DESIGN OF CLIENT AWARE SCHEDULER FOR XEN WITH ENHANCED TECHNIQUES TO IMPROVE CLOUD PERFORMANCE

A Thesis by

Nani Tippa

Bachelor of Technology, SKTRMCE, JNTU, India 2008

Submitted to the Department of Electrical Engineering and Computer Science
and the faculty of the Graduate School of
Wichita State University
in partial fulfillment of
the requirements for the degree of
Master of Science

May 2013

DESIGN OF CLIENT AWARE SCHEDULER FOR XEN WITH ENHANCED TECHNIQUES TO IMPROVE CLOUD PERFORMANCE

The following faculty members have examined the final copy of this thesis for form and content, and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science with a major in Computer Networking.

Ravi Pendse, Committee Chair

Krishna Krishnan, Committee Member

Abu Asaduzzaman, Committee Member

DEDICATION

To my dad, Bhima Shankar Rao, my mom, Parvathi, and my sister, Shrirekha

ACKNOWLEDGEMENTS

I would like to wholeheartedly thank my advisor and professor Dr. Ravi Pendse for his invaluable insight, guidance and constant encouragement throughout the thesis and Master's study at Wichita State University. I would also like to extend my appreciation to Mrs. Vidya Suryanarayana for her guidance, advice and time during the thesis.

I would also like to thank my committee members Dr. Krishna Krishnan and Dr. Abu Asaduzzaman for their precious time and suggestions.

I would like to thank my leads Amarnath Jasti and Vijay Ragothaman for all their support during my tenure at work. Finally, I would also like to thank my family and friends for their constant encouragement and support during my research.

Finally, I thank my family and friends—Neethu, Pruthvi, Karthik, Navya, Ramya, Sai, Harsha, Datta, Budha, Harindra, Prudvi, Madhu, Dwarak, Vamshi, Srikanth, Surya, Vijay and Hardeep—for their constant encouragement and support during my research.

ABSTRACT

Infrastructure as a Service (IaaS) in cloud provides ample scope for various high volume applications to be run on the servers across the WAN, availing a fair service to the end clients. Many effective schedulers have been designed to consider the contention of the computational and communicational resources, which provide a guaranteed effectiveness for resource sharing. However, the vast diversity of client devices in a cloud demand scheduling based on their features and capabilities. Mobile clients, workstations, laptops, PDAs and thin clients in the cloud vary in aspects such as processing power, screen size, battery life, geographical distance and many. Algorithms in the cloud, which are based on client capabilities result in many consumer benefits such as network load balancing, saving battery, reducing latency, efficient processing. In this thesis, the authors propose a client aware credit scheduler for virtualized server setup in a cloud that schedules the client requests based on the client device features and capabilities. Rich Internet Applications (RIA) is proposed, in order for the server to realize the client device capabilities such as the type of client, the battery remaining and the location of the client. Results show that the client-aware credit scheduler is effective in terms of saving energy and reducing response latency.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION AND OVERVIEW OF THE THESIS

Cloud computing is on its way to forming new trends in the IT industry. Cloud computing uses the Internet to deliver services to the recipient. The recipients are charged on the basis of service rendered. Rather than buying and maintaining own servers, software, data center space, and network equipment, the Infrastructure as a service (IaaS) provides users the abstract of computer infrastructures such as storage, processor, networking as virtualized service over the cloud. Clients buy these resources as a fully outsourced service, which are maintained at another end of the WAN. Cloud services are resourcefully utilized only when virtualization takes its place on the real servers.

In an enterprise cloud environment, the Cloud Service Providers (CSP) accept various requests from the end clients in the form of application services. The end clients depend upon the CSP to compensate resource deficiencies and also to reduce the workload. This type of resource utilization brings in the latencies that could be incurred upon the client. In order to reduce the cost of ownership for deploying the number of physical machines to serve multiple service requests, CSP uses virtualization with multiple Virtual Machines (VMs) installed on servers. Hence, virtualization was proposed to consolidate servers that had various physical machines requiring multiple operating systems. These Virtual Machines, which run on the emulated software or hardware virtualization, are maintained by Virtual Machine Monitors (VMM), which run~~s~~ through the server using independent configurations of operating systems, drivers, and software-- providing multiple services on a single physical host machine [1].

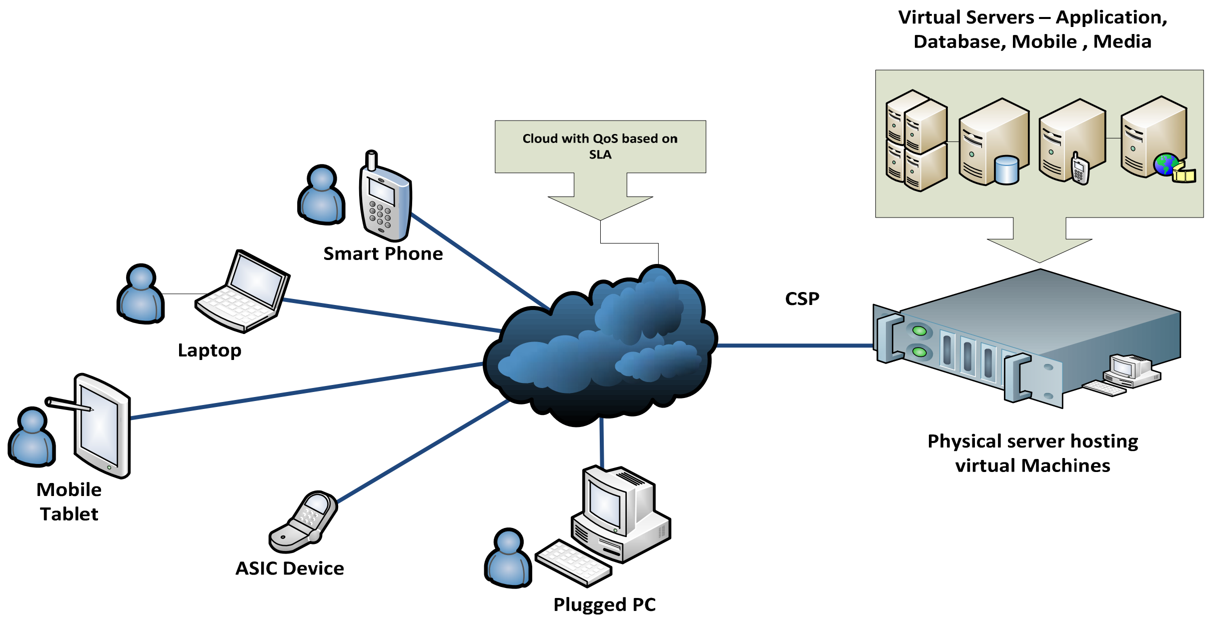
Virtual Servers – Application, Database, Mobile , Media
Cloud with QoS based on SLA
Smart Phone
Laptop
CSP
Mobile Tablet
ASIC Device
Plugged PC
Physical server hosting virtual Machines

Figure 1. Cloud Services

With virtualization in effect, VM's can be used create virtual entities of modules such as storage, network, software and hardware resources under each abstract partition~~s~~ of VMs as shown in Figure 1. Some of the advantages that make virtualization very effective are listed below.

- **Consolidation:** Workloads of multiple servers can be consolidated into **a** single physical machine**,** which guarantees cost effectiveness.
- **Heterogeneity:** Each VM has its own operating system and they can be functional at one time on a single physical machine, which increases efficiency and also implicates compatibility with different applications.
- **Migration:** Redundancy and load balancing is possible because VM can migrate from one physical machine to another using the concept of mobile software compartment.
- **Isolation:** Each VM keeps its resources to itself behaving like a standalone server.

Server virtualization is categorized based on hardware built as Full Virtualization, Partial Virtualization, and Para Virtualization. They differ in the simulations and interactions of the operating system with the physical resources on the server. Xen, a virtualization platform whose architecture is based over Para Virtualization and is a hot topic in the research field because of the benefits like open source, lower performance bottlenecks and cost of ownership.

In Xen hypervisor, as the client places service requests, VMM takes the job of creating the workload event and assigning them to the appropriate VM domain. VMM has the difficulty of allocating the appropriate resources accordingly, due to individuality of sophisticated mechanisms and polices run by the operating systems on the virtual machines compared to light weight components and narrow interfaces of the machine. To bridge this gap, schedulers have been designed to fairly allocate the resources to the VMs based on few parameters which are explained later in chapter 2. Credit Scheduler (CS) and Simple Earliest Deadline First (SEDF) are default schedulers on the Xen hypervisor. Although these schedulers provide a solution to the contention among the computational and communicational process events, there still exists some latency in the services requested by the end clients. When dealing with a cloud, running a variety of traffic, it is important to prioritize certain requests (such as compute intensive, media, online processor intensive game) in order to reduce the latency and also to fairly service the client requests. This latency diversifies when it comes to media-driven applications over the cloud, which requires priority over the other services withheld by the VMM. Virtual machines or virtual servers in Xen are called “domains.” In this thesis, the words virtual machine, virtual server, and domains are used interchangeably.

The cloud encompasses a vast diversity of client devices. Client devices such as mobiles phones, workstations, laptops, PDAs, thin clients, iPad, differ in attributes such as processing power, battery life, screen size, geographical distances and so on. It also becomes important to ensure that the requests from the clients are scheduled based on their attributes for fairness. For example, a client that has low battery remaining calls for higher priority in scheduling when compared to a client that has enough battery life. Rich Internet Applications (RIA) has enabled servers to realize the capabilities of client devices and has driven the deployment of client aware technologies. With RIAs, application codes download to the client device and execute on it using a RIA software framework, which is typically based on HTML5. RIAs may be used to infer the hardware properties of the clients such as processing power, battery life, network bandwidth and other properties [2].

Several enhancements have been proposed for the cloud, which improve the response time for latency sensitive workloads. However, not many schedulers are proposed for the cloud, which take into account the client device attributes and capabilities at servers, when serving the client requests. Client aware scheduling is very important for the design of a cost effective cloud architecture because of the wide variety of clients available and their limitations in various features. Especially because requests in a cloud travel over a WAN, it is important not only to ensure a low latency of service but also to schedule the requests based on the client capabilities.

In order to address the above challenges and issues, the authors in this research propose a Client Aware Credit Scheduler (CACS) for virtual servers in a cloud. The proposed scheduler is a variant of the default credit scheduler, in which the type of client, battery life and distance of the client from the server attributes are used for scheduling the requests.

Chapter 2

XEN AND CLOUD COMPUTING

2.1 Virtualization

Recently many virtualization techniques have evolved in the market for various platforms. A survey conducted states that 50% of the small and medium business organizations are inheriting virtualization by the end of next year [3]. The reasons are obvious with the outcomes in efficiency, performance, cost cuttings, security and less administration. The concept of virtualization and VMM came into existence in the early 1970's, as a software abstraction layer partitioning physical hardware platforms into VMs [4]. During the 1980's they got out of the mainstream with the advent of mainframe, super computers and the cost drop downs of hardware. Finally, it evolved and became a hot research topic after 2005. Following are the key terminologies which would be used thoroughly in the thesis:

- **Hypervisor:** or *Virtual Machine Monitor* is medium defined between hardware resources and guest operating systems for controlling and monitoring the system. Upon the request from corresponding guest operating systems, the hypervisor provides required hardware resources to perform the instructed task, thereby creating the environment, which is identical to the original machine and programs. Lastly, an emulator which can directly transfer the instruction set from the VMs virtual processor to the host processor without any software medium.

