler and it is not essential that every one of the functions available in the structure get implemented for any give scheduler. In the event that one of the functions is not implemented, it should be assigned with a NULL pointer and it will be disregarded. An implementation of a basic scheduler such as Round Robin can be provisioned by setting most of the listed functions to NULL, but the level of performance and lack of sophisticated VM handling make it undesirable.

In any case, not every one of these functions can be set to invalid, for instance the function do_schedule() must always be a valid function since it picks the next VCPU to run. Functions calls are facilitated by a macro which tests for a non-NULL value, if available, or returns a 0 otherwise. So a scheduling scheme which does not implement the do_schedule function will behave erratically and crash.

The function pointed by the do_schedule pointer, receives the time as an input parameter and returns a struct which indicates the next VCPU to run. The return value also has the

measure of CPU time slice allotted to the VCPU to run, provided it will not be pre-empted by a higher priority VCPU. In order to return the VCPU with this function, the scheduler has to keep a record of other VCPUs, which are eligible for a context switch. Whenever a VCPU is started the necessary scheduling parameters are initialized by calling the VCPU initialization function.

```
1  struct scheduler {
2  char _name ; /* scheduler name */
3  char _opt_name ; /* optional handle for the scheduler */
4  unsigned int sched_id ; /* Scheduler ID */
5
6  void (_ init) (void) ;
7
8  int (_ init_domain) (struct domain _) ;
9  void (_ destroy_domain) ( struct domain _) ;
10
11 int (_ init_vcpu ) ( struct vcpu _) ;
12 void (_ destroy_vcpu) ( struct vcpu _) ;
13
14 void (_ sleep ) ( struct vcpu _) ;
15 void (_wake ) ( struct vcpu _) ;
16 void (_ context_saved ) ( struct vcpu _) ;
17
18 struct task_slice (_do_schedule ) ( s_time_t ) ;
19
20 int (_ pick_cpu) ( struct vcpu _) ;
21 int (_ adjust ) ( struct domain _ ,
22 struct xen_domctl_scheduler_op _) ;
23 void (_ dump_settings ) ( void) ;
24 void (_ dump_cpu_state ) ( int ) ;
25
26 void (_ tick_suspend ) ( void) ;
27 void (_ tick_resume ) ( void) ;
28  } ;
```

Xen scheduler development interface [27]

The scheduler must be able to utilize the VCPU initialization function call, to stay informed of the available VCPUs in the system and their accessibility at any given time. The schedule.c

file in the Xen source code contains any additional supporting functions defined in the scheduler interface. Any preliminary operations essential to be performed, before calling any scheduler-oriented functions, are handled by the funcions available in this piece of code. The function schedule() defined in schedule.c starts by descheduling the current domain on the CPU and then proceeds by calling the associated scheduler functions. The return value from the scheduler function contains a new task to be scheduled along with the run time available. Then end of the time quantum is triggered by a timer put into motion by schedule.c at the beginning of scheduling of the current task. The triggering operation is reset when the next task's time quantum initiates.

As shown in the table before, the scheduler interface has four non optional fields one of which is he name field containing a user facing name for the scheduler's identification. The opt_name field can hold an abbreviated form of the scheduler's name in order to choose the scheduler during the boot time of the hypervisor via "sched" option. Examples of this field are "sched=sedf" and "sched=credit2" for switching to the SEDF and Credit2 schedulers respectively. The sched_priv pointer in the VPCU's structure is used to store the private data from the scheduler pertaining to that VCPU. The memory requirement for the VCPU structure is allocated by init_vcpu followed by data initialization. Once the VCPU is decided to be discarded the associated memory has to be deallocated by the scheduler while calling the destroy_vcpu function.

The physical CPU or PCPU also has a domain structure associated with it, which contains a VCPU data structure for maintaining the list of VCPUs owned by that particular domain. Pinning of VCPUs to PCPUs ensures that the domain receives uninterrupted CPU time. In such privileged cases the scheduler has to differentiate the domain's VCPUs for expedited handling. The degree of fairness ensured by the scheduler, particularly in case of uneven weight configurations between domains, depends on identifying the owner domains. In some cases involving concurrent or parallel computing applications, all the VCPUs of a domain may be required to run at the same speed and time interval. In the event that the user decides to

dedicate one VCPU per process then all that the scheduler will be responsible is the assignment of tasks to VCPUs.

4.5 Results and Analysis

The experimental setup used for comparative analysis of the Credit, SEDF and Credit2 algorithms is employed again to aid in the evaluation of the Enhanced Scheduler. The Scheduler is tested on handing Disk I/O for observing how it compares to the host domain's disk I/O handling performance.

