Journal of Theoretical and Applied Information Technology
31st October 2014. Vol. 68 No.3

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

SAIVMM: SELF ADAPTIVE INTELLIGENT VMM SCHEDULER FOR SERVER CONSOLIDATION IN CLOUD ENVIRONMENT

S.SURESH[1], S.SAKTHIVEL[2]

[1]Associate Professor, Department of Computer Science and Engineering, Adhiyamaan College of Engineering, Hosur-635109, Tamil Nadu, India
[2]Professor, Department of Computer Science and Engineering, Sona College of Technology, TPTC Main Road,Salem-636005, Tamilnadu, India
E-mail: [1]ssuresh.siv.72@gmail.com , [2]sakvel75@gmail.com

ABSTRACT

Cloud computing is an on-demand resource provisioning technology and server virtualization act as a driving force of cloud. Virtualization consolidates multiple physical machines into one machine, thereby cut cost and improves efficiency of data center. However, as all virtual machines (VM) share the same physical resources, contention for shared resources cause significant variance in observed system response time and throughput. Diverse and unpredictable workloads in cloud environments, needs resource allocations to be continuously optimized to ensure the hosted services meet their service level objectives (SLO). However, the current VMM algorithms are more oriented with providing fair access to the VMs; Lack the ability to adaptively determine the effects of changing resource allocations on the performance of the hosted IT services. Furthermore, as hardware getting evolved and multi core processor technology has increased density of processor cores in a computer at a faster rate, effective usage of the resources becomes a great challenge to software. This is a major bottleneck in cloud applications where performance plays a vital role for user acceptance. Taking this all into account, the paper propose a novel system using meta-heuristic combinatorial search techniques that automatically regulates the VMM CPU scheduler related to the applications on-the-fly with dynamic changes in the environment to maximize throughput and minimize response time. We used this resource allocation algorithm in an evaluation, consist of various scenarios with synthetic workloads. Simulation based results indicate that proposed model improves CPU utilization and make the best tradeoff between resource utilization and performance by 2% on average and up to 6% compared to the default VMM scheduler configurations. The proposed model discussed in this paper can readily be extended to a multi-tier cloud computing environment applications to reduce the overall performance delay.
Keywords: *Cloud computing, Virtualization, VMM Scheduling, Workload, Simulation*

1. INTRODUCTION

Clouds are a large poll of hardware or software resources that can be accessed on-demand like a utility computing. These cloud services can be provided without any knowledge of the physical location of the servers and the systems that provide the computing services. It is continuously gaining popularity, due to its ease-of-use, on-demand resource provisioning, pay per use business model, and ability to support execution of applications of diverse types. Virtualization act as a driving force of cloud by simplifying load balancing, dealing with hardware failures and easing system scaling through server consolidation. Server consolidation enables one to consolidate applications running on possibly tens of thousands of servers each significantly underutilized on the average; Thus by running multiple virtual machines on the same physical resources, virtualization promises a more efficient usage of the available hardware in cloud data centers. However, as all VMs share the same physical resources, contention for shared resources cause significant variance in the observed system response time and throughput. A recent survey [1] of datacenter applications show that some of the most common workloads targeted for virtualization are parallel computing applications, databases, web hosting, mail exchange and file hosting. These are multi-threaded that uses CPU, memory and Input/output (I/O) heavily. Thus, the success of next generation cloud computing infrastructures depends on how effectively these infrastructures, instantiate and dynamically maintain computing platforms, constructed out of cloud resources and services. This meet varying resource and service

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

requirements of cloud costumer applications are characterized by Quality of Service (QoS) requirements.

Everything is delivered to cloud users as services. In these regard, two important problems are frequently encountered with deploying IT applications in cloud. The first is overload or under load. A simple solution for this problem is to assign resources to VMs in static manner. However, static allocation becomes inefficient under varying load. The second problem is QoS. As cloud hosted services and applications are user oriented, QoS has a great impact on growth and acceptability of cloud computing paradigm. However, providing QoS requires a solid model that needs detailed insights of computing centers. Indeed, it is very difficult to dynamically allocate resources for multitier applications. i.e. How to effectively increase resource utilization and meet service level an objective (SLOs) in a shared virtualization environment is a great challenge. So it is a large challenge for the scheduler that can achieve the goal of good fairness, efficient workload balancing, and minimal wasted CPU time, when allocating the physical CPU time to VMs. Thus the scheduler with a good adaptiveness can make a better trade off among these factors, and can change its strategy for VMs with the different workload properties. Besides the reduction in infrastructure and ongoing operating costs, this work also has societal significance as it decreases carbon-dioxide footprints and energy consumption by modern IT infrastructures.