- **Virtual Machine:** It is an identical version of the real server or machine. It is efficient because it has a virtual point of view towards physical resources. Booting and running the VM is equivalent to program**s** with fork and execution steps.
- **Guest OS:** Guest OS is a VM with an operating system delivering the instruction set to the processor via VMM.
- **Host OS:** Sometimes this is also referred to as *dom 0*, because in hosted architecture one of the VM is assigned the task of managing the virtualization services and partitioning of the other domains or VMs.

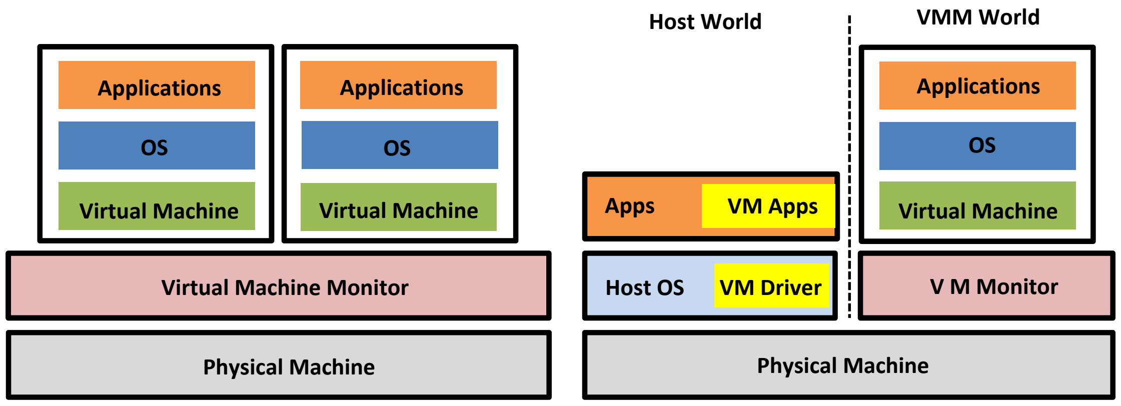

Figure 2. Hosted and Bare metal Hypervisors

2.1.2 Architecture

Robert P. Goldberg classified two architectures designed to support server virtualization – Hosted and Bare Metal [5]. Hosted hypervisor run on the top of the operating system, which is the second software layer and the Guest OS on the third level with the Host OS in level one as shown in the Figure 2. On the other hand bare metal hypervisor can run directly on bare hardware with no layer in between, taking the instructions directly and managing the Guest OS.

The hosted hypervisor provides a wide range of configurations and has some key advantages over the bare metal as following [6]:

- Less space for hypervisor code.
- Existing device drivers of the operating system can used to drive the VMs.
- Low compatibility issues

Bare metal has the advantage of fully owning the hardware-- implying direct access to the physical resources with added scalability and better performance. But at the same time, it has some downsides to the approach in terms of support for the Guest OS. Building drivers will add overhead. With many devices and software on the market compatibility concerns are created.

2.1.3 Virtualization Techniques

There are three significant virtualization techniques to handle the instructions to virtualize the CPU [7, 8].

- **Full Virtualization:** The guest OS is unmodified in this type of virtualization because of the ability of the hypervisor to convert all the instructions that cannot be passed to the CPU. Technically speaking, the hypervisor converts the kernel code of the operating system to new instruction set compatible on the server which is intended to affect the virtual hardware work-- keeping the user-level instruction unmodified at native speed. The advantage is that any operating system can work on this hypervisor without modifying the original code. The downside of this technique is that it adds a lot of overhead in regards to the performance and cost. Some examples based on full virtualization are VMware ESX/EXSi, VMware workstation, VMware server, and VirtualBox.

- **Para Virtualization:** The guest OS is modified to run on the servers, which are Para virtualized. Thus, the operating system is aware that it is running on a virtualized environment. This technique is little fast compared to the fully virtualized model since guest OS is modified to run on ring 1. The kernel of the guest OS is modified by replacing the instructions with hyper calls to interact directly with ring 1 which improves hypervisor performance. This requires the vendor to support the modification. Since Linux is free and open source, most of the Linux distributions support this technique. Xen is one of the best para virtualized platform.
- **Hardware Assisted Virtualization:** In this virtualization**,** the guest OS is given the privilege of ring 0 of the protected domains and hypervisor is given a special domain called root privilege. Any operating system can be virtualized using hardware-assisted virtualization without modifying the guest OS. The performance overhead is also improved compared to other virtualization techniques. Intel VT is based on this technique.

2.2 Xen Architecture

Xen— an open source hypervisor, or VMM, is designed on the fundamentals of separation of policy and mechanism [9, 10], that is it implements the mechanism but leaves the policy to be controlled by the domain 0 which is explained in the next section. Based on bare metal architecture, VMs are executed securely and efficiently with near native performance [11]. As mentioned before, for IA32 platforms the applications are put in ring 3 privilege, hypervisor in ring 0**,** and kernel in ring 1. In IA-64 platforms**,** because of only 2 rings, the kernel has been moved to the ring 3 along with applications [9]. The application binary interface is also unmodified so that guest OS applications run unchanged**—** that is**,** only machine dependent code change in the kernel. The other requirement which adds to the build of a solid hypervisor is the replacement of the privileged instructions with hyper calls in the guest OS.

2.2.1 Role of Domains

In Xen**,** the instance of an operating system is referred to a domain in the virtualized environment. There are two types of domains defined:

- Domain 0 (*dom 0*): *Dom 0* is the instance of host OS run on the machine. It is very important for Xen and it is mainly responsible for managing the physical resources in support to the VMM. When Xen is booted, *dom 0* is the first operating system to be loaded.
- Domain U (*dom u*): The “U,” in *dom u*, stands for underprivileged. *dom u* is the instance of guest OS exported by the VMM. It is restricted with access and the communication to the hypervisor is only possible through the front end of the split drivers.

Dom 0 runs at a higher privilege than the other domains to so that it can be accessible. Furthermore, *Dom 0* administrates the devices with which it has to function. The major role of *dom 0* is that it has the ability to multiplex the communication channels for the VMs with the devices. This is mainly done with the help of split device drivers. It has the ability to create and terminate the other domains and to control their physical memory allocations, scheduling parameters and their access to resources like network devices and CPU [11, 12].

2.2.2 Xen I/O model

When the guest OS is booted virtual interfaces for the block devices are created and initialized. This comprises of frontend parts, which reside in *dom u* and are backend in *dom 0*.

These two function together to provide control, communication, and data transfer between domains. The frontend interfaces, which combine both the netfront and blockfront entities, are operated by the *dom u* drivers, which perform the data transfer and corresponding event communications to their backend counterparts through shared memory and inter-domain event channel notifier. The backend drivers in *dom 0* are completely responsible for the scheduling of I/O operations of guest domains.

Xen also uses descriptor ring buffers to transfer the data between the domains and sometimes between the interface and domains. I/O descriptors of corresponding block address are placed in the circular queue, and the corresponding action is performed after the appropriate event notification. The data resides in shared memory between *dom u* and *dom 0*.

2.2.3 Event Channel Notifier

Xen utilizes event channel mechanism**s** to deliver Interrupt Requests (IRq's) to the driver domain. Pending Array, a two level hierarchy of bit vectors is between the driver domain and hypervisor kernels, where the second level of the hierarchy corresponds to the port**,** which maps to the IRq number. A pending event is notified with the set bit in the array. The first level is used to optimize the excessive search of set bit, which is ~~also~~ set when the bit is set in the second level. The scanning of the set bit is done in sequential manner. After detecting the set bit and processing the IRq, the scan begins from the vector where it has left off. It will start scanning from the beginning once the two level hierarchy has been completely scanned.

2.2.4 Credit Scheduler

There are three main schedulers in Xen— Borrowed Virtual Time Scheduler (BVT), Simple Earliest Deadline First Scheduler (SEDF)**,** and Credit Scheduler. The BVT and SEDF schedulers are not being considered in this thesis because of their lack of support for load balancing across physical CPUs. They also do not support the non-work conserving mode in scheduling the domains [13, 14]. In default**,** Xen 3.0 comes with Credit Scheduler**,** which is advanced and actually does automatic load balancing. The domains are allocated with few parameters with which they are scheduled. The typical parameters in use are the number of VCPUs, weight, and cap value. Virtual CPU (VCPU) seen by guests corresponds to the underlying physical CPU (PCPU). The important design factor of the scheduler includes the balancing of the VCPUs across the underlying PCPUs. Cap is the maximum amount of CPU a domain can use and it is expressed in terms of percentage. Weight is a complex parameter which gives the relative share of the CPU given to domain.

Current Credit scheduler is designed to fairly allocate the processor resources, which are weighted by the number of credits allocated to each domain [15]. Each domain is assigned a certain number of credits according to the weight, and it is usually done by the system administrator. 100 credits are debited from the currently running domain every 10ms.When the sum of the credits of all the domains goes negative, new credits are assigned to each of them. Domains are classified as belonging to either UNDER, where a domain has some credits remaining, and OVER when a domain has run out of credits. The credit scheduler schedules the ones with the UNDER state first and then the ones with OVER state, which are eventually put into a run queue based on a first-in first-out flow. The domain at the head of the run queue is allowed to run for 30ms as long as it has credits remaining.

In order to achieve low I/O response latency, another state known as BOOST state was introduced to the original model [15, 3]. A domain in the BOOST state has higher priority than the UNDER and OVER state. Generally**,** when an I/O event comes over the event channel, the idle domain is boosted to the BOOST state, which preempts the currently running domain and starts running immediately without entering the tail of the run queue.

2.3 Rich Internet Applications

RIAs are web applications that have many of the features of traditional desktop applications, as they introduce another layer of processing on the client that can handle richer features such as rendering, state management**,** and storage. RIAs can be used to infer the hardware capabilities of client devices. Several whitepapers have been written on how to use RIAs to exploit the client information. The authors in [16, 17, 2] explain how to get the input device information in Linux such as the presence of a physical keyboard, physical pointing mouses, and device changes. According to the paper, in Linux, the Hardware Abstraction Layer (HAL) can be used to retrieve information about the hardware that makes up a computer system. HAL can be queried using the command line**,** and the same information can be obtained inside of an application using data bus. Using the HAL, the server can infer if the client requesting service is a mobile client or wired client. Similarly, the author in [17] explains how to detect processor information such as number of cores and processor utilization using RIAs. [18] Explains the role of RIAs in retrieving storage information of a client device.

Industries are currently working towards increasing the user experience in a cloud and they are widely deploying RIAs to exploit the client information in order to render better user experience**s**. Cloud application developers are increasingly looking for a means to offer Rich Internet Applications, which distribute processing between the cloud and the client device to improve application responsiveness [2]. With RIAs, application codes download to the client device and executes on the client using a RIA software framework, which is typically based on

HTML5 [2]. In this thesis, the authors use the benefits offered by RIAs in order to efficiently schedule the requests in a cloud based on the device capabilities.

Chapter 3

LITERATURE SURVEY AND PROBLEM DESCRIPTION

The work in this thesis is related to the redesigning of the VMM scheduler in Xen environment, and previous work has been done in this area to achieve improvements in terms of CPU contention, network throughput, TCP, communication between VMs and VMs deployed to route traffic. The work emphasizes the latencies in terms of application requests, which hit the VMs for computation. The RIA explained in chapter 2 can be exploited in terms software framework to provide the hardware capabilities to the server with which an effective scheduler has been proposed in next few sections.

3.1 Literature Survey

One of the Cloud Computing forums describes Mobile Cloud Computing (MCC) as an infrastructure in which both data processing and the date storage move the computing power and data away from the mobile platform to the cloud with servers capable of servicing large numbers of subscribers [19]. In [20] Hoang T. Dinh et al studied the importance of MCC and explained how battery consumption on a mobile device is reduced by moving data processing from mobile or to high performance servers. Similar findings were done by A. Rudenko et al in [21] specifically on laptops. Based on evaluations, both studies show that 45% of battery consumption is reduced by opting to use remote process execution.