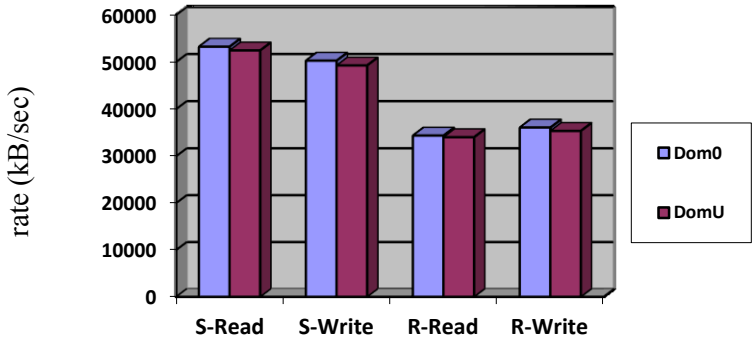


Figure 33 Enhanced scheduler Disk I/O performance

Figure 34 Comparison of DomU's Disk I/O performance when Dom0 idle

Figure 33 shows that Enhanced scheduler's DomU reaches 98%-99% near-native disk I/O performance when Dom0 is idle, which exhibits the scheduling fairness among the VCPUs promoted by the new scheduling scheme. A clear improvement over the Credit scheduler which could only reach a maximum of 93% of near-native disk I/O performance.

The comparison shown in Figure 34 shows that the enhanced scheduler has a near native performance index on par with Credit2 and evidently better than the Credit and SEDF scheduler.

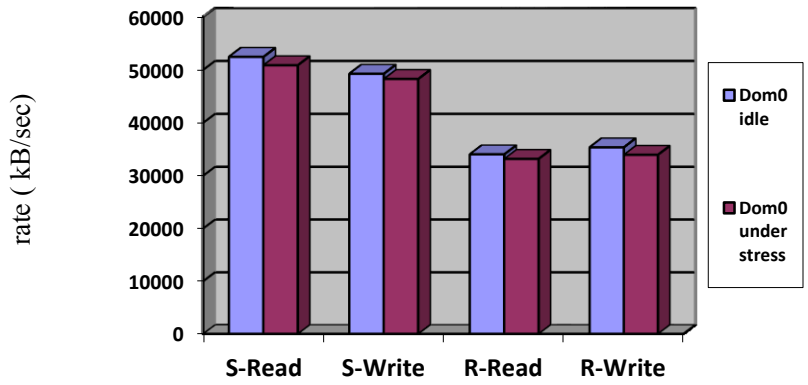


Figure 35 Enhanced Scheduler DomU disk I/O performance

The Enhanced Scheduler suffers a 2%-4% degradation in disk I/O performance when Dom0 is performing a compute-intensive workload as shown in Figure 35.

The comparative display of degradation in S-write performance is shown in Figure 36, wherein the enhanced scheduler is the least to deviate from its optimal performance followed by Credit2, Credit and finally SEDF.

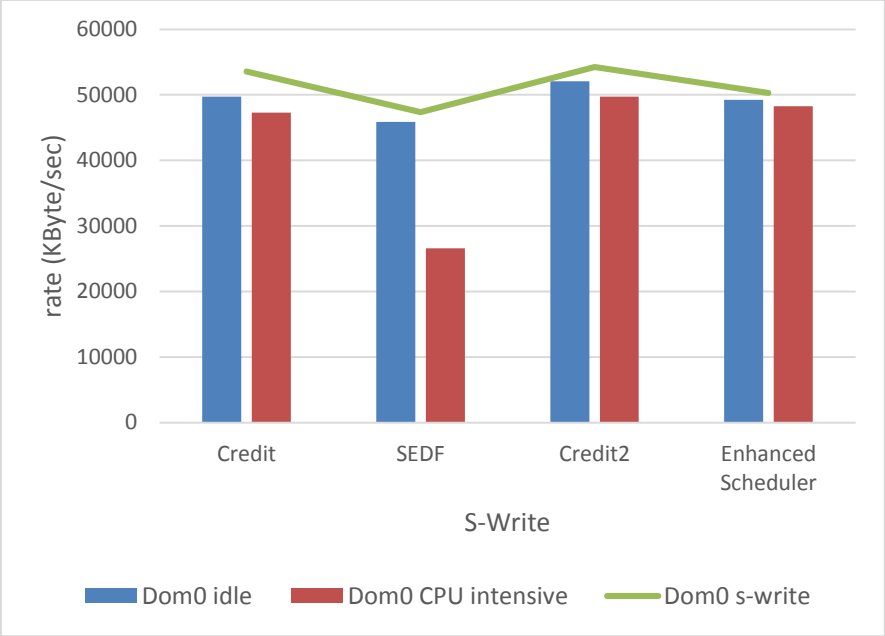


Figure 36 Comparison of degradation in S-Write rate of DomU

The Enhanced Scheduler has to be evaluated in the handling of bandwidth intensive network I/O. The simulated workloads will generate a compute-intensive stress on Dom0 and the Netperf benchmark transfers files ranging from 1MB to 4MB from a client running on DomU to a remote host.