Virtualization solutions ranging from VMware, KVM and XEN can be implemented within a cloud; each has its strength and weakness. The, performance of a hosted application is sensitive to the hypervisor scheduler configuration parameters on which the application is running. However, the exact relationship between the value of the scheduler configuration parameters of the VM, and the application performance metrics such as response time or throughput is not obvious. Therefore, determining the appropriate parameter values that would provide certain SLA for an application is a hard problem due to dynamic nature of the workload; thus most of the time the parameters are left as default values. Subsequently, existing tools for performance and resource management of virtualized infrastructures lack the ability to dynamically determine the effects of changing resource allocations on the performance of hosted IT services.

Modern virtualization and middleware technologies create many possible ways for resource allocation and system reconfiguration by adding or removing virtual CPU cores to VMs or by changing the hypervisor scheduling parameters. Similarly, application servers provide means to create application server clusters and add or remove cluster nodes. There are advantages and drawbacks in the mentioned dynamic configuration options. Some of them can be used on-the-fly, but they require a special system setup or introduce high reconfiguration overhead. Thus, the proposed self-adaptive resource allocation algorithm, focus on adding or removing virtual CPUs to an application server cluster; that can achieve the goal of good fairness, efficient workload balancing, and minimal wasted CPU time.

As hardware support continues to be added and adopted to bring the performance of virtualized systems closer to that of native execution, the virtualization of these computing resources is not substantial. With respect to Moore's Law, Chip Multi-Threading (CMT) is becoming increasingly important because of multi-core CPUs that combine multiple processor cores inside a single physical CPU. Thus, the increasing number of processing cores has become a promising way of improving the performance of servers by quickly adapt to cores changing capabilities, resulting in numerous performance and other benefits compared to existing techniques. Thus, it can be used to improve the scheduling of consolidated servers by allowing the cores of a chip to be dynamically partitioned among guest virtual machines. Compared to conventional scheduling/multiprocessor scheduling, dynamic partitioning provides higher throughput, lower transaction latency, and more isolation. Yet it can quickly adapt to bursts in demand and it also has changing capabilities of the underlying hardware.

Virtualization delivers dramatic resource usage and cost cutting, but overall data center efficiency metrics may still not be what they should, with respect to the state-of-art hardware and software advancements. Thus, this research aims at presenting the evaluation done on, how does application response time in cloud get impacted with adaptive CPU resource allocation with changing customer workloads? When should a reconfiguration be triggered for achieving effective resource usage without compromising SLAs? How rapidly and at what level be the reconfiguration triggered?

Evaluating new scheduling techniques under various, controllable and repeatable conditions are impossible in real Cloud. Because, system level scheduling implementation requires deep

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

understanding on hardware architecture, as well as low level programming and debugging. Consequently, it makes qualitative evaluation of scheduling algorithm difficult and requires significant effort. In addition, the proposed algorithms are sometimes affected by the existing system architecture [3]. In order to address this problem, the cloud virtualization environment, is simulated to evaluate the performance of the defined algorithm using CSIM simulation toolkit, and C/C++ library that allows assembling a complete virtualization system with flexible configurations. Although the framework's components are constructed using the CSIM model, the users of the framework are not required to know CSIM or its underlying concepts. All they need is to use CSIM library, type in parameters say workload distribution, and write a C/C++ function to express the scheduling function.

In summary, the contribution of this paper is multifold and our works are as follows: i) Study on server virtualization and its various solutions. ii) Study on state of the art work in VMM scheduling. iii) Conducting two performance case studies that focus on how the performance is affected by the amount of CPU allocation. iv) Construct an adaptive system that automatically adjust the VMM CPU scheduler based on the workload fluctuations to give guarantee QoS using combinatorial heuristic search techniques. v) The best CPU scheduler configuration for each synthetic workload with a given target amount of users was determined. vi) Investigate and evaluate more scheduler configurations. vii) The performance of various scheduling algorithms in various configurations was measured.

The rest of the paper is organized as follows. Section 2 provides some background information and basic concepts related to server virtualization, chip multithreading, workload forecasting, and meta-heuristic search techniques. Section three deals with review of literature. Section four presents resource influence on VM and state-of-the-art VMM schedulers. Section five illustrates the soundness of proposed mechanisms to be built into the systems to enable self-management, for the case of cloud. Section 6 and 7 presents experimental methodology, setup, and the discussion of results. Section 8 presents the conclusion and suggestions for future research.

2. BACKGROUND

This section presents some background information and definitions on server virtualization and its solutions, chip multithreading, combinatorial search techniques and control decisions regarding workload forecasting that helps to understand the conceptual decisions made.

2.1 Server Virtualization

Server virtualization is an abstraction of underlying physical hardware. It creates virtual environments that allow running multiple applications in separate virtual containers hosted on single hardware that enables workload isolation and makes consolidation possible. VMM, a software layer, VMM separates underlying hardware from the software running on top of it, creates a notion of the hardware for a virtual machine. Thus, it creates and manages processor, memory, and I/O units for the virtual OS. The first and foremost application of virtualization is server consolidation. It is a process encapsulating the single server workloads into VMs and running them in a shared hardware platform via hypervisor. Server consolidation provides an effective solution to parallelism and high utilization of modern multi-core processors by running multiple virtual machines on top of the single platform with virtualization. Virtualization Technology can be classified into three major approaches[2,4], based on the performance, ease of installation and administration, level of security between each virtualization images, and supported hardware platforms and methods of executing guest OS code with / without hardware access.