A. Carroll in [22] also performed a detailed analysis on power consumption by concentrating in the area of Smart phones. Component-wise breakdown has been done to measure the power consumption and has been tested with micro-benchmarks, which are built on realistic scenarios. The author showed how these different usage scenarios translate into overall power consumption and consumption of battery life.

Y. Liu and co in [23] performed an empirical evaluation on the battery power consumption of the mobile devices for the date streaming applications. The authors analyzed the most frequently used streaming protocols used in the internet for streaming data. They show that chunk-based streaming was more power efficient because of its adoption to PSM traffic shaping technique. They also show that P2P streaming is not efficient since it adds load by re-uploading and transmitting control traffic.

The authors in [24] enhanced the client-aware technologies and apply them to desktop virtualization and cloud by inferring main advantages from the end-point device capabilities. They approach the objectives by featuring a private cloud that has the ability to infer the device attributes and capabilities and to adapt services accordingly. They emphasize on creating a balance between client-hosted virtualization and server-hosted virtualization by implementing client-aware technologies like workload shifting, multimedia redirection, command remoting, reverse seamless technology, and rich internet applications.

D. Ongaru et al in [15] have extensively studied the Credit Scheduler of the Xen hypervisor and proposed few enhancements, which effectively improved the scheduling performance of the domains. The improvements include sorting the run queue by boosting the I/O handling domains giving priority over the other. They also propose a mechanism to reduce the pre-emption of the event channel notifier improving the overall latency and performance aspects.

G. Liao et al in [25] have demonstrated some software techniques to improve the virtualized I/O performance in multi-core systems. They state that the extra driver domain for I/O requests processing and the scheduler for the VMM are responsible for the degradation of the I/O performance. To reduce these overhead they propose to solutions – a cache aware scheduling to reduce the inter domain communication latency and a mechanism to steal the credits from other domains so they can allocate them to the I/O intensive VCPUs.

Saransh Mittal in [6] has studied the application patterns in the virtualized environment and proposed a mechanism to do quantitative based network scheduling. The author combined the working of the CPU credit scheduler in Xen and DWRR scheduler used in routers to develop a mechanism and he implemented on the network backend drivers. Evaluations show that QoS requirements are fulfilled because of the corresponding bandwidths allocated for respective application.

In [26] S. Govindan et al have developed an algorithm to improve the performance of the virtualized based hosting platforms to enable satisfactory application performance even under conditions of high consolidation, while still adhering to the high-level resource provisioning goals in a fair manner. They develop the algorithm to schedule the CPU of VMM that considers the I/O behavior of the VMs— creating an unfair advantage to communication intensive applications over CPU intensive ones.

3.2 Problem Description

In this section**,** the issues in the existing Default Credit Scheduler model in cloud environment ha**ve** been elaborated.

- Although the default credit scheduler provides fairness of CPU to its domains, it is not suitable for client-aware scheduling. In [15], the authors added the BOOST state so that an I/O domain can start executing immediately without having to wait for other domains to execute. This prioritizes the I/O requests when compared to processor intensive requests. In an IaaS and PaaS cloud, most of the requests will be I/O and will need to access data from the disk arrays. In such a case, the BOOST state will not be efficient.
- In a cloud, since the requests traverse through the WAN, certain client device capabilities like battery remaining, distance from the server, and processor utilization need to be taken into account. The default credit scheduler is more focused on providing fair CPU resources to the virtual servers and it does not prioritize the requests.

Hence, the authors propose a client-aware credit scheduler that schedules the domains according to their priorities, which is based on the device capabilities. The task of priority assignment is offloaded to Domain 0, which is the driver domain. From there, the driver domain decides the priority of a request according to the client device capabilities. It must be noted that priorities are assigned to requests present in the same queue as they were added by the cloud middleware. Therefore, requests within the High, Medium, and Low priority queues are assigned to their respective queues and those queues are serviced separately. This ensures that the services and the QoS according to the SLA are still met.

Chapter 4

PROPOSED SOLUTION

4.1 Client Aware Scheduling

The proposed client-aware credit scheduler schedules the client requests based on the client device features and capabilities. The server is made aware of the client features and capabilities by using the Rich Internet Application (RIA) and the way it is done is explained below.

4.1.1 Need for Client Aware Credit Scheduler

The cloud encompasses a variety of client devices. A lot of industries and organizations these days are resorting to the cloud and requesting software, infrastructure, and platforms as services. Mobile clients, PDAs, workstations, laptops, tablets, ASIC devices and smart phones are some of the client devices to name that request services from the cloud. These devices vary in a wide variety of attributes such as screen size, screen resolution, battery usage, location, processing power and so on. Workstations are usually plugged to a power outlet while mobile clients need to be power aware. Laptops and workstations have a higher processing power compared to mobiles clients and smart phones. Certain clients are located closer to the servers than the other clients. Mobile clients and smart phones have built in GPS capabilities. In order to achieve a balance in providing fair service [27] to the mobile clients and plugged in clients, the client-aware credit scheduler is proposed and it has many advantages.

4.1.2 Advantages in terms of Energy Consumption

In order to be beneficial to the clients, certain features and capabilities may be taken into account to schedule the requests from the clients in a cloud. Battery is one of the main concerns for mobile devices [27, 28]. The energy consumption of a mobile device is affected by the complete end-to-end event chain from the client sending request to the server and server responding back. In web applications, the server response times can have a significant impact on the energy consumption of mobile clients [27, 29]. Hence, by reducing the server response time, the power consumed by the mobile client may be reduced.

According to [27], the virtual server response time (T_r) is a function of scheduling time ($Sched_d$) and seek time ($Seek_d$). By reducing either the scheduling time or the seek time, the response time of a virtual server may be reduced. For local virtual server architectures,

$$T_r = Sched_d + Seek_d \tag{1}$$

In this thesis**,** the above equation for a cloud is modified. Since WAN is an important entity, the total response time (T_r) depends on the network latency as well. Hence, total response time for a client in a cloud can be written as:

$$T_r = Network_i + Service_i + Seek_d \tag{2}$$

$Service_i$ is the time taken to service the request from client 'i' by the virtual server 'd'. In a virtual server setup, the service time is the sum of the time required to schedule the domain and the time required for the domain to execute. The service time can be further written as:

$$Service_i = Sched_d + E_d \tag{3}$$

The main aim of this paper is to reduce the scheduling time of the domains based on the client attributes which are requesting for service from the domain. Once a mobile client has issued a request, it is in the suspended state. This means that the client application processor is in idle state, while the communications processor is active in order receive the response from the server. Because there is a greater response time, greater communication power is also consumed. Since energy (*Energy*$_{idle}$) is a product of power and time [30], the energy used by a client when waiting for a response can be written as:

$$Energy_{idle} = Pwr_{idle} \times Time \tag{4}$$

In a cloud, the time for which the client waits is the response time for the client. Hence,

$$Energy_{idle} = Pwr_{idle} \times T_r \tag{5}$$

Substituting equation (3) in 2,

$$Energy_{idle} = Pwr_{idle} \times (Network_i + Sched_d + E_d + Seek_d) \tag{6}$$

As seen in equation (6), a reduction in the scheduling time results in less power wastage in mobile devices. Hence, requests from mobile clients may be scheduled earlier than a request from a workstation.

4.1.3 Advantages in Terms of Reducing Latency

Similarly, a client that is distantly located may be scheduled earlier than a client that is closely located. In a cloud, since the requests travel through the WAN, a client device that is far away from the server needs to be serviced sooner than a client device that is nearer to the server, so that the response time for the client decreases. It can be seen from equation (3) that, as the network latency increases, the response time for the client also increases. Most of the network latency comes from the distance between the client and the server and also from the processing latency at the routers. The proposed client-aware credit scheduler services the requests from far away clients faster than nearer clients, in order to alleviate the response time. In order for the server to realize these client device capabilities, the authors propose is to use the Rich Internet Applications. The RIA and its benefits are explained in chapter 2.

Therefore, if the total response time is reduced, the energy wastage by a mobile client can be minimized and the latency for a distant client can be minimized. However, battery life and distance are two independent attributes and have trade off over the other. In order to achieve a balance in saving the power used by mobile clients and to reduce the latency of distant clients, the proposed client-aware credit scheduler uses two queues, one for holding mobile client requests and the other for holding plugged in client requests.

4.2 Client Aware Credit Scheduler

When the clients in a cloud place a service request, the middleware in the cloud assigns the requests to a high, medium or a low priority queue based on the SLA [31]. Each of these queues are sent to the servers separately for getting serviced. In server virtualized environments, a single physical server hosts multiple virtual servers and each of the virtual servers accept the appropriate requests to service.

The client-aware priority scheduler proposed in this research takes into account the client device capabilities inside the high, low, and medium priority queues and schedules the requests inside each queue based on the client capabilities that placed the requests. This ensures that the original priority and SLA of the requests is still maintained, yet scheduling is client-aware inside each of the priority queues. The three main device capabilities considered in this research include the type of client (mobile or plugged in), the battery remaining, and the distance of the client from the server. There may be other important parameters that can be considered, such as bandwidth, storage capacity, and so on. For example, client bandwidth information may be used to dynamically adjust data delivery and may be used for load balancing. However, in this research, the authors use only device type, battery capacity, and distance metrics for scheduling purposes.

4.2.2 Ordering the Run Queue

The domains ready for scheduling are sent to the VMM. The virtual servers intending to execute are added to a run queue and the domain at the head of the run queue executes before the next domain. In the default credit scheduler, the domains in the run queue are arranged in the First Come First Serve (FCFS) order. In [25, 34], the research done by the authors reveals that an FCFS scheduling is not suitable for latency sensitive applications. The authors in [25, 32, 33] also state that the default credit scheduler does not take the task information in each VM into account. Hence, they implement a mechanism using shared variables in which every virtual server reveals the priority and status of its tasks to the VMM. Based on this information, the priority of a guest domain is taken as the highest priority of tasks in the run queue of the guest domain. The algorithm proposed by the authors in [32] takes into account both the run queue and wait queue of a guest domain to choose the next VM to be scheduled.

In the client-aware credit scheduler, the authors propose some modifications to the algorithm proposed in [15] in order to make the credit scheduler client aware. In the client-aware credit scheduler algorithm, the authors propose that using the same mechanism proposed in [2], the guest domains reveal the device type, battery remaining, and distance of the client attributes of the requests in their run queues to the VMM. Every guest domain knows the client capabilities, whose requests are being serviced using the RIA downloaded on the client. Using this information, the VMM rearranges the domains in its run queue based on the battery life and distance from the client. The following section explains the scheduling of domains by the VMM.

Unlike the default credit scheduler, in the client-aware credit scheduler, two run queues are proposed at the VMM. The first run queue holds the requests from mobile clients and the second run queue holds requests from plugged in clients. In the proposed ordering of the domains in the run queues, first a virtual server that services request from a mobile client is pushed to the front end of the first run queue. When two or more domains are servicing requests from mobile clients, the battery remaining of the clients is checked. The virtual server servicing the request of a mobile client with the lowest battery life is present near the head of the queue for earlier execution. For example, if VM1, VM2**,** and VM3 are servicing the requests from mobile clients M1, M2, and M3 with battery remaining 65%, 55% and 80% respectively, then VM2 is run first, VM1 is run next, and VM3 is run last. This type of ordering of run queue ensures that clients having low battery are serviced before they can run out of charge.