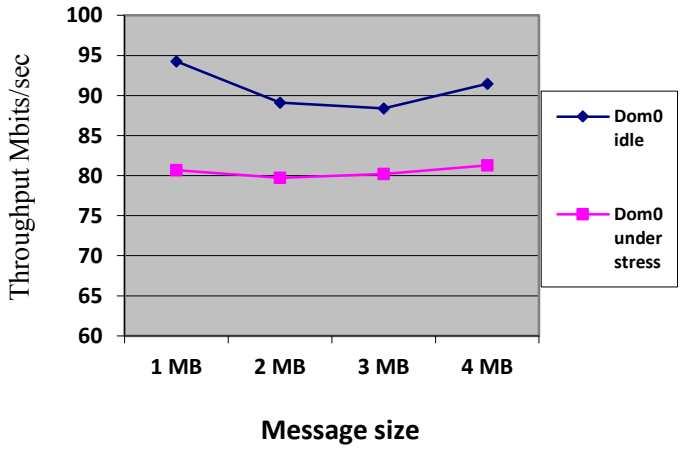


Figure 37 Enhanced Scheduler Network Performance

The results from network I/O handling benchmark as shown in figure 37 indicate a performance drop in throughput from an average of 90.81 Mb/s to 80.47 Mb/s when Dom0 is under CPU stress.