2.1.1 Full virtualization

It is a type of virtualization in which hardware is completely emulated via hardware and software that allows operating systems, and its kernel to run unmodified in a VM. It executes privileged guest code in software, typically with just in time translation for speed. Binary translation virtualization technique solved this problem by having the VMM, examine the binary instructions before they were executed, and dynamically rewrite sections of code that would try to execute a privileged instruction so that the VMM would maintain control of the system and emulate the effect of the instruction. Here VMM runs itself as an application inside the host OS, subsequently, all resource allocations and scheduling facilities are offered by the host OS. The VMM presents an interface to VM that is indistinguishable from physical hardware. Hardware virtualization support in recent x86 CPUs support a privilege level beyond supervisor mode, used by the hypervisor to control guest OS execution. As every resource request from the guest machine needs to go through

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

the VMM and hence, there is a lot of overhead on the VMM. For example, if there is a request for disk read/write operation, the request is routed via VMM, which then validates the request and transfers the requested data back to the guest OS. This results an additional delay along with the normal response latency and transfer time. However, data prefetching and efficient scheduling can help to improve the delay in such a case.

2.1.2 Para virtualization

This approach defines new hardware software mixture architecture known as hypervisor, which is similar to physical hardware emulated CPU, memory and I/O instructions, that are replaced by hypercalls. It reduces overhead inherent in dynamic translation. By running directly on the machine, the hypervisor has direct communication with the hardware. It does not virtualize a total computer environment, in the manner that previous approach does. Thus, it allows the guest OS to execute directly in user mode, by providing a software interface (API), which is similar to the underlying hardware allows the guest to use to perform privileged functions. In that, the guests are aware that they are virtualized and need modifications in the guest device drivers. Here, the hypervisor perform scheduling and resource allocation for all VMs. Thus, it stimulates VMM directly, controls scheduling of VCPUs on PCPUs rather than having VCPUs execute as threads in a host OS that are scheduled based on the host OS CPU scheduling algorithm and makes work simpler. In addition, if the virtualization layer supports direct communication with the hardware through available facilities like hardware assisted virtualization (section 2.1.3), the calls can be mapped directly to the hardware. Unlike the previous approach, this technique has a lower communication overhead as the hypervisor does not completely intermediate the guest and the physical hardware. Also, the performance of the paravirtualized setup depending on the workload of the guests, leads to instability in performance swinging between the two ends of spectrum.

2.1.3 Hardware assisted virtualization:

It is a hardware extension which includes a new CPU execution mode, guest mode, in addition to the original mode provided by the vendors like Intel and AMD. CPU execution in host mode remains with full ring privileges (from ring 0 to 3); however the rings in guest mode are deprivileged for hypervisor trap and emulation. Thus, hardware assisting processors give the guest OS the necessary authority to have direct access to platform resources without sharing control over the hardware. Formerly, the VMM should emulate the hardware to the guest OS while it retains control of the physical platform. These new processors give both the VMM and the guest OS the authority each needs to run without hardware emulation or OS modification. Thus, privileged instructions are trapped in the hardware and redirected to hypervisor, eliminating the need of ring deprivileging used in the former solution; Simplifying the hypervisor implementation, and therefore supports unmodified guest OS (Windows). Furthermore, hardware assisted shadow page table for virtualizing memory supports unmodified guest or hardware virtual machine, to manage the translation from guest linear memory address to host physical memory address. Subsequently, key state information for CPU and guest OS can be stored in the protected memory which can be accessed only by the VMM, protecting the integrity of the handoff process. All trapped, emulated I/O devices and paravirtualized I/O devices are enabled in hardware virtualization. This provides virtual machines greater capabilities, and scalability. As guest shares the page table entries among all of its VCPUs, the hypervisor needs to track the page table access for each VCPU. The corresponding SPT entries are also shared among these VCPUs.

2.2 Scheduling in a Hypervisor

Aforesaid, three virtualization approaches infer, VMM is an important entity as it routes any request by the guest to access the hardware. Thus, it is very important to VMM to ensure that all the guests are given reasonable access to the hardware to ensure SLO to the guests. Such a SLO mechanism is provided by scheduling the guests appropriately. A scheduling mechanism guarantees that a guest performs its transactions in its scheduled time and allows all the guests to get a reasonable chance to access the resources. This kind of a scheduling provides equality in access and it trades off the latency of the access. The VMM gives a portion of the full physical machine resources to each guest domain, thus multiple guests must share the available resources. Therefore, the VMM does not generally picture the full power of the underlying machine to any single guest. Instead, it allocates a portion of the resources to each guest domain. It either attempts to partition resources evenly or favors some guests over others in a biased fashion. As the workload and demand varies, it grants each guest OS a limited amount of