The domains in the run queue that service requests from power plugged clients are ordered in the second run queue based on their distances from the server. A client that is farthest from the server is serviced before the client that is nearest to the server. This type of ordering ensures that a client farthest from the server does not incur high latency in waiting for the request to be serviced. The same domain might be servicing request**s** from both a mobile client and a plugged in client. In such a case, the domain will be present in both the run queues. The two run queues that hold the mobile and plugged-in client requests are proposed to be assigned to separate cores on the physical server for parallel execution.

4.3 Working of the Client Aware Credit Scheduler

Let us assume, for example, there are five virtual servers – v1, v2, v3, v4**,** and v5 that process the client requests from 5 clients – c1, c2, c3, c4, and c5. Table 1 explains the capabilities of the 5 clients. As mentioned earlier, for an IaaS and PaaS clouds, most of the request will be I/O. Hence, in this research, the author assumes that the clients have I/O requests. The client-aware credit scheduler works even if there are no I/O requests as well. In the above scenario, let us assume that the middleware adds the 10 requests from the clients into a high priority queue and sends it to the physical server. Since all the requests involve I/O, the VMM forwards the requests to the Driver domain. The Driver domain is also another domain and it is aware of the client capabilities that placed the requests though RIA.

The driver domain finds that R2, R3, R5, R6, R7, and R9 are from mobile clients and R2, R9 and R11 have a battery life lesser than 50%. Hence, the driver domain pushes the VCPU tasks generated into the run queues. If the request is from the mobile client they are pushed to the run queue operating only for mobile clients. If they belong to the plugged category, then they are pushed into the run queue operating only for plugged. The segregation is based on the type of client attributes and it is flexible once the capabilities are determined through RIA. The credits owned by virtual servers decide the time for which it executes.

The driver domain assigns the remaining requests to the respective guest domains. Using the mechanism implemented by the authors in [16, 17], the guest domains then reveal the client attributes, whose requests they are processing to the VMM. The information revealed to the VMM looks as shown in Table 1.

Table 1. Attribute List

Client Request Number	Designated Virtual Server	Type	Battery Remaining	Distance (Propagation Delay)
R1	1	Plugged Laptop	P	100
R2	3	Smart Phone	45%	65
R3	2	Mobile Phone	60%	45
R4	3	Desktop	P	150
R5	2	Smart Phone	55%	60
R6	2	PDA	70%	75
R7	1	Thin Client	75%	90
R8	2	Desktop	P	80
R9	1	Unplugged Tablet	40%	65
R10	4	Plugged Laptop	P	110
R11	4	Smart Phone	48%	90
R12	5	Plugged Laptop	P	85
R13	5	PDA	30%	105
R14	3	Mobile Phone	66%	120
R15	5	Desktop	P	100

The VMM finds with the help of the RIA downloaded, that all the domains except domain 4 have requests from mobile clients. It adds mobile client servicing domains into the first run queue from the head to the tail in ascending order of their battery life. Domain 5 is put at the head of the run queue, domain 2 is next, domain 3 is added next**,** and then domain 1 is added. Domain 4 will not be a part of the first run queue because it does not process a mobile client. The run queue at this instant is as shown in Figure 3.

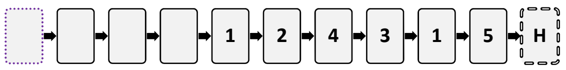

Figure 3. Run Queue for Battery devices

The VMM adds the domains servicing plugged in client requests into the second run queue. The domains in the second run queue are ordered based on their distances from the server. The domain servicing a client that is farthest is near the head of the run queue. For the case study, the second run queue is as shown in Figure 4.

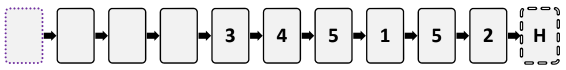

Figure 4. Run Queue for plugged in devices

As per the default credit scheduler, the run queue is rearranged every 30ms and hence, Table 1 gets refreshed every 30ms. It should be noted that the above Table 1 is dynamically updated as more requests arrive at the domains and decisions about scheduling changes depending on the requests that have arrived.

4.4 CAS Algorithm

The algorithms for respective modules incorporated in the design of CACS scheduler are given in the below sections.

4.4.1 VCPU Segregation to Dedicated Run Queue

```
If packet is from Plugged_source_client
    Push IO_VCPU to runQ_plugged
Elseif packet is from Battery_source_client
    Push IO_VCPU to runQ_battery
```

```
Elseif VCPU is not IO task
    Push to respective runQ depending on the task
```

4.4.2 Sorting the runQ_battery based on battery percentages

```
Sort_runQ_battery (battery_percent[ ], domains__to_run[ ])
{
Current_domain = Second domain in the run queue;
While battery_percent of current domain < battery_percent of next domain in the run queue
{
Scan the run queue
}
Swap the higher battery_percent domain with the current domain
}
```

4.4.3 Sorting the runQ_plugged based on distances

```
Sort_runQ_battery (distance[ ], domains__to_run[ ])
{
Current_domain = Second domain in the run queue;
While distance of current domain < distance of next domain in the run queue
{
Scan the run queue
}
Swap the higher distance domain with the current domain
}
```

Chapter 5

SIMULATIONS AND RESULTS

In this chapter a detailed analysis of the proposed system model has been done. The analysis includes mathematical reasoning classified in terms of various latencies involved in the server delay. It also includes simulation of scenario created with java socket programming. The simulation is done to support the theoretical reasoning.

5.1.1 System Model

The proposed client-aware scheduling model works in conjunction with the traditional hardware available. Furthermore, software enhancements made to the hypervisor work in conjunction with the client-aware scheduler model as well. When a packet reaches the Network Interface Controller (NIC) card of the real hardware of the Xen server hosting several VMs, it is first stored in the buffer of the NIC card to be processed further. The hypervisor interrupts the *dom 0* about the packet to be multiplexed to respective user domain. The *dom 0* identifies the packet and puts the packet into the circular buffer and intimates the respective user domain about the packet through the netfront and netback driver communication process as explained in chapter 2. Once the packet is identified by the virtual NIC of the user domain**,** a new interrupt is generated for the VCPU to run the task or instructions. The hypervisor takes the control and is responsible for putting the VCPU tasks into the run queue. As proposed, two special run queues work compared to the original model. Requests from battery operated clients are pushed into the *runQ_battery* and requests from plugged clients are pushed into the *runQ_plugged*, thus segregating the VCPUs according to the type of incoming packet. A dynamic table similar to

Table 1 listing the type of incoming packet, respective battery percentages, and distance is maintained by the hypervisor, which is actually shared from RIA configured to work in respective domains. This table is associated with the VCPU tasks generated and used to segregate and sort. The process flow of the proposed model is shown in Figure 5.

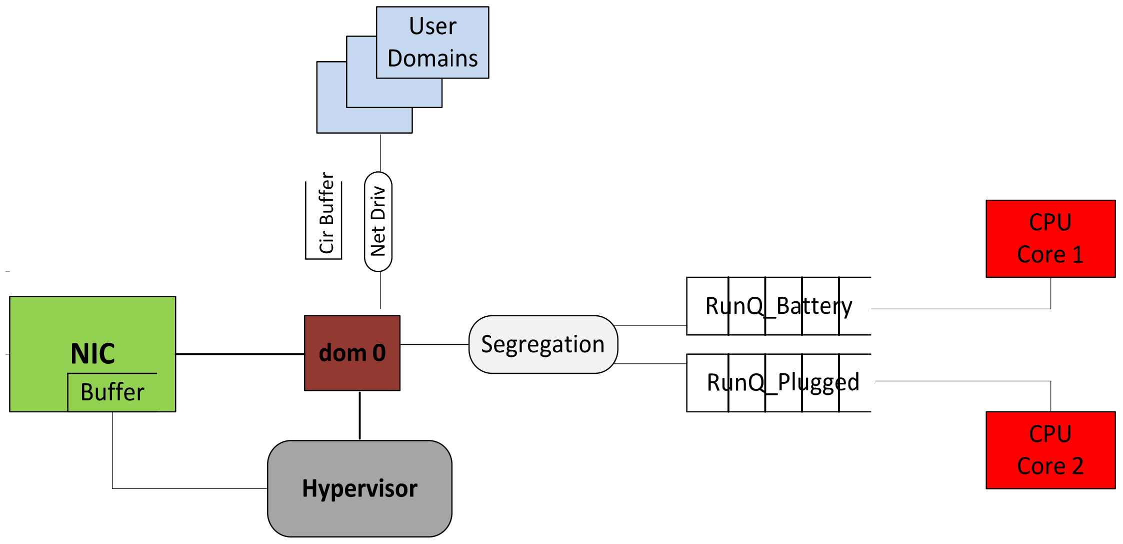

Figure 5. Process Flow for Client Aware Scheduler

As the clock ticks, the VCPU tasks are sorted in increasing order of battery percentages of the respective client requests in the *runQ_battery*. And *runQ_plugged* has the distant client at the head of the queue so that the client that is further away is served faster than the others at that instant of time. The VCPUs in the run queue are processed with the available credits accounted for individual VCPU. As the credits get replenished, they are again accounted for every 10ms to load balance the VMs.

5.2 Parameters

Several parameters are defined from the architecture of the system model. These parameters are used in the mathematical analysis of the thesis to prove the benefits of the model proposed. The parameters are listed below in the Table 2.

Table 2. List of Parameters

Symbol	Description
T_S	Time stamp at the client when a request is created and packet leaves the interface.
T_{Prop}	Propagation delay
N	Incoming Packets at the NIC card of the interface of the server at time T_s+T_{prop}
R_{NIC}	Incoming packet rate at the NIC card of the server interface at time T_s+T_{prop}. R_{NIC} = N packets/second
K	Packets before allocating to their respective run queues at time.
S_B	Size of the *runQ_battery*
S_P	Size of the *runQ_plugged*
S_I	Other non-IO packets.
R_{HYP}	Incoming packet rate at the hypervisor. R_{HYP} = K packets/second
T_{DA}	Time to allocate the packets to respective user domains by the hypervisor.
T_{Seg}	Time taken to segregate the VCPUs into the run queue based on type of packet.
T_{Sort}	Time taken to sort the run queue after every clock tick.
T_{QD}	Queuing delay or amount time a VCPU waits before getting to the core.
T_P	Processing delay for individual VCPU.
S_W	Size of the sorting window for every clock cycle.
BDR	Battery dissipation rate of the client in Joules per second.

Table 2. List of Parameters (continued)

$B_{\%}$	Battery percentage of the client
B_L	Battery level of a client in Joules.
T_R	Receive time stamp when reply is received by the client.
RTT	Round Trip Time of a packet – time interval between the request placed and corresponding reply received.
RTT_{sorted}	RTT calculated for the sorted run queue
$RTT_{unsorted}$	RTT calculated for the unsorted run queue (original model)

5.3 Mathematical Analysis

The best approach to calculate the energy consumed while waiting for a reply for a request placed to the server is RTT [35]. Let a battery-operated client place a service request for a server and the packet leaves the client at T_S and let T_R be the time at which the reply packet arrives at the client's interface. The time line for a request packet undergoing the service hosted by the server is shown in Figure 6. The time line is approximated to the events which cannot be considered as real values.