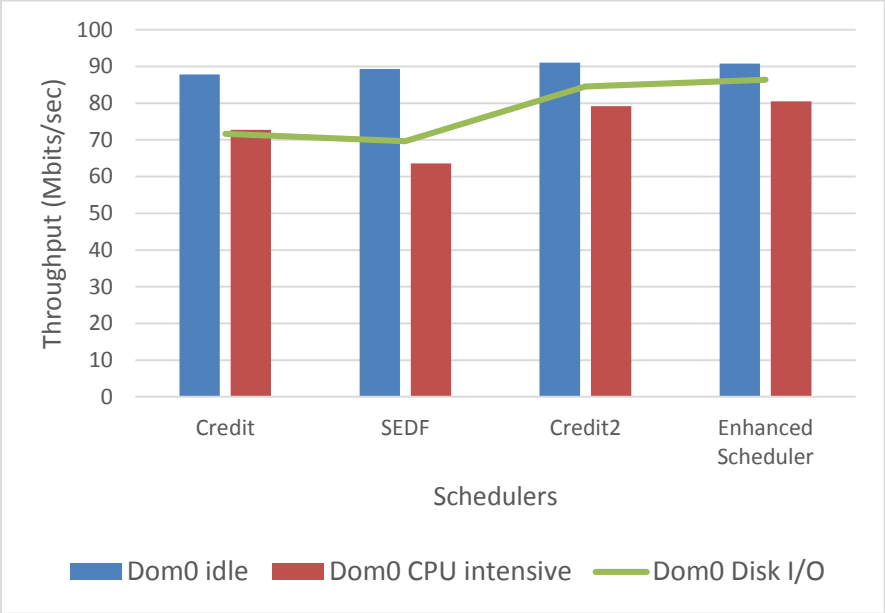


Figure 38 Network Performance of Credit, SEDF, Credit2 and Enhanced Scheduler

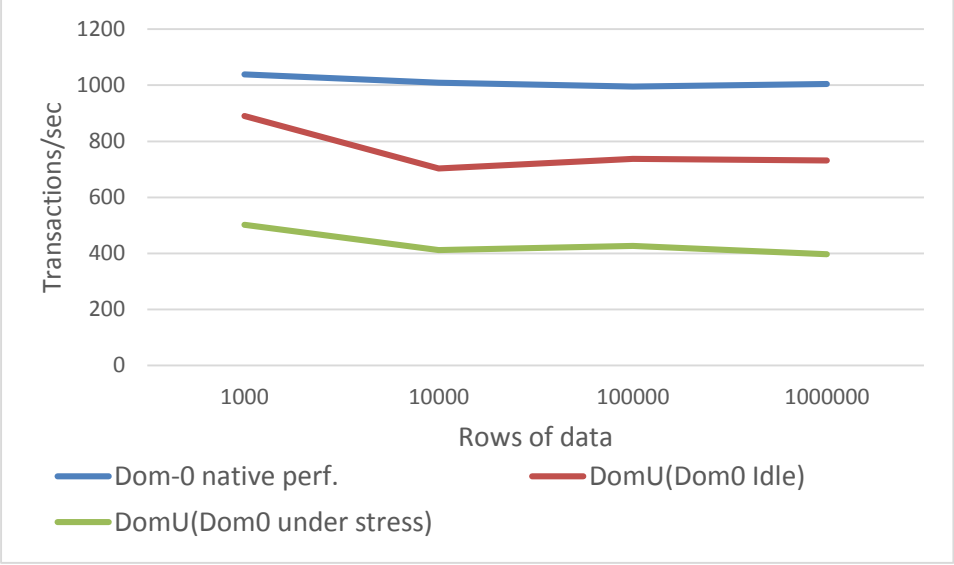


Figure 39 Enhanced scheduler database server performance

The comparative chart for network intensive benchmarking of the four schedulers is shown in the Figure 38, which indicates that the enhanced scheduler has the minimal performance drop when a network I/O intensive domain is competing with a compute-intensive domain or an I/O intensive domain.

The Enhanced scheduler exhibits superior handling of database server requests, shown in Figure 39, with the native domain performance similar to that of Credit and Credit2. The loss of number of Transactions per second when domU is handling is lower than Credit2 exhibiting the high I/O throughput rate which is characteristic of the design.

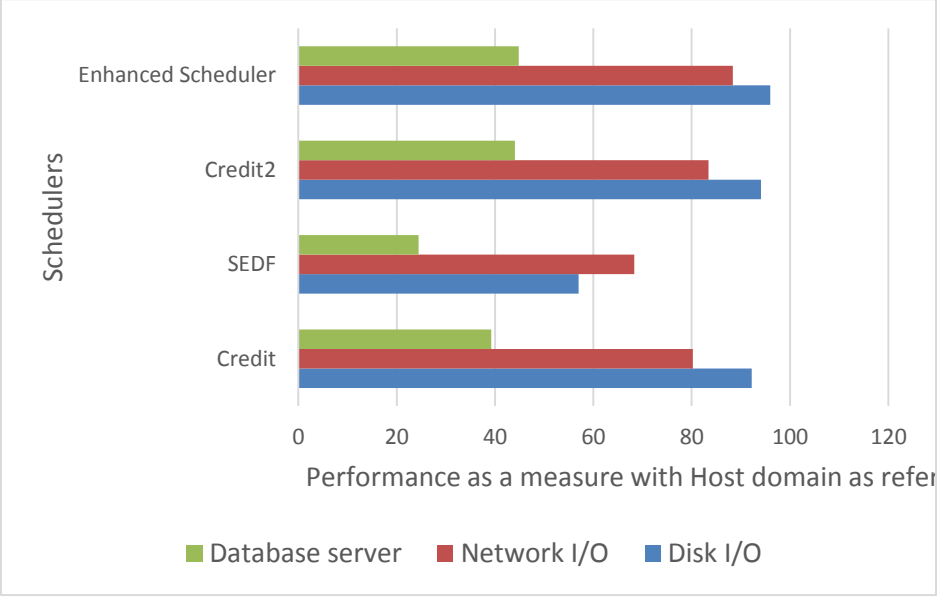


Figure 40 Comparison of Enhanced Scheduler in a consolidated workload environment

The enhanced scheduler's handling of I/O heavy domains marginally surpasses the Credit2 scheduler in a consolidated workload experimental setup. As shown in Figure 40, the proposed scheduler has a slight edge over the Credit2 while performing database transactions and Network intensive I/O handling.

4.6 Summary

The Enhanced scheduler designed and implemented in this chapter has been put under different realistic workload scenarios and benchmarked. The results indicate that the motives

of design, which are improved handling of compute-intensive domains and to improve effective I/O throughput, have been realised. Though the performance of the scheduler in idle conditions is not on par with the credit scheduler due to the overhead context switching accumulate on the CPU, the behaviour of the scheduler when subjected to stressful tests yielded favourable results. The enhanced scheduler's handling of I/O heavy domains with better throughput rate than the Credit2 scheduler in a consolidated workload exhibits the design characteristics of the proposed VCPU scheduling scheme. As shown in Figure 39, the proposed scheduler has a slight edge over the Credit2 while performing database transactions and Network intensive I/O handling. The enhanced scheduler upheld its performance defiantly which indicates that when competing with other domains on the host, regardless of compute intensive or I/O intensive nature, every domain is ensured scheduling fairness.