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

specified resource, like memory and allows each guest OS only its fair share of other resource like CPU. Similarly, it does not like all guests to have access to every physical device in the system and thus it exposes only the devices it wants each guest to see. Sometimes, it even creates virtual devices that have no corresponding underlying physical device like virtual network interface. Thus, from the perspective of VMM's CPU scheduler, each VM is represented by one or more virtual resources, and the primary goal is to maintain proportional share of CPU time allocation of each guest domain, according to user defined SLA based on unpredictable workloads. Thus, a core requirement for an effective virtual machine scheduling algorithm is the ability to dynamically provision its various resources to its various users according to their instantaneous needs and in fulfillment with negotiated SLAs to afford an environment like cloud.

2.3 Chip Multithreading (CMT) and Virtual CPU (VCPU)

CMT is to combine the resources of multiple CPUs in a single host system, in which all the processors behave identically. CMT lets any processes in the system can execute on any processor. By providing additional CPU resources to execute multiple threads simultaneously, it works faster and more efficient than a single core processor; it shares the processing load, improving overall system performance. The system also can shut down portions of the cores that aren't in use, saving power and generating less heat. In addition, as each core on the chip has its own memory controller, significantly improves memory performance. Also, connecting the processor cores together lets data flow freely and reduces latency problems. Thus increasing number of processing cores (Moore's law) becomes a promising way of improving the performance of servers.

Logical Domains technology provides flexible assignment of hardware resources to domains, with options for specifying physical resources for a corresponding virtual resource. One of the distinguishing features of logical domains compared to other hypervisors is the assignment of CPUs to individual domains. Each domain is assigned an exclusive use of a number of CPUs, also called threads. Within a domain, these are called virtual CPUs (VCPUs) and the granularity of assignment is a single VCPU.A domain can have from one VCPU up to all the VCPUs on the server. The number of CPUs in a domain can be dynamically and none disruptively changed on the fly and the change in the number of VCPUs in a running domain takes effect instantly. The number of CPUs can be managed routinely with the logical domains, by dynamic resource manager, provided by the respective tool. This method avoids the frequent context switches that usual VMM must implement to run several guests on a CPU and to intercept privileged operations, has an impressive benefit in terms of simplicity and reduction of the overhead. Because each domain has dedicated hardware circuitry, a domain can change its state without causing a trap and emulation. Typically, a virtual machine can be assigned multiple VCPUs and an application with multiple processes or threads can have considerable performance improvements when multiple VCPUs are executed on diverse physical CPUs. To improve the CPU usage effectiveness in the CMT configuration further, the CPU scheduler must implement a global load balancing functionality that quickly reassigns VCPUs among available physical CPUs.

2.4 Dynamic Resource Allocation

Dynamic resource allocations are quite essential for adaptive computing. Generally, there are two approaches namely proactive allocation and reactive allocation. Reactive allocations are used to adjust resources based on demand and recent behavior. It is preferred more to handle momentary fluctuations smoothly. Whereas, proactive resource allocations involve taking up actions to make resources available for upcoming load spikes. It is good for managing resources in a multi-tenant cloud where it is common to have hot spots at some locations, while still having spare resources scattered throughout the datacenter. Inability to quickly use fragmented spare resources in a datacenter during load spikes causes host level over provisioning. When predictions are accurate, this scheme provides very good performance. Forecasting can be achieved by applying many techniques [5]. However, we describe here two very popular forecasting techniques namely weighted moving averages, and exponential smoothing. *Moving Average:* A simple and widely used forecasting method is the simple moving average (SMA), which makes the value to be forecasted for the next period from the average number of previous observations. With this technique only one value can be forecasted at a time. This limitation can be overcome by the weighed moving averages where the value to be forecasted maintains an almost constant value for quite a while before changing significantly. The forecasted value is computed as a weighed average of a given number

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

of the most recent observations. The weights are chosen in a way that reacts the relative importance of the newest/oldest observations. *Exponential Smoothing:* Exponential smoothing technique is an additional popular forecasting technique known for its peculiarity for its peculiarity to make predictions from time series data that exhibit upwards and/or downwards trends. It computes Forcasted Value = ∂ x Previous Actual Value + (1-∂) x Previous Forcasted Value. Where, ∂ is used to gauge the relative importance that is associated with the previous prediction as opposed to the earlier observation (i.e., more weight is given to observed values rather than to the predicted ones if ∂ is close to 1 and vice versa).