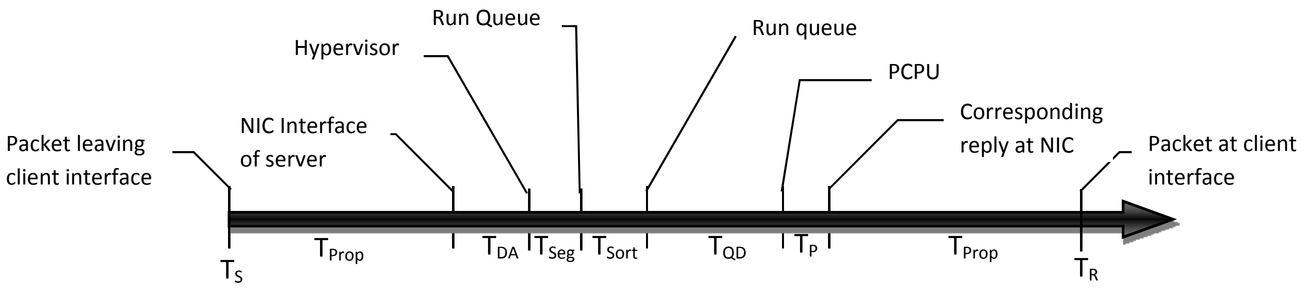

Figure 6. Timeline for packet flow

As the packet arrives at the NIC card it undergoes the process explained in section 5.2**,** and many functional delays are added [36]. The receiving time T_R can be calculated as shown below in the equation (7):

$$\mathrm{T_R = T_S + T_{DA} + T_{Seg} + T_{Sort} + T_{QD} + T_P + 2T_{Prop}} \tag{7}$$

$$\mathrm{RTT = T_R - T_S} \tag{8}$$

The total servicing delay for a request to be processed is given by equation (9).

$$Service\ Delay = \mathrm{T_{DA} + T_{Seg} + T_{Sort} + T_{QD} + T_P} \tag{9}$$

$$\mathrm{RTT} = Service\ Delay + \mathrm{2T_{Prop}} \tag{10}$$

Analyzing the equation (7), the T_{DA} depends on the rate R_{HYP}. R_{HYP} is proportional to the rate at which packets arrive at the interface, the buffer space allocated and also rate at which the buffer is read. The relation is shown in equation (11):

$$\mathrm{R_{HYP} \propto R_{NIC}} \tag{11}$$

If N packets arrive at the NIC card per second and K packets are read out of the buffer**,** the relation can be shown in the equation (12) which is derived from equation (11).

$$\mathrm{K \leq N} \tag{12}$$

Hence**,** domain allocation is a function of packet arrival rate at the interface and hypervisor as shown below in equation (13).

$$\mathrm{T_{DA} = Fn\,\{R_{HYP}, R_{NIC}\}} \tag{13}$$

5.3.1 Segregation Delay T_{Seg}

The time taken to segregate the tasks is independent of the packet arrival rate and buffer read rate since the packets are segregated only after an interrupt is placed by the user domains in the event channel notifier. Events are read one after the other as explained in section 2.2.3. The events are created with the set bits corresponding to the request. The type of packet is shared with hypervisor and the VCPUs are pushed into the respective queues. The T_{Seg} is negligible compared to other delays incurring at the various levels of processing and is not considered in our mathematical analysis.

5.3.2 Sorting Delay T_{Sort}

Once the VCPUs are pushed into respective queues, the sorting takes place. The sorting is triggered to happen for every clock cycle so that appropriate load balancing could be done to the run queue. The range for the number of packets to be sorted is given below.

$$\text{Packets to be sorted} \quad = \begin{cases} min = 0 \ (if \ \mathrm{R_{NIC}} \ is \ 0) \\ max \ = S_W \ which \ is \ fn \ K \end{cases}$$

T_{Sort} can be considered in the RTT to be a constant since it is the time taken to run the sorting algorithm on the [0, K] VCPU tasks during one clock cycle.

5.3.3 Queuing Delay T_{QD}

After sorting the VCPU tasks, the domains have to wait for the T_{QD} amount time to get processed by the allocated PCPU core. The waiting time for each VCPU is a function of the number of VCPUs tasks in the sorting window and the processing time of each VCPU. The waiting time for nth packet in the sorting window is $(n-1)T_P$ as written in equation (15). The processing time for each request is assumed to be the same for our thesis.

$$T_{QD} = (n-1)T_P \tag{15}$$

5.3.4 Processing Delay T_P

The processing delay is an individual parameter and it is independent of Sorting delay, Domain allocation delay, and segregation delay. The only factors affecting this delay is the time taken to execute the set of instructions on PCPU and the number of credits left for the processor to execute these instructions.

All of the above delays, which influence the RTT of service request, are used in the thesis work and mathematical calculations are performed over various conditions to understand the advantages leveraged for the various genres of clients.

5.4.1 RTT vs. Battery% Graph

Assuming that the battery operated client sends the packet at time $T_S = 0$ and T_R would be the total round trip time for the request to be serviced and reach back to the client. The new equation for RTT_{sorted} is given in equation (16) after neglecting T_{DA}, T_{Seg} and T_{Sort}. These delays are ignored because they have minimal effect on the final equation.

$$RTT_{sorted} = T_{QD} + T_P + 2T_{Prop} \tag{16}$$

Uniform random values of inputs are tabulated in an excel sheet and are used as inputs of the equation (16) to manually calculate the $RTT_{unsorted}$ and RTT_{sorted}. The Figure 7 shows the plot for RTT against battery percentage with constant propagation delay.

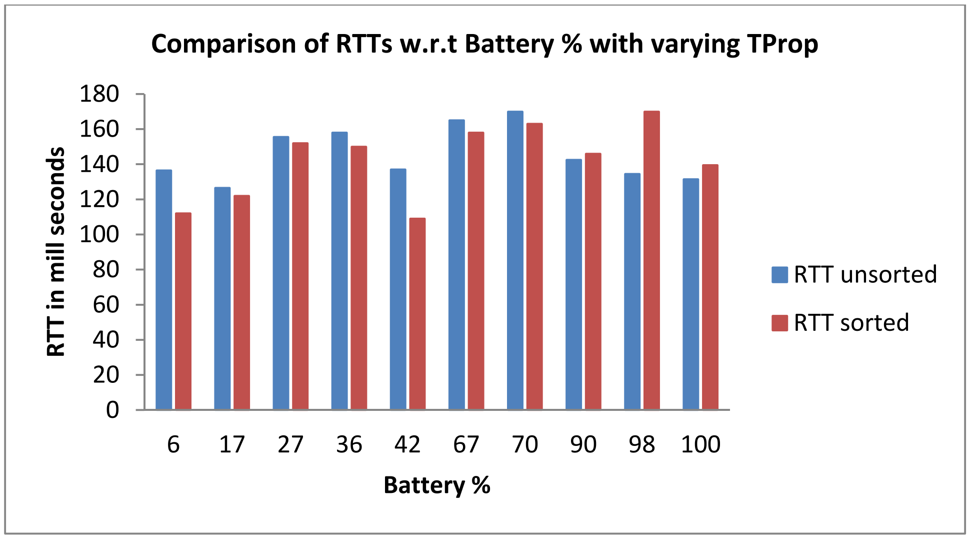

Figure 7. Comparison of RTTs w.r.t Battery % with varying TProp

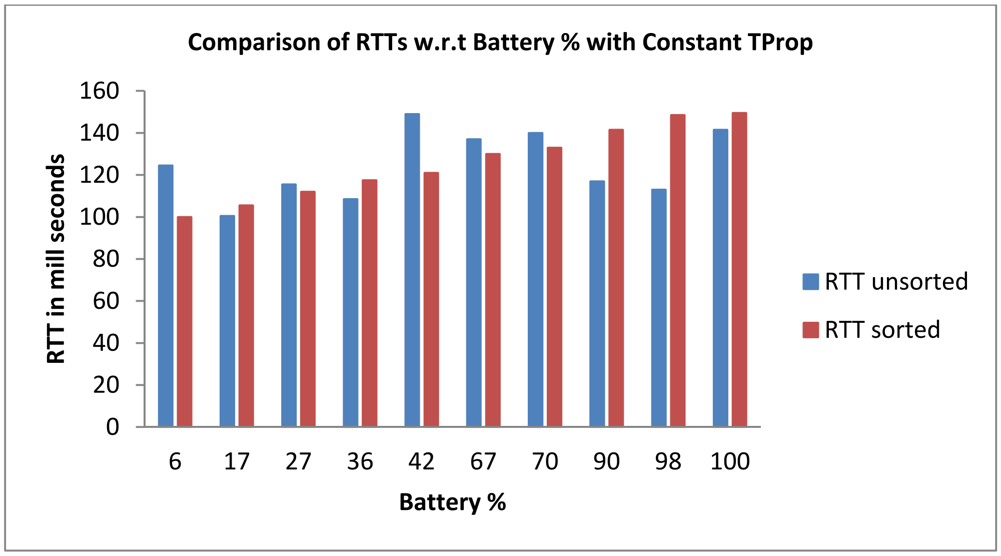

Figure 8. Comparison of RTTs w.r.t Battery % with constant TProp

The plot for RTT against the battery percentage for varying propagation delay is shown in Figure 7. In the above plots**,** the X-axis of the above graph shows the discrete battery percentages of clients sending packets to the server NIC card interface at time $T_S = 0$**,** and Y-axis is the round trip time taken to get back to the client. It can be understood that when client-aware scheduler is used the round trip time of the packets are reduced for the device with low battery percentages. The constant propagation delay in Figure 8 gives a much better example of a linear increment positive slope with the increasing battery percentage. Table 3. shows the values used for plotting the graphs.

Table 3. Values for RTT vs. Battery Percentages

Battery Percentages	Varying T_{prop}		Constant T_{prop}	
	RTT Unsorted	RTT Sorted	RTT Unsorted	RTT Sorted
6	136.5	112	124.5	100
6	121.5	88.5	133.5	100.5
7	127.5	125	103.5	101
9	145.5	129.5	117.5	101.5
10	145	122	125	102
10	155	130.5	127	102.5
10	162.5	135	130.5	103
12	119.5	121.5	101.5	103.5
13	147.5	138	113.5	104
15	140.5	122.5	122.5	104.5
16	165	123	147	105
17	146.5	151.5	100.5	105.5
17	126.5	122	110.5	106
18	112	106.5	112	106.5
19	98.5	101	104.5	107
19	178.5	141.5	144.5	107.5
21	149.5	134	123.5	108
22	128	108.5	128	108.5
23	145.5	111	143.5	109
24	172.5	149.5	132.5	109.5
24	143.5	116	137.5	110
26	115	124.5	101	110.5
26	157	129	139	111
26	176.5	139.5	148.5	111.5
27	155.5	152	115.5	112
28	109.5	114.5	107.5	112.5
29	98	109	102	113
30	114.5	109.5	118.5	113.5
31	142.5	130	126.5	114
31	178.5	156.5	136.5	114.5

Table 3. Values for RTT vs. Battery Percentages (continued)

32	165.5	133	147.5	115
33	142.5	143.5	114.5	115.5
34	109	116	109	116
35	158	158.5	116	116.5
35	132.5	115	134.5	117
36	128.5	137.5	108.5	117.5
36	158	150	126	118
38	126	134.5	110	118.5
38	113	121	111	119
39	113.5	111.5	121.5	119.5
41	106	122	104	120
42	127	128.5	119	120.5
42	137	109	149	121
43	97	115.5	103	121.5
44	122	138	106	122
44	153	140.5	135	122.5
46	165	145	143	123
47	127.5	141.5	109.5	123.5
48	178.5	156	146.5	124
49	145	136.5	133	124.5
52	90	115	100	125
53	136	141.5	120	125.5
54	134.5	144	116.5	126
55	131	136.5	121	126.5
55	134.5	133	128.5	127
58	159	171.5	115	127.5
59	155	160	123	128
62	129.5	146.5	111.5	128.5
62	160	151	138	129
63	102.5	129.5	102.5	129.5
67	165	158	137	130
69	144	156.5	118	130.5
69	143.5	155	119.5	131
69	150	157.5	124	131.5
69	133.5	130	135.5	132
70	184	180.5	136	132.5
70	170	163	140	133
71	167.5	171.5	129.5	133.5
72	163.5	148	149.5	134
73	163	166.5	131	134.5
74	156	145	146	135

Table 3. Values for RTT vs. Battery Percentages (continued)

75	168	171.5	132	135.5
75	145.5	142	139.5	136
78	106	134.5	108	136.5
79	133	163	107	137
79	167.5	173.5	131.5	137.5
80	154.5	172	120.5	138
80	159.5	172.5	125.5	138.5
80	155	149	145	139
81	138	135.5	142	139.5
82	124.5	158	106.5	140
88	153.5	166.5	127.5	140.5
89	183	183	141	141
90	171	195.5	117	141.5
90	142.5	146	138.5	142
91	167.5	164.5	145.5	142.5
92	136	165	114	143
92	162	157.5	148	143.5
93	149	164	129	144
93	172.5	176.5	140.5	144.5
93	146	147	144	145
94	150	173.5	122	145.5
94	154	166	134	146
94	136.5	140.5	142.5	146.5
95	156	173	130	147
96	119.5	161.5	105.5	147.5
98	134.5	170	112.5	148
98	151	186.5	113	148.5
100	111	155	105	149
100	131.5	139.5	141.5	149.5

5.4.2 RTT vs. Propagation Delay Graph

The plot in Figure 9 shows the RTT against distances for varying propagation delay. The distance parameter considered for this analysis is the propagation delay of one way from client to server. The propagation delay for each packet is accounted and fed to the client aware scheduler model to sort the VCPU tasks in the plugged run queue. Thus sorting function is run in the descending in order to serve the farthest client at a given instant of time.