Chapter 5 - Conclusion & Future Work

5.1 Conclusion

Based on the concepts discussed, the more commonly known definition of virtualization can be expanded to include that it is an abstraction layer between two software layers as well. The multitude of virtualization types available for improving the effective usage of every component of a computing infrastructure shows the distance the technology has come, invoking industrial research contributions to open source projects. We have also discussed about the challenges faced in virtualizing computing resources such as CPU, memory, I/O, network, application and the solutions implemented by the vendors in their products, thus effectively paving a path for further exploration on such issues. The discussion on server sprawl problem and consolidation as a solution shed light on the VM manageability and migration troubles which can rise in the control plane. Xen hypervisor's architecture has been looked upon briefly following the challenges in the VM environment to be expected by a VMM.

The choice of CPU scheduling algorithm used by Xen hypervisor evidently impacts the performance of the applications hosted on the machine. We analysed the CPU scheduling algorithms incorporated into Xen so far, since its earliest version and conducted exhaustive experiments using simulated workloads on the latest batch of schedulers released by the Xen developers' community.

The most significant inference from the experiments is that servicing of I/O by the driver domain depends heavily on Xen's scheduling capacity. By focussing on the I/O handling tendency of each scheduler provided an insight into the right usage of the hypervisor for workload specific applications. Though varying the scheduler parameters could even out some odds in the scheduler's functionality, true scheduling fairness is achieved by achieving optimal performance on all domains without disproportionate usage of resources.

The behaviour of the hypervisor when faced with heavy workloads was discussed and challenge it poses to the resource usage by the scheduler in charge. Credit and Credit2 have proven to be robust in their handling of heavy I/O workloads such as Disk, bandwidth intensive network and database server. Credit2, though in experimental phase has been observed to exhibit and deliver the promised improvement in I/O handling over the Credit scheduler. The older SEDF scheduler has been observed to fall behind in contention with both the Credit scheduler versions. ARINC 653 is a special case scheduler built for uniprocessor architectures and hence cannot be compared with the other schedulers in this version of Xen.

The Enhanced scheduler designed and implemented has been put under different scenarios and benchmarked. The results indicate that the motives of design which are improved handling of compute-intensive domains and to improve effective I/O throughput have been realised although overlooking a small percentage of context switch overhead. Though the performance of the scheduler in idle conditions is not on par with the credit scheduler due to the overhead accumulated because of context switching, the behaviour of the scheduler indicates that when competing with compute intensive or I/O intensive domains, every domain is ensured scheduling fairness.

5.2 Future work

Our future work involves further study on the capabilities of Xen's scheduling algorithms when employed as a hypervisor in collaboration with an open standard cloud computing platform such as Openstack. The live VM migration features of Xen and efficient VM management in a cloud environment seem to be the logical and viable step forward.

Furthermore the application potential of Xen as a Network Function Virtualization (NFV) solution for current SDN based networks can be explored. Another focus area for supplementing the efforts of this research, in terms of green computing, is the study of VM migration strategies for energy efficient data center management.

References

[1] J. Sahoo, S. Mohapatra, R. Lath , "Virtualization: A Survey on Concepts, Taxonomy and Associated Security Issues," 2nd International Conference on Computer and Network Technology (ICCNT), pp. 222-226, 2010.

[2] Smith, J.E.; Ravi Nair; , "The architecture of virtual machines," *Computer* , volume 38(5), pp. 32- 38, May 2005

[3] VMware white paper on, "Understanding Full Virtualization, Paravirtualization and Hardware Assist", latest revision: November 10, 2007. Retrieved from: http://www.vmware.com/files/pdf/VMware_paravirtualization.pdf

[4] Khan, A.; Zugenmaier, A.; Jurca, D.; Kellerer, W.; , "Network virtualization: a hypervisor for the Internet?," *Communications Magazine, IEEE* , volume.50(1), pp.136-143, January 2012

[5] Xianmin Wei; , "Application of Server Virtualization Technology in Enterprise Information," *Internet Computing & Information Services (ICICIS), 2011 International Conference on*, pp.25-28, 17-18, 2011

[6] VMware product overview, VMware vCenter Converter™, http://www.vmware.com/products/converter/overview.html

[7] VMware product overview, VMware VMotion, http://www.vmware.com/files/pdf/VMware-VMotion-DS-EN.pdf

[8] Barham P, Dragovic B, Fraser K *et al*. Xen and the art of virtualization. *ACM Symposium on Operating Systems Principles* 2003, pp.164–177.