2.5 Genetic Algorithms (GA)

As scheduling problems belong to combinatorial optimization problems, one tends to use optimization algorithms to find optimal solutions, which have been studied from Simulated Annealing, Genetic Algorithm, Hill Climbing and Particle Swam. Compared to standard heuristics, genetic algorithms [6, 7] are well suited for fine tuning structures which are very close to optimal solutions. GA is a computational model that emulates biological evolutionary theories to solve optimization problems. In computing terms, GA maps a problem to a set of binary strings, each of them representing a possible solution. The GA then manipulates the most promising strings searching for improved solutions, through a simple cycle of four stages: i) creation of a "population" (randomness) of strings, ii) evaluation of each string (*reproduction*), iii) selection of "best" strings using *fitness function*, and iv) genetic manipulation (cross over) to create the new population of strings. Here, the decision variables are the CPU usage limits to be enforced on the co-located VMs. In this work, a genetic algorithm based technique is applied. It searches the space of various possible CPU usage limits and finds a near to optimal solution. It uses the negative of utility optimization objective given in section 6 as the fitness function since the GA is designed to minimize the fitness function. As a result, it maximizes the system utility. The genetic algorithm generates a new population of candidate solutions and evaluates their fitness values in various iterations. Through the research, it is observed that genetic algorithm is able to converge within 50 iterations or generations.

3. RELATED WORK

VMM resource allocation, scheduling, and analysis of virtualization performance are some of the most important problems in server consolidated virtualization research in enterprise applications. In specific, optimizing the performance of the resource virtualization is an ongoing research area, and there are several new techniques are proposed to implement resource virtualization. Hence, the related work is divided into the following groups: (i) the performance behavior of the applications running inside the virtual machines (ii) approaches for optimizing the resource provisioning to improve the performance of virtualization in the real systems and (iii) approaches for dynamic resource provisioning optimizing the performance of virtualization using simulation.

Cherkasova et al. [8] have compared three schedulers in XEN, SEDF (Simple Earlier Deadline First), BVT (Borrowed Virtual Time), and Credit. They studied that the credit scheduler used in XEN are performance-oriented and do not accordance with configured values for some workloads and choosing the right parameters for individual virtual machines is crucial in order to get the desired application performance. Similarly a study is made to this by Liu et al.[9] to develop a mechanism that adaptively regulate these parameters, In which they designed a feedback controller that takes care of regulating the scheduling parameters of VM such that each application gets its relative level of performance. Govindan et al. [10] proposed an algorithm to schedule the CPU of VMM that considers the largest number of network packages to minimize the delay of the packages caused by virtualization, by creating an unfair advantage to communication intensive applications over CPU intensive ones. It enhances the performance even under the high consolidated virtualized applications, while still adhering to the high-level resource provisioning goals in a reasonable manner. Similarly, Ongaro et al [11] examined the impact of the CPU scheduling on network I/O performance in XEN and inferred that XEN privileged allocating CPU resources to CPU-bound VMs rather than I/O-bound VMs, and which could influence network bandwidth and latency in unwanted ways. They proposed few enhancements to boost the I/O performance of VMs, which sort the domains in the run queue based on their remaining credits and to place all the I/O intensive domains in the same domain that reduce the pre-emption of the event channel notifier, improves the overall latency and performance. In [12], author propose a coarse-grained feedback control framework that works based on transfer functions to model the dynamic relationship between a performance metrics and physical control features, for better performance

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

prediction. It dynamically allocates resources to applications running in a virtualized environment. Subsequently, for reconfiguration options focus on the hypervisor's cap and disk share parameters. However, the valuation with RUBiS and TPC-W in combination with a production trace driven workload is hopeful. In [13], authors developed a mathematical scheduler modeling to analyze and empirically compare XEN's scheduling algorithms such as co proportional, proportional share scheduling strategies that provide a convenient infrastructure to quickly examine new idea based algorithms. Chieu et al. [14] proposed a reactive algorithm for dynamic VM provisioning of Platform as Service and Software as Service applications, whereas this current approach considers adaptive reconfiguration of available virtual instances, say increase or decrease their resource capacity based on their request arrival rate and performance metrics. Watson et al. [15] proposed a probabilistic performance model using quantile regression making it possible to predict the response time of the web authentication benchmark depending on the allocated resources at the virtual machine level. However, the performance relevant factors are rarely explicitly provided. In [16], author developed a multi-level resource allocation framework Mistral that adapt a VM's CPU capacity, by add or remove a VM, live-migrate a VM between hosts and shutdown or restart a host. This method considers performance, transient costs and power consumption in its reconfiguration algorithm. However, it is based on a simple multi-tier application with read-only transactions and a fixed web tier modeled with a layered queuing network. Lim et al. [17] proposed a mathematical model to characterize workload using multiple resource usages. They characterize a host as a collection of *m* resource queues. They also characterize each application as a vector of size *m*, where i^{th} element is calculated as the amount of time using i^{th} resource divided by its runtime when running in standalone mode. However, this model is not practical. As per this model, running multiple applications together not take longer than their sequential execution. This is not true in virtualized environments. Severe contention between two VMs may lead to slowdown of more than twice. The fundamental problem is how to obtain resource vectors. Generally, resource usage is represented as utilization or throughput. Measuring the amount of time using I/O and network is unusual and not easy. In [18], author proposed a novel system PAC, which estimates VM's performance on each host by measuring difference between time series of VM and host by periodically characterizing each virtual machine as a repetitive time series of resource consumption, and each host as a repetitive time series of remaining resources. Consequently, it then schedules the VM to a host with minimum difference. For VM schedulers, the way to characterize workload determines how to estimate VM's performance on each target, in the sense either physical core or host. Thus, scheduling decision is just a process of selecting the most suitable objective based on evaluation. In [19], author proposed the problem of optimal load distribution of generic tasks on multiple heterogeneous servers preloaded with special tasks in a cloud computing environment. He formulated it as a problem of multi-variable optimization based on a queuing model. He developed algorithms to find the numerical solution of an optimal load distribution and the minimum average response time of generic tasks. In [20], discussed the overall view on several components while focusing on the overhead of virtualization. They conducted diverse experiments on two unlike virtualization platforms and on a native machine. They also calculated the virtualization overhead of some components like network, memory and disks. Later, they proposed a model which helps to estimate the performance of virtualized machines based on linear regression. In [21], author used Artificial Neural Network, a machine learning technique to predict the performance impact of real-time scheduling parameters, VM deployment and workload type on the system performance based on measurements using various MATLAB benchmark tests. In [22], author used analytical queuing models to quantify the slowdown of virtualized applications in server virtualization scenarios. However, it is shown that using the total CPU time as distribution factor to derive workload specific virtualization overhead, typically results in an uneven estimation at best. In [23], author presents a new approach for self-adaptive resource provisioning in virtualized environments based on online architecture level performance models. He investigates the use of such models as a means for online performance prediction allowing to predict the effects of changes in user workloads as well as to predict the effects of particular reconfiguration actions, undertaken to obey SLA efficient resource usage. By using virtualization techniques, they applied these allocation changes to evaluate the use of such models for online performance prediction. However, in the present research, the influences of virtualization on system performance to integrate the gained insights into the proposed performance