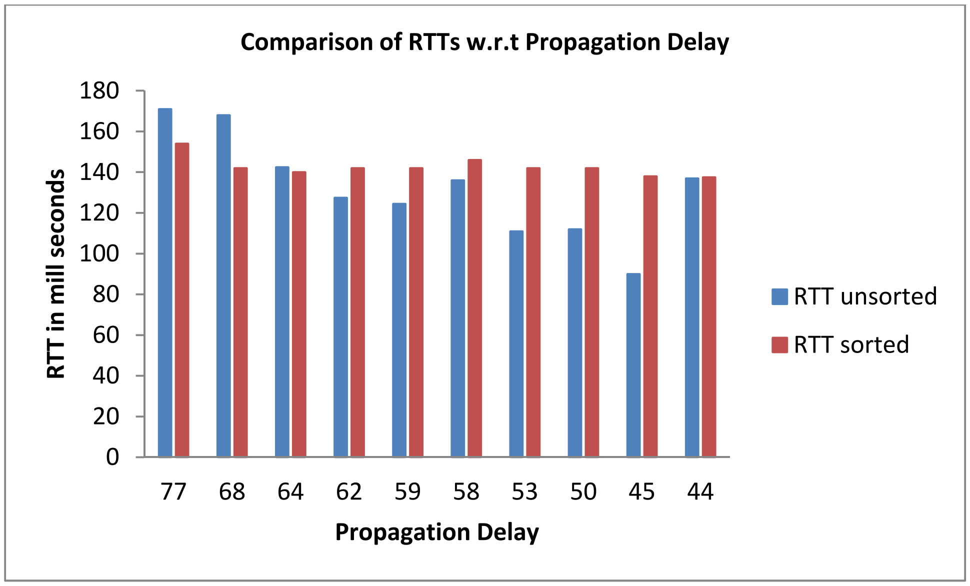

Figure 9. Comparison of RTTunsorted and RTTsorted w.r.t Propagation Delay

The X-axis of the above graph shows propagation delays for clients sending packets to the server at time $T_S = 0$ and the Y-axis represents the round trip time taken to get back to the client. The red plot shows the round trip time for the client-aware scheduler and is compared with the blue plot, which represents the original first-come first-serve (FCFS) basic model. It is evident that the RTT values decrease with decreasing the propagation delays. Table 4. shows the values used to plot the graph.

Table 4. Values for RTT vs. Propagation Delay

Battery Percentages	Varying T_{prop}	
	RTT Unsorted	RTT Sorted
77	171	154
74	184	148.5
73	146.5	147
72	159	145.5
71	158	144
71	178.5	144.5
71	183	145
70	155.5	143.5
70	172.5	144
69	151	142.5
69	167.5	143
68	167.5	141.5
68	168	142
67	147.5	140.5
67	154.5	141
67	159.5	141.5
67	178.5	142
66	155	140.5
66	158	141
66	162.5	141.5
66	163	142
66	172.5	142.5
66	178.5	143
65	170	141.5
64	142.5	140
64	145.5	140.5
64	150	141
64	155	141.5
64	165	142
64	176.5	142.5
63	133	141
63	144	141.5
63	149.5	142
63	150	142.5
63	153.5	143
63	156	143.5

Table 4. Values for RTT vs. Propagation Delay (continued)

62	127.5	142
62	143.5	142.5
61	134.5	141
61	136	141.5
61	160	142
61	165	142.5
61	167.5	143
60	128.5	141.5
60	145	142
60	149	142.5
60	154	143
59	119.5	141.5
59	124.5	142
59	127.5	142.5
59	129.5	143
59	134.5	143.5
59	140.5	144
59	153	144.5
59	157	145
59	165	145.5
59	165.5	146
58	122	144.5
58	126	145
58	126.5	145.5
58	136	146
58	142.5	146.5
57	115	145
57	119.5	145.5
57	162	146
57	163.5	146.5
56	136.5	145
56	145	145.5
55	131	144
55	155	144.5
55	156	145
54	127	143.5
53	111	142
53	134.5	142.5
53	143.5	143

Table 4. Values for RTT vs. Propagation Delay (continued)

53	145.5	143.5
52	142.5	142
51	106	140.5
51	109.5	141
51	113	141.5
51	145.5	142
51	146	142.5
50	102.5	141
50	109	141.5
50	112	142
50	128	142.5
49	106	141
49	132.5	141.5
49	133.5	142
48	98	140.5
48	114.5	141
48	138	141.5
47	97	140
47	98.5	140.5
47	136.5	141
46	113.5	139.5
45	90	138
45	131.5	138.5
44	121.5	137
44	137	137.5

5.4.3 RTT vs. Battery Dissipation Rate Graph

The client-aware scheduling model, if it includes the BDR, can show better performance benefits in terms of lowering the battery consumption rates. A. Carroll in [22] researched on the battery consumption of a mobile device and came up with close values for BDR. It is understood that a mobile device has a delta of dissipation rate between 144 – 166 milli-watts while listening on the network. The author in this thesis uses the range in conjunction with the client-aware scheduler to understand the behavior of the system.

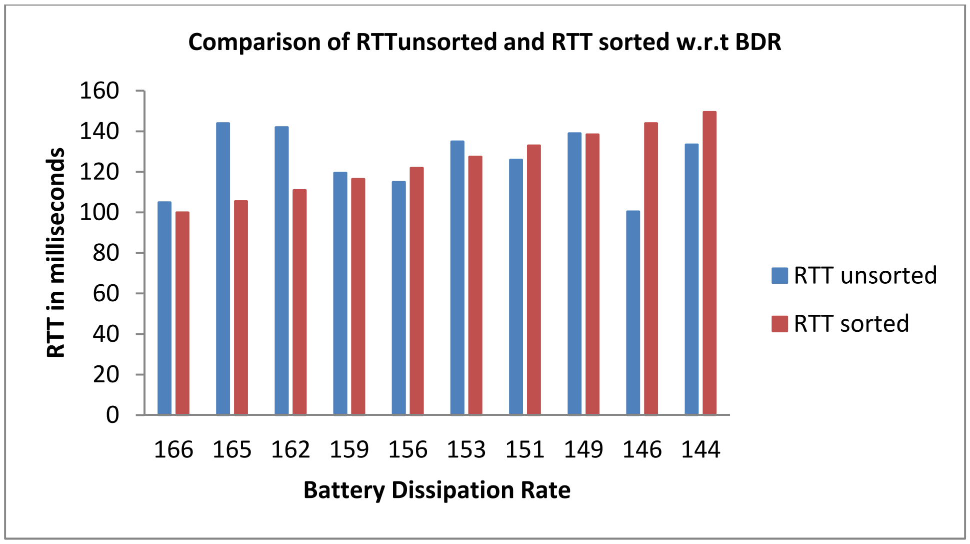

Figure 10. Comparison of $RTT_{unsorted}$ and RTT_{sorted} w.r.t BDR

The X-axis of the above graph is BDRs of clients sending packets to the server at time $T_S = 0$, and the Y-axis is the round trip time taken to get back to the client. The red plot shows the round trip time for client-aware scheduler and it is compared with the blue plot which represents the (FCFS) basic model. Propagation delay is constant to get closer aspects of the results. It is understood that the RTT increases with the increasing battery percentages because of the client aware scheduler in effect. Table 5. shows the values used to plot the graph.

Table 5. Values for RTT vs. BDR

Battery Dissipation Rate	Varying T_{prop}	
	RTT Unsorted	RTT Sorted
166	105	100
166	112.5	100.5
166	113	101
166	141.5	101.5
165	105.5	102
165	122	102.5
165	129	103
165	130	103.5

Table 5. Values for RTT vs. BDR (continued)

165	134	104
165	140.5	104.5
165	142.5	105
165	144	105.5
165	148	106
164	114	106.5
164	117	107
164	138.5	107.5
164	141	108
164	145.5	108.5
163	127.5	109
162	106.5	109.5
162	120.5	110
162	125.5	110.5
162	142	111
161	107	111.5
161	108	112
161	131.5	112.5
161	145	113
160	131	113.5
160	132	114
160	139.5	114.5
160	146	115
160	149.5	115.5
159	118	116
159	119.5	116.5
159	124	117
159	129.5	117.5
159	135.5	118
159	136	118.5
159	140	119
158	102.5	119.5
158	137	120
157	111.5	120.5
157	123	121
157	138	121.5
156	115	122
156	121	122.5
156	128.5	123

Table 5. Values for RTT vs. BDR (continued)

155	100	123.5
155	116.5	124
155	120	124.5
154	109.5	125
154	133	125.5
154	146.5	126
153	103	126.5
153	106	127
153	135	127.5
153	143	128
152	104	128.5
152	111	129
152	119	129.5
152	121.5	130
152	149	130.5
151	108.5	131
151	109	131.5
151	110	132
151	116	132.5
151	126	133
151	134.5	133.5
150	114.5	134
150	118.5	134.5
150	126.5	135
150	136.5	135.5
150	147.5	136
149	101	136.5
149	102	137
149	107.5	137.5
149	115.5	138
149	139	138.5
149	148.5	139
148	128	139.5
148	132.5	140
148	137.5	140.5
148	143.5	141
147	104.5	141.5
147	110.5	142
147	112	142.5

Table 5. Values for RTT vs. BDR (continued)

147	123.5	143
147	144.5	143.5
146	100.5	144
146	113.5	144.5
146	122.5	145
146	147	145.5
145	101.5	146
145	117.5	146.5
145	125	147
145	127	147.5
145	130.5	148
144	103.5	148.5
144	124.5	149
144	133.5	149.5

5.5 Simulations with Java Socket Programming

Java Socket Programming is used to write the network application programs [36]. To run the network applications, client and server codes are written and together they create client and server environment. Java is simple and it is helpful when writing neat and understandable code of network applications.