[9] Isolation Benchmark Suite, http://web2.clarkson.edu/class/cs644/isolation/index.html

[10] Netperf, http://www.netperf.org/netperf/training/Netperf.html

[11] IOzone Filesystem Benchmark, http://www.iozone.org/

IOzone Documentation, http://www.iozone.org/docs/IOzone_msword_98.pdf

[12] B. Snyder, “Server virtualization has stalled, despite the hype,” InfoWorld, 2010. http://www.infoworld.com/t/server-virtualization/what-you-missed-server-virtualization-has-stalled-despite-the-hype-901

[13] P. Barham, B. Dragovic, K. Fraser et al., "Xen and the art of virtualization," ACM Symposium on Operating Systems Principles, pp. 164–177, 2003.

[14] S. H. VanderLeest, "ARINC 653 hypervisor," 29th IEEE/AIAA Digital Avionics Systems Conference (DASC), pp. 3-7, Oct 2010.

[15] K. J. Duda and D. R. Cheriton, “Borrowed-Virtual-Time (BVT) scheduling: supporting latency-sensitive threads in a general-purpose scheduler”, 17th ACM SIGOPS Symposium on Operating Systems Principles”, volume 33(5) of ACM Operating Systems Review, pp. 261–276, Dec. 1999.

[16] I. M. Leslie, D. McAuley, R. Black, T. Roscoe, P. Barham, D. Evers, R. Fairbairns, E. Hyden, "The design and implementation of an operating system to support distributed multimedia applications," IEEE Journal on Selected Areas in Communications, volume 14(7):1280-1297, Sept. 1996

[17] Credit scheduler, http://wiki.xensource.com/xenwiki/CreditScheduler

[18] L. Cherkasova, A. Vahdat, D. Gupta, “Comparison of the three CPU schedulers in Xen”, ACM SIGMETRICS Performance Evaluation Review (PER), volume 35(2), pp. 42-51, Sept. 2007.

[19] D. Kim, H. Kim, M. Jeon, E. Seo, J. Lee, “Guest-aware priority-based virtual machine scheduling for highly consolidated server”, Euro-Par 2008--Parallel Processing, pp. 285-294, 2008.

[20] N.M.K. Varma, D. Min, E. Choi, "Diagnosing CPU utilization in the Xen virtual machine environment," 6th International Conference on Computer Sciences and Convergence Information Technology (ICCIT), pp. 58-63, 2011.

[21] I. M. Leslie, D. Mcauley, R. Black, T. Roscoe, P. T. Barham, D. Evers, R. Fairbairns, and E. Hyden. The Design and Implementation of an Operating System to Support Distributed Multimedia Applications. IEEE Journal of Selected Areas in Communications, 1996.

[22] T. Wood, P.J. Shenoy, A. Venkataramani, M.S. Yousif, “Sandpiper: black-box and gray-box resource management for virtual machines”, The International Journal of Computer and Telecommunications Networking, volume 53(17) 2923–2938, December 2009.

[23] Corporation, Virtualization technologies from Intel, 2015. http://www.intel.com/content/www/us/en/virtualization/virtualization-technology/intel-virtualization-technology.html

[24] Advanced Micro Devices, AMD’s virtualization solutions, 2015. http://www.amd.com/en-us/solutions/servers/virtualization

[25] Citrix, Citrix XenServer,2015. https://www.citrix.com/products/xenserver/overview.html

[26] Microsoft, Microsoft Hyper-V Server, 2015. https://technet.microsoft.com/library/hh831531.aspx

[27] Xen Project, Linux foundation collaborative projects, 2015. http://www.xenproject.org/

[28] HaiBing Guan, YaoZu Dong, RuHui Ma, Dongxiao Xu, Yang Zhang, Jian Li, "Performance Enhancement for Network I/O Virtualization with Efficient Interrupt Coalescing and Virtual Receive-Side Scaling", IEEE Transactions on Parallel & Distributed Systems, volume 24(6), pp. 1118-1128, June 2013.

[29] Cisco white paper on "Unify Virtual and Physical Networking with Cisco Virtual Interface Card", 2011. Retrieved from : http://www.cisco.com/c/en/us/products/collateral/interfaces-modules/ucs-m81kr-virtual-interface-card/white_paper_c11-618838.pdf

[30] Jianhua Che; Wei Yao; Shougang Ren; Haoyun Wang, "Performance Analyzing and Predicting of Network I/O in Xen System," in Dependable, Autonomic and Secure Computing (DASC), 2013 IEEE 11th International Conference on , pp.637-641, Dec. 2013

[31] Sailer, R.; Jaeger, T.; Valdez, E.; Caceres, R.; Perez, R.; Berger, S.; Griffin, J.L.; van Doorn, L., "Building a MAC-based security architecture for the Xen open-source hypervisor," in 21st Annual Computer Security Applications Conference , pp.-285, Dec. 2005