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

models are investigated. Guenter et al. [24], developed energy aware on demand virtual machine consolidation system focused on web, where SLAs are defined in terms of the response time. They applied weighted linear regression to forecast the future workload and proactively optimize the resource allocation. This approach is in line with the Local Regression (LR) algorithm. In [25], the authors present a new fuzzy controller for load balancing in cloud computing, which requires two input data like processor speed and assigned load of VM and provides the balance load to reduce the application response time. This method can be used only for CPU intense applications where the SLA is related to CPU speed. On the contrary, the technique applied in this present research is more general as it does not make any guess regarding the CPU speed of the machines existing. Kraft et al. [26], proposed two approaches based on queuing theory to predict the I/O performance of consolidated virtual machines. First, the trace based approach which simulates the consolidation of homogeneous workloads that is modeled as a single queue with multiple servers having service times fitted to a Markovian Arrival Process. Second, they foresaw storage performance in mixed workload consolidation scenarios. They generated linear estimators based on mean value analysis. In addition, they also created a closed queuing network model, with service times fitted to a Markovian Arrival Process. Both methods use monitored measurements on the block layer that is lower than typical applications run. Besides, they pay attention on performance prediction without considering the performance effects due to changes in the workload amount. In [27], the authors proposed a fuzzy controller for allocating virtualized resources with respect to the application response time. Both works consider only the response time and its deviation from the SLO value as input parameters to the controller. Instead, this present paper combines the information regarding the response time with the VCPU utilization. The combination of these two parameters allows the adaptive genetic controller to gain more knowledge on the system load, thus results in a more accurate CPU capacity allocation. In [28], author proposed solution for the synchronization problem of a server consolidation by modifying the XEN Credit Scheduler, in which new priority TURBO added to the scheduler to avoid the scheduling decisions that, was made for synchronization. TURBO allows need to synchronization VCPUs to preempt and being picked up to run at the next time slice without impacting overall system fairness; thus the threads in the concurrent program can be synchronized. Consequently, proposed scheduler works fine and greatly enhances the performance in concurrent workload by decreasing CPU allocation errors; but it incurs minor performance drop in a parallel workload due to the extra overhead of finding the most urgent work from other PCPUs.

Thus, it is argued that earlier works have missed a good opportunity of cost and performance optimization by disregarding workload aware resource allocation or scheduling in multi-core systems. In addition, to the best of our knowledge, no works exploit the genetic algorithm extensively. Thus our approach taken by genetic algorithm usually falls into two different categories, whereby (1) the genetic algorithm is used to model the behavior of a system, or (2) the genetic algorithm is used to design a controller to act on the system at run time in order to guarantee a specific QoS.

4. RESOURCE INFLUENCE ON APPLICATION PERFORMANCE IN VM AND THE STATE OF THE ART VMM SCHEDULERS

As performance provision is the major concern of VMM scheduler in cloud, this section provides 1) two quantitative case studies that focus on how the VM performance is affected by adapting a VM's CPU capacity and 2) two qualitative case studies of widely popular virtualization VMM schedulers.