As a profound approach to support the theoretical calculations, simulations have been performed using the Java Socket Programming. Based on the CAS algorithm in section 4.4 a java program has been written which nearly simulates the proposed model. The authors have accounted the majority of the parameters which nearly approximates the working of the original Xen client server architecture. The client program generates 100 packets by creating 100 threads which equivalent to 100 clients sending packets at time T_S. The packets contain the battery percentages and the distance as the payload. The server program, which listens for the traffic receives the packets in a first-come first-serve basis and are all put into a buffer space. Then, each packet is read from the buffer and pushed into a new buffer where packets are sorted for every clock cycle of 5 milliseconds. To add the processing delay, a random generator function generates random processing delays T_P f. The program has to process the packet and return a reply to respective client.

In order to show the processing time, the program is made to stay idle for T_P amount of time for every packet. Thus, the timestamp of the reply packet is accounted and plotted against the normal non-sorting model. Figure 11 shows the plot for $RTT_{unsorted}$ and RTT_{sorted} against Battery %. It is understood from the plot that client with lower battery levels are served before the others. A much better understanding can be obtained from the service time vs. battery % graph as show in Figure 12. Service time is function of queuing delay. It is clearly evident that the queuing delay increases with the increasing battery percentages and compared to the random times of the unsorted version. The energy consumed by the clients while waiting for the requests are shown in Figure 13. Table 6. shows the values used to plot the respective graphs.

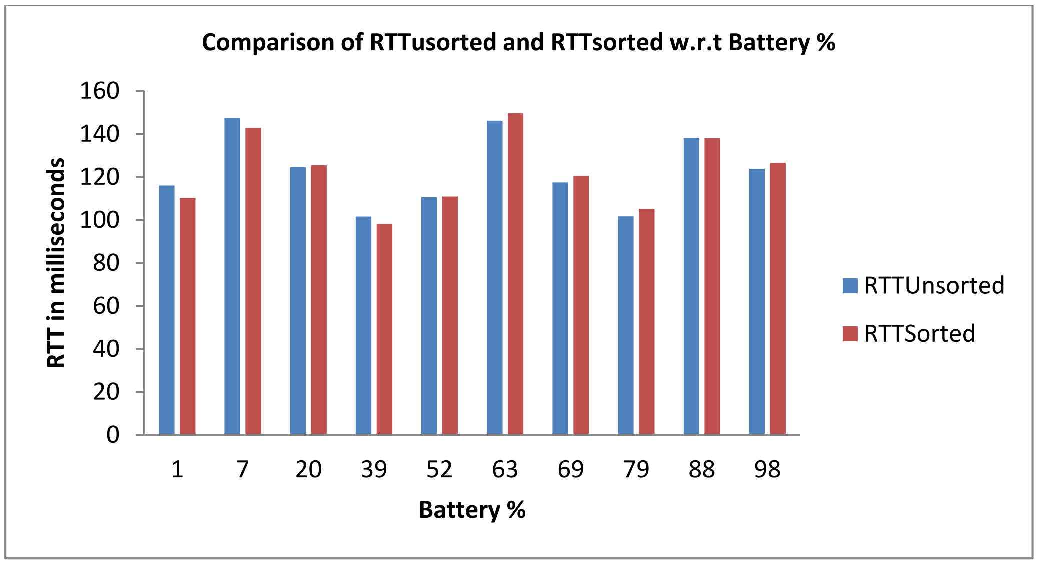

Figure 11. Comparison of RTTUsorted and RTTsorted w.r.t Battery % (Simulated)

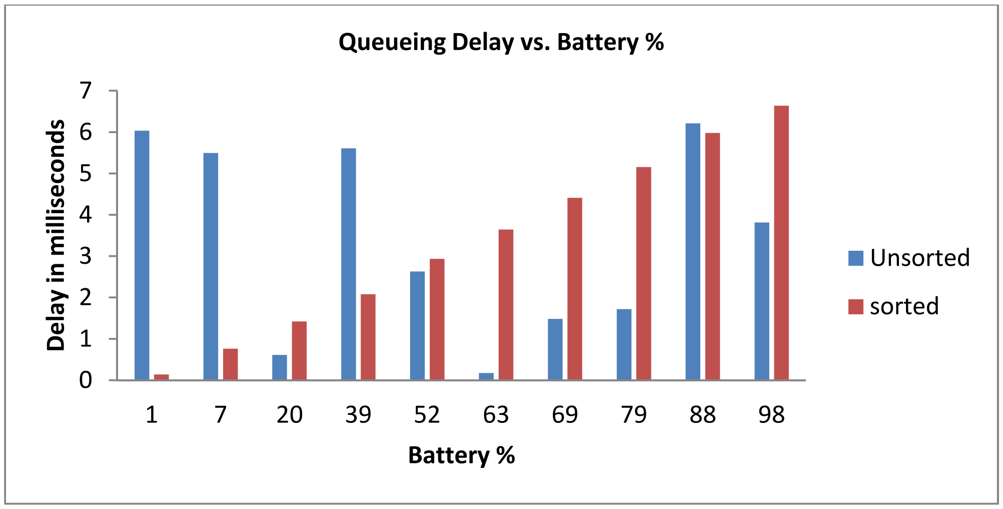

Figure 12. Comparison of Service Times w.r.t Battery % (Simulated)

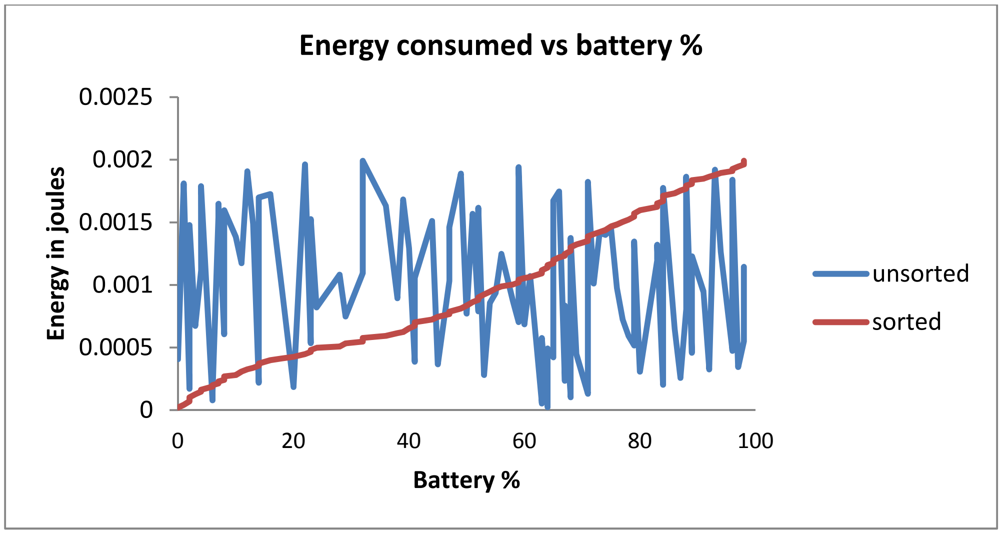

Figure 13. Energy Consumed Waiting to be Served

Table 6. Values for Simulated $RTT_{Unsorted}$ and RTT_{Sorted}

Battery Percentages	**Varying T_{prop} in Milliseconds**		**Queuing Delays in Milliseconds**		**Energy consumed In Joules**	
	RTT Unsorted	**RTT Sorted**	**RTT Unsorted**	**RTT Sorted**	**RTT Unsorted**	**RTT Sorted**
0	101.3544	100.0674	1.35436	0.067407	0.000406	2.02E-05
1	116.0324	110.1385	6.032366	0.138458	0.00181	4.15E-05
2	94.56704	94.23302	0.567037	0.233024	0.00017	6.99E-05
2	121.0399	118.2962	3.039875	0.296227	0.000912	8.89E-05
2	146.9243	142.3409	4.924297	0.340906	0.001477	0.000102
3	144.2425	142.422	2.24251	0.422021	0.000673	0.000127
4	109.7172	106.4898	3.717228	0.489796	0.001115	0.000147
4	139.9613	134.5388	5.961316	0.538816	0.001788	0.000162
6	114.2602	114.6255	0.260192	0.625474	7.81E-05	0.000188
6	100.4725	100.6626	0.472471	0.662592	0.000142	0.000199
7	109.1319	104.7116	5.131853	0.711645	0.00154	0.000213
7	147.4937	142.7625	5.493687	0.762524	0.001648	0.000229
8	130.0211	128.8041	2.021106	0.804082	0.000606	0.000241
8	111.321	106.9009	5.320979	0.900871	0.001596	0.00027
10	124.6141	120.9327	4.614068	0.93273	0.001384	0.00028
11	141.9093	139.0278	3.909305	1.027804	0.001173	0.000308
12	124.3572	119.088	6.357217	1.087965	0.001907	0.000326
13	132.96	129.1237	4.960014	1.123683	0.001488	0.000337
14	94.72926	95.1819	0.729264	1.181902	0.000219	0.000355
14	99.66284	95.23872	5.662841	1.238724	0.001699	0.000372
16	127.7521	123.328	5.752129	1.328013	0.001726	0.000398
18	129.201	127.3748	3.200951	1.374753	0.00096	0.000412
20	124.6119	125.4196	0.611874	1.419589	0.000184	0.000426
22	94.53917	89.49109	6.539173	1.491094	0.001962	0.000447
23	123.7813	123.5528	1.781348	1.55278	0.000534	0.000466
23	127.0828	123.6001	5.082799	1.600056	0.001525	0.00048
24	118.7371	117.6564	2.737067	1.656353	0.000821	0.000497
28	129.603	127.6932	3.603037	1.69315	0.001081	0.000508
29	128.4927	127.7754	2.492655	1.775426	0.000748	0.000533
32	103.6495	101.8218	3.649453	1.821843	0.001095	0.000547
32	120.6371	115.9197	6.637068	1.919738	0.001991	0.000576
36	137.4428	133.9774	5.442809	1.977382	0.001633	0.000593
38	130.9767	130.0502	2.976671	2.050201	0.000893	0.000615
39	101.606	98.08066	5.606018	2.080661	0.001682	0.000624
40	140.3084	138.1769	4.308444	2.17693	0.001293	0.000653
41	121.287	122.2471	1.286953	2.247121	0.000386	0.000674

Table 6. Values for Simulated $RTT_{Unsorted}$ and RTT_{Sorted} (continued)