[32] Jia Tian; Yuyang Du; Hongliang Yu, "Characterizing SMP Virtual Machine Scheduling in Virtualization Environment," in Internet of Things (iThngs/CPSCom), 2011 International Conference on and 4th International Conference on Cyber, Physical and Social Computing , pp.402-408, Oct. 2011

[33] Alin Zhong; Hai Jin; Song Wu; Xuanhua Shi; Wei Gen, "Optimizing Xen Hypervisor by Using Lock-Aware Scheduling," in Cloud and Green Computing (CGC), 2012 Second International Conference on , pp.31-38, Nov. 2012

[34] Kun Cheng; Yuebin Bai; Rui Wang; Yao Ma, "Optimizing Soft Real-Time Scheduling Performance for Virtual Machines with SRT-Xen," in Cluster, Cloud and Grid Computing (CCGrid), 2015 15th IEEE/ACM International Symposium on , pp.169-178, May 2015

[35] Sysbench benchmark tool: https://dev.mysql.com/downloads/benchmarks.html

[36] Xiaobo Ding; Zhong Ma; Xinfa Dai; Alin Zhong, "Scheduling Parallel Virtual Machine in Xen Based on Credit," in Services Computing Conference (APSCC), 2014 Asia-Pacific , pp.191-198, Dec. 2014

[37] Babu, S.A.; Hareesh, M.J.; Martin, J.P.; Cherian, S.; Sastri, Y., "System Performance Evaluation of Para Virtualization, Container Virtualization, and Full Virtualization Using Xen, OpenVZ, and XenServer," in Advances in Computing and Communications (ICACC), 2014 Fourth International Conference on , pp.247-250, Aug. 2014

[38] Ruiqing Chi; Zhuzhong Qian; Sanglu Lu, "Be a good neighbour: Characterizing performance interference of virtual machines under xen virtualization environments," in Parallel and Distributed Systems (ICPADS), 2014 20th IEEE International Conference on , pp.257-264, Dec. 2014

[39] Xudong Jia; Liang Heng, "Virtualization in Enterprises' Different Growth Stages and Its Motives: A Classical Grounded Theory Research," in Computational Sciences and Optimization (CSO), 2014 Seventh International Joint Conference on , pp.228-232, July 2014

[40] Wei Chen; Hongyi Lu; Li Shen; Zhiying Wang; Nong Xiao, "DBTIM: An Advanced Hardware Assisted Full Virtualization Architecture," in Embedded and Ubiquitous Computing, 2008. EUC '08. IEEE/IFIP International Conference on , volume 2,pp.399-404, Dec. 2008

[41] Yida Xu; Hongliang Yu; Weimin Zheng, "A Consistent Backup Mechanism for Disaster Recovery that Using Container Based Virtualization," in China Grid Annual Conference (ChinaGrid), 2012 Seventh , pp.95-100, Sept. 2012

[42] Regola, N.; Ducom, J.-C., "Recommendations for Virtualization Technologies in High Performance Computing," in Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on , pp.409-416, Nov.2010

[43] Natural Resources Defense Council (NRDC), Data Center Energy Assessment report, August 2014. Retrieved from: https://www.nrdc.org/energy/files/data-center-efficiency-assessment-IP.pdf

[44] Hussain, T.; Habib, S., "A redesign methodology for storage management virtualization," in Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on , pp.676-679, May 2013

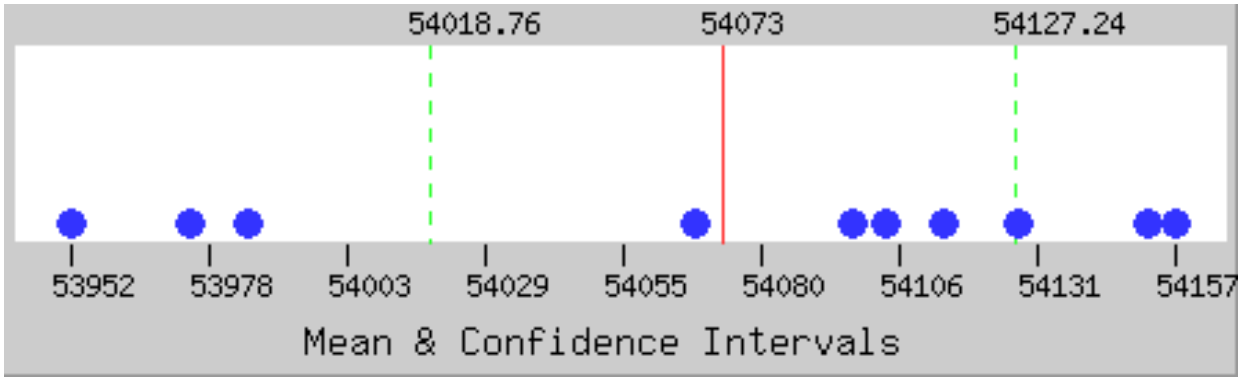
[45] Ackaouy, E.: The Xen Credit CPU Scheduler, XenSource: Open Source Enterprise Virtualization, September 2006.