4.1 Performance Influencing Factors

As virtualization introduces dynamics and increases flexibility, a variety of additional factors can influence the performance of virtualized systems. In [20, 29] having analyzed major representative virtualization platforms, abstracted a generic performance model of VM performance influencing factors as shown in figure 1. Those are virtualization type, hypervisor's architecture, resource management configuration and workload profile. Though, several influencing factors are grouped under the resource management configuration, the CPU scheduling configuration has a significant influence on the virtualization platform's performance and chief among them are virtual CPUs allocated to a VM, the number of VMs and resource over commitment. Managing virtual CPU requires an additional management layer in the hypervisor and the number of VMs has

Journal of Theoretical and Applied Information Technology
31st October 2014. Vol. 68 No.3

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

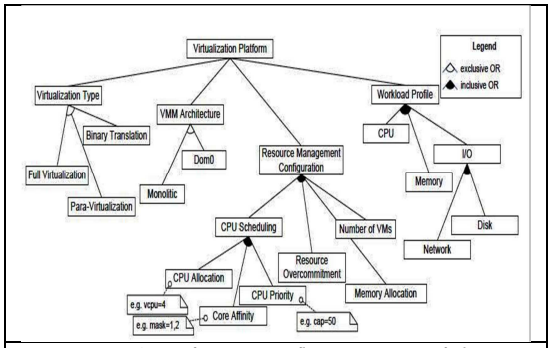

Figure 1: Performance Influence Factors of the Virtualization Layer (Adapted from [20])

a direct effect on how the available resources are shared among all VMs.

4.2 PERFORMANCE STUDY

4.2.1 Impact of CPU allocation

As it was discussed earlier, CPU cores are one of the main sources of performance interference as shown in Figure 1. Even with such physical isolation for the CPU, the typical relationship between application performance and the CPU allocation is difficult. Thus, this complexity is demonstrated by executing two types of experiments, targeted at the component and application level in virtual machine environment by setting CPU limit at different levels. The performance of the CPU intensive applications (kernel compilation) and virtualized applications (OLTB) are measured while varying the VM's CPU limit from 1 core to 8 cores. All resource allocations were kept high enough to ensure that those are all not the bottleneck. All the experiments were conducted on physical hardware configured with AMD FX 8-Core Black Edition FX-9590. It has 8 *4.7 GHz AMD Opteron 8 core processors with 3MB L2, 6MB L3 cache each, 8 GB DDR2-667 main memory, 100 GB of storage and 10/100/1000-BaseTEthernet connections. Both host and virtual machine are configured with 8 VCPUs and 4 GB RAM, 50GB HDD with Ubuntu 14.04 LTS (Trusty Tahr). The virtualization solutions considered for the experiment is XEN 5.0. In all solutions, hardware virtualization support is used to virtualize 64-bit guests over a 64-bit host. For the XEN machines, virtual NICs use the default bridged network driver. Two types of benchmarks [30, 31] namely Linux kernel compile, MySQL-SysBench are used and are targeted at the component and application level of influencing factors.a) *Linux kernel compile:* The kernel build benchmark unarchieved the Linux kernel source archieve, and build a particular configuration. It heavily used the disk and CPU. It executed many processes, exercising fork(), exec(), the normal page fault handling code, and thus stressing the memory subsystem. It accessed many files and used pipes, thus stressing the system call interface. b) *MySQL SysBench:* It is a modular, cross platform and multi-threaded benchmark tool for evaluating OS parameters that are important for a system to run a MySQL database under intensive load to evaluate its performance. SysBench, which was run on a separate client machine, was configured to send multiple simultaneous queries to the MySQL database with zero think time.A simple database that fit entirely in memory is used. As a result, these workloads both saturated the virtual CPU and generated network activity, with very little disk I/O. For various numbers of threads the experiment is conducted and the results of both are given in the Figure 2 and Figure 3.It shows the normalized performance of these examinations. As seen from the graph, both the benchmarks workloads behave linearly, and the performance slope is different at various CPU allocation ranges. While the kernel compilation saturate quickly at 3 VCPUs, SysBench performance, on the other hand, varies almost linearly with CPU allocation. But at some time the saturated point is visible because of resource over provisioning. Thus, this data reveals the fact that virtualized workloads can have quite different performance curves with respect to number of CPU allocation. The above analysis of hypervisor's behaviors demonstrates that resource pools are one of the vital factors in the constitution of virtualization overhead and current scheduling scheme in conventional VMM the shows bottlenecks on the massive advanced system with heavier load i.e., more VMs and heavier stress as shown in the experiment. Hence, in order to maximum the hardware resource utilization, VM management has become an important research field of virtualization technology. Thus, VM scheduling is crucial for the throughput of a system and affects the overall system performance.