41	119.5334	118.3376	3.533418	2.337596	0.00106	0.000701
44	111.0355	108.4131	5.035523	2.413104	0.001511	0.000724
45	99.21676	100.4854	1.216762	2.485385	0.000365	0.000746
47	135.4429	134.558	3.442944	2.557976	0.001033	0.000767
47	152.8796	150.6211	4.879618	2.621132	0.001464	0.000786
49	124.2971	120.7057	6.297056	2.705739	0.001889	0.000812
50	98.57052	98.78361	2.570524	2.783608	0.000771	0.000835
51	123.2242	120.8759	5.224191	2.875946	0.001567	0.000863
52	110.6308	110.9362	2.630774	2.936196	0.000789	0.000881
52	135.3852	133.0004	5.385164	3.000381	0.001616	0.0009
53	116.9336	119.0791	0.933596	3.079088	0.00028	0.000924
54	112.8553	113.1531	2.855292	3.153133	0.000857	0.000946
55	135.1188	135.2321	3.118819	3.232077	0.000936	0.00097
56	136.1585	135.2953	4.158457	3.295289	0.001248	0.000989
58	94.90385	95.34385	2.903853	3.34385	0.000871	0.001003
59	156.3422	157.402	2.342212	3.402014	0.000703	0.001021
59	134.4677	131.4719	6.467669	3.47194	0.00194	0.001042
60	120.284	121.5135	2.284047	3.513477	0.000685	0.001054
61	131.5662	131.5463	3.566239	3.546298	0.00107	0.001064
63	146.1735	149.6435	0.173535	3.643465	5.21E-05	0.001093
63	135.917	137.722	1.91698	3.721951	0.000575	0.001117
64	90.07637	93.79832	0.076368	3.798318	2.29E-05	0.001139
64	119.6388	121.8591	1.638828	3.859137	0.000492	0.001158
65	119.4071	121.9119	1.407079	3.911855	0.000422	0.001174
65	95.57556	93.99373	5.575558	3.993726	0.001673	0.001198
66	107.8218	106.0634	5.821765	4.063362	0.001747	0.001219
67	106.7792	110.1133	0.77921	4.113307	0.000234	0.001234
67	136.7812	138.1575	2.781247	4.157488	0.000834	0.001247
68	118.3414	122.2387	0.341403	4.238699	0.000102	0.001272
68	92.58221	92.33615	4.582209	4.336145	0.001375	0.001301
69	117.4807	120.4098	1.480738	4.409805	0.000444	0.001323
71	96.43535	100.5038	0.435353	4.503754	0.000131	0.001351
71	146.0995	148.5821	2.099474	4.582122	0.00063	0.001375
71	128.0754	126.6252	6.075445	4.6252	0.001823	0.001388
72	137.3704	138.6887	3.370353	4.688716	0.001011	0.001407
73	122.7204	122.7425	4.720351	4.742455	0.001416	0.001423
74	102.6666	102.795	4.666612	4.794999	0.0014	0.001438
75	102.8165	102.8911	4.816462	4.891111	0.001445	0.001467
76	115.2514	116.9415	3.251382	4.941541	0.000975	0.001482

Table 6. Values for Simulated $RTT_{Unsorted}$ and RTT_{Sorted} (continued)

77	130.4104	133.0097	2.410378	5.009708	0.000723	0.001503
78	123.9795	127.0723	1.979548	5.072277	0.000594	0.001522
79	101.7197	105.1531	1.719663	5.153112	0.000516	0.001546
79	116.4848	117.2346	4.484763	5.234558	0.001345	0.00157
80	119.0217	123.3226	1.021669	5.322631	0.000307	0.001597
83	129.9968	131.4101	3.996762	5.410088	0.001199	0.001623
83	144.4033	145.505	4.403317	5.50496	0.001321	0.001651
84	102.671	107.5641	0.671045	5.564132	0.000201	0.001669
84	123.3068	125.6196	3.306837	5.619586	0.000992	0.001686
84	107.9123	107.7101	5.912296	5.710117	0.001774	0.001713
86	142.1614	145.772	2.161395	5.772037	0.000648	0.001732
87	114.8549	119.8477	0.854889	5.847716	0.000256	0.001754
88	126.6808	129.8977	2.680769	5.897712	0.000804	0.001769
88	138.2124	137.9792	6.212449	5.979205	0.001864	0.001794
89	117.5271	122.0255	1.527059	6.025526	0.000458	0.001808
89	136.0952	138.124	4.095246	6.124009	0.001229	0.001837
91	129.1542	132.1594	3.154212	6.159402	0.000946	0.001848
92	127.0806	132.2184	1.080624	6.218357	0.000324	0.001866
93	120.3977	120.2589	6.397743	6.258884	0.001919	0.001878
94	140.2122	142.3126	4.212176	6.312602	0.001264	0.001894
96	103.578	108.3636	1.578009	6.363552	0.000473	0.001909
96	116.131	116.4191	6.130956	6.419063	0.001839	0.001926
97	103.1445	108.4829	1.144481	6.48292	0.000343	0.001945
98	139.8385	144.5401	1.838494	6.540066	0.000552	0.001962
98	123.8142	126.6371	3.81423	6.637068	0.001144	0.001991

Chapter 6

CONCLUSION AND FUTURE WORK

This research analyzes the performance of the Xen credit scheduler in a cloud. The default credit scheduler follows a fair allocation policy to all the VMs and does not consider the priorities of requests. This mechanism is unsuitable in a cloud where some requests require preferential treatment over the others. The authors proposed two enhancements to the credit scheduler, segregating the requests into two run queues based on the type of request and then sorting the run queues according to the client capabilities. These optimizations decrease the latencies suffered by high priority client**s** which require a faster service on a cloud environment. The information about client capabilities such as battery level, battery dissipation rate, distance from the server**,** and many applicable parameters such as battery percentages, distances from the client are delivered to the hypervisor by the RIA technology. This information is maintained in a dynamic table and is shared between the hypervisor and *dom 0* to do the necessary sorting. To support the research work done**,** both mathematical calculations and simulations have been performed. The results obtained show desired outputs in a normal environment with hundreds of clients requesting services from the server.

The condition with empty credits after sorting has to be addressed and is left for future work; a method to get the solution using credit stealing is possible. The future work includes the deployment of the application on a real hardware which involves the altering the kernel of the Xen hypervisor and developing the workable Credit Aware Scheduler for it.

REFERENCES

LIST OF REFERNCEES

[1] J. E. Smith and N. Ravi, "The architecture of virtual machines," *Computer,* vol. 38, pp. 32-38, 2005.

[2] Applying Client-aware Technologies for Desktop Virtualization and Cloud Services. White Paper, Intel, 2011.

[3] Moving to Virtualization: A Guide to What's Possible from VMware., White paper, *VMWare*, 2010.

[4] Rosenblum, M.; Garfinkel, T., "Virtual machine monitors: current technology and future trends," *Computer* , vol.38, no.5, pp. 39- 47, May 2005, doi: 10.1109/MC.2005.176

[5] Goldberg, Robert P., "Architectural Principles for Virtual Computer Systems". *Harvard University*. pp. 22–26. Retrieved 2010-04-12.

[6] Saransh Mittal,. “Differentiated Network QoS in Xen”. *IIT Bombay*, July 2009.

[7] Understanding Full Virtualization, Paravirtualization and Hardware Assist., White paper, *VMWare*, Nov 2007.

[8] Oliver Garraux, Information Technology Fundamentals, How Virtualization Works and its Effects on IT, Oct 2007.

[9] David Chisnall, The Definitive Guide to the Xen Hypervisor, Prentice Hall PTR, Nov 2007.

[10] M. Bourguiba, K. Haddadou, and G. Pujolle, "Evaluating and Enhancing Xen-Based Virtual Routers to Support Real-Time Applications," in Consumer Communications and Networking Conference (CCNC), 2010 7th IEEE, 2010, pp. 1-5.

[11] David E. Williams, Juan R. García, Virtualization with Xen: Including XenEnterprise, XenServer, and XenExpress, Elsevier Science, 2007.

[12] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, et al., "Xen and the art of virtualization," SIGOPS Oper. Syst. Rev., vol. 37, pp. 164-177, 2003.

[13] K. J. Duda and D. R. Cheriton, "Borrowed-virtual-time (BVT) scheduling: supporting latency-sensitive threads in a general-purpose scheduler," SIGOPS Oper. Syst. Rev., vol. 33, pp. 261-276, 1999.

[14] L. Cherkasova, D. Gupta, and A. Vahdat, "Comparison of the three CPU schedulers in Xen," SIGMETRICS Perform. Eval. Rev., vol. 35, pp. 42-51, 2007.

[15] D. Ongaro, A. L. Cox, and S. Rixner, "Scheduling I/O in virtual machine monitors," presented at the Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments, Seattle, WA, USA, 2008.

[16] Moblin Platform Awareness Service, http://software.intel.com/en-us/articles/moblin-platform-awareness-service/., browsed 04/25/2012 at 16.25PM.

[17] Laszlo: An Open Source Framework for Rich Internet Applications, http://today.java.net/pub/a/today/2005/03/22/laszlo.html, browsed on 06/21/2012 at 09:20 AM

[18] Allaire, J.Macromedia Flash MX - A next-generation rich client. Macromedia, 2002

[19] Mobile Cloud Computing Forum, http://www.cloudcomputinglive.com/events/170-mobile-cloud-computing-forum.html, 2010, browsed on 08/21/2012 at 9:50 PM.

[20] Hoang T. Dinh, Chonho Lee, Dusit Niyato, and Ping Wang., "A Survey of Mobile Cloud Computing: Architecture, Applications, and Approaches". *Wireless Communications and Mobile Computing*, doi:10.1002/wcm.1203.

[21] A. Rudenko, P. Reiher, G. J. Popek, and G. H. Kuenning, "Saving portable computer battery power through remote process execution," SIGMOBILE Mob. Comput. Commun. Rev., vol. 2, pp. 19-26, 1998.

[22] A. Carroll and G. Heiser, "An analysis of power consumption in a smartphone," presented at the Proceedings of the 2010 USENIX conference on USENIX annual technical conference, Boston, MA, 2010.

[23] Y. Liu, L. Guo, F. Li, and S. Chen, "An empirical evaluation of battery power consumption for streaming data transmission to mobile devices," presented at the Proceedings of the 19th ACM international conference on Multimedia, Scottsdale, Arizona, USA, 2011.

[24] Enabling Emerging Enterprise Usages with Client-Aware Technologies, White Paper, Intel, Feb 2012.

[25] G. Liao, D. Guo, L. Bhuyan, and S. R. King, "Software techniques to improve virtualized I/O performance on multi-core systems," presented at the Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems, San Jose, California, 2008.

[26] S. Govindan, A. R. Nath, A. Das, B. Urgaonkar, and A. Sivasubramaniam, "Xen and co.: communication-aware CPU scheduling for consolidated xen-based hosting platforms," presented at the Proceedings of the 3rd international conference on Virtual execution environments, San Diego, California, USA, 2007.

[27] Y. Wang, X. Wang, M. Chen, and X. Zhu, "Power-Efficient Response Time Guarantees for Virtualized Enterprise Servers," presented at the Proceedings of the 2008 Real-Time Systems Symposium, 2008.

[28] C. Krintz, Y. Wen, and R. Wolski, "Application-level prediction of battery dissipation," presented at the Proceedings of the 2004 international symposium on Low power electronics and design, Newport Beach, California, USA, 2004.

[29] N. Thiagarajan, G. Aggarwal, A. Nicoara, D. Boneh, and J. P. Singh, "Who killed my battery?: analyzing mobile browser energy consumption," presented at the Proceedings of the 21st international conference on World Wide Web, Lyon, France, 2012.

[30] Robert B. Leighton, Richard Feynman, *The Feynman Lectures on Physics*. Vol I, 1963 Chapter 13, § 3, pp 13-2,3

[31] M. Dakshayini, H. S. Guruprasad. "An Optimal Model for Priority based Service Scheduling Policy for Cloud Computing Environment". *In International Journal of Computer Applications. '11*, 23-29,

[32] S. Xi, J. Wilson, C. Lu, and C. Gill, "RT-Xen: towards real-time hypervisor scheduling in xen," presented at the Proceedings of the ninth ACM international conference on Embedded software, Taipei, Taiwan, 2011.

[33] C. Xu, S. Gamage, P. N. Rao, A. Kangarlou, R. R. Kompella, and D. Xu, "vSlicer: latency-aware virtual machine scheduling via differentiated-frequency CPU slicing," presented at the Proceedings of the 21st international symposium on High-Performance Parallel and Distributed Computing, Delft, The Netherlands, 2012.

[34] M. Bourguiba, K. Haddadou, and G. Pujolle, "Evaluating and enhancing xen-based virtual routers to support real-time applications," presented at the Proceedings of the 7th IEEE conference on Consumer communications and networking conference, Las Vegas, Nevada, USA, 2010.

[35] Sarah Vallieres, "Understanding EdgeSight Network Data", White Paper, Citrix, 2010.

[36] James F. Kurose, Keith W. Ross, *Computer Networking*, 5th edition, 2009.