[46] Uddin, Mueen, and Azizah Abdul Rahman. "Energy efficiency and low carbon enabler green IT framework for data centers considering green metrics." Renewable and Sustainable Energy Reviews, Volume 16, Issue 6, pages: 4078-4094. August 2012.

Appendix A: Standard Deviation, Standard Error and Confidence Intervals

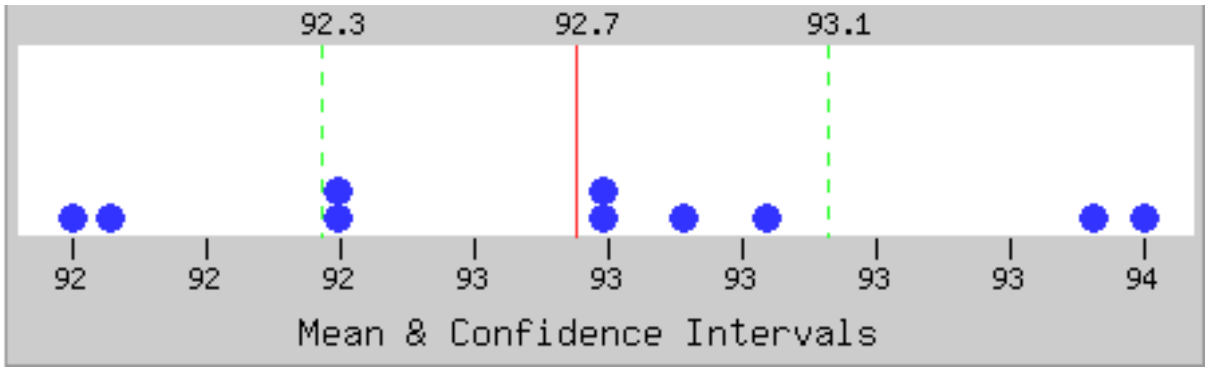
Credit Scheduler	IOzone - S-Read, Dom0 KB/sec	Netperf - Dom0 idle, File size 1MB Mb/sec	Sysbench Dom0 native perf. Transactions/sec
Mean	54073	92.67	1013.7
Variance	5580	0.34	0.57
Standard Deviation	75.79	0.58	0.76
SE Mean	23.967	0.19	0.25
95% confidence Interval	54018.76 to 54127.24	92.3 to 93.1	1013 to 1014.4

IOzone - S-Read, Dom0 Disk I/O (KB/sec):

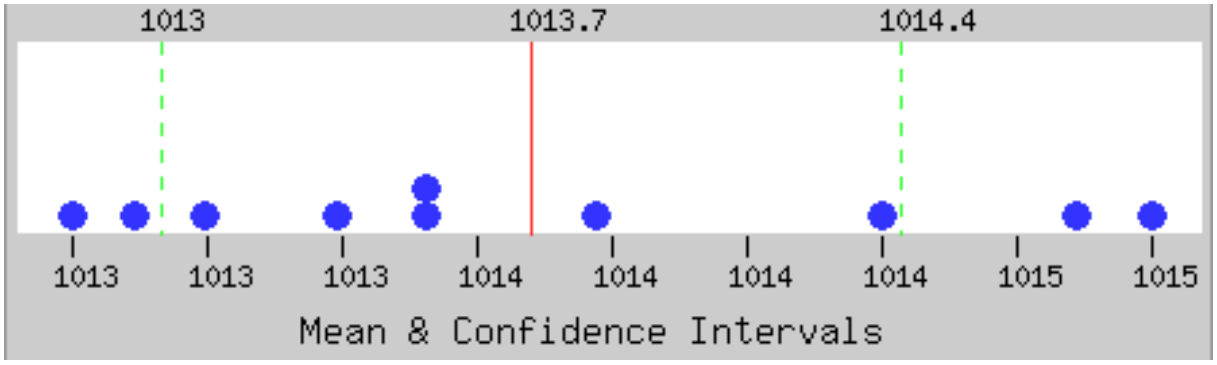


Appendix A: Standard Deviation, Standard Error and Confidence Intervals

Netperf - Dom0 idle, File size 1 MB (Mb/sec):



Sysbench - Dom0 native performance (Transactions/sec):



Appendix B: IOzone CPU and Buffer cache effect [11]