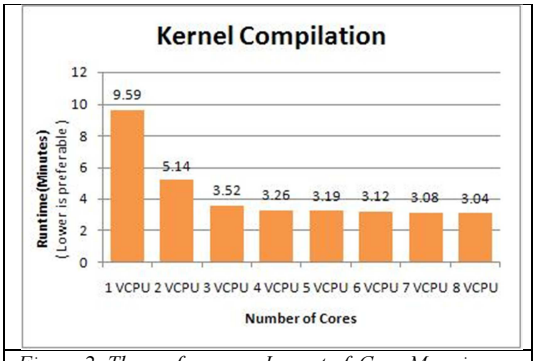

Figure 2: The performance Impact of Core Mappings

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

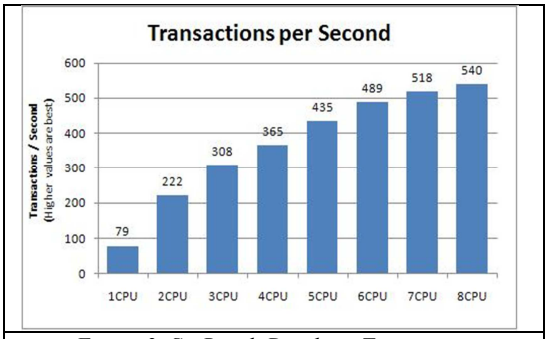

Figure 3 :SysBench Database Transaction

This leads to the conclusions that resources allocated to a virtual machine directly have an impact on the hosted application's performance and choosing appropriate control knobs to handle resource allocation for a VM is critical to ensure desirable performance and create a robust model.

4.3 State of the art VMM Schedulers

VCPU scheduling remains as a challenge for Virtualization technologies, especially with hypervisors starting to host Chip Multithreading VMs. A naive, yet popular, implementation is to use a simple Round-Robin algorithm when assigning processor resources to each VCPU. This option is available in most hypervisors. e.g. in KVM or Virtual Box hypervisors. However, this approach can cause additional synchronization latency for guest VMs due to VCPU preemption. Whereas, VMware ESX and XEN are two of the leading virtualization systems for the x86 architecture, and they both allow for CMT virtualization. However, implementing CMT virtualization is difficult because the two technologies have different goals, and virtualization in particular can conflict with the expected behavior of a CMT system. As the implemented prototype in this paper is a generic, it discusses briefing the main features of these two VMM scheduler's algorithms in specific.

4.3.1 CPU scheduling algorithms in XEN

XEN is quite unique among VM platforms because it allows user to choose among different CPU schedulers. It implements a higher level abstraction scheduling operations, where each scheduler needs to implement its own scheduling policy and registers itself to this interface. XEN supports three different types of schedulers [32, 8] namely Borrowed Virtual Time (BVT), Simple Earliest Deadline First (SEDF) and Credit Scheduler .The users can set the scheduler option during XEN's boot time by passing the parameter value of sched *i) Borrowed Virtual Time (BVT):* It is a proportional share scheduler that is suited for I/O intensive domains. The scheduler adjusts itself dynamically with the varying I/O intensities when specified with the correct parameters. It is based on the concept of virtual time, dispatching the runnable VM with the smallest virtual time the low latency support is provided in BVT for real time and interactive applications by allowing latency sensitive client to warp back in virtual time to gain scheduling priority. And the client can effectively borrow virtual time from its future CPU allocation. Each runnable domain Dom_i will receive CPU proportion according to its weight w_i, and the virtual time vt_i of Dom_i is incremented by its running time rt_{ij} in the j^{th} scheduling around, divided by w_i: $vt_i \leftarrow vt_i + rtr_j / w_i$. However, due to the lack of Non Work Conserving (NWC) mode (unused CPU cycles of one domain can't be used by the other domain), its usage is severely limited in many application environments. *ii) Simple Earliest Deadline First (SEDF):* In this algorithm, the domains request a minimum time slice that requires for communication. The request is a tuble of (s_i, p_i, x_i), which means Dom_i will receive s_i units of time in each period of length p_i. The x_i is a boolean flag indicating whether Dom_i is scheduled in WC-mode or NWC-mode. SEDF performs well when the workload is low, but when running in heavy workload, many clients are observed to miss their deadlines and the scheduling overhead significantly increases, where the domain requests for't' slices every 'p' periods of CPU time. One main shortage is the lack of global workload balancing on multiprocessors, and the CPU fairness depends on the value of the period. Besides, the lack of global load balancing on multiprocessors, implementation also limits its usage. *iii) Credit Scheduling:* BVT lacks NWC-mode while SEDF is found to be unstable under heavy workload and does not support CMT well, so both of them were replaced by Credit scheduler in XEN. The credit based scheduler is recently incorporated into XEN and it provides better load balancing and low latency mechanisms. This algorithm is a kind of proportional share (PS) strategy, featuring automatic workload balancing of virtual CPUs across physical CPUs on a CMT host. According to the scheduling algorithm of Credit Scheduler using in XEN hypervisor, each virtual CPU is asynchronously assigned to a physical CPU by CPU scheduler in order to maximize the throughput. Specifically, when there is no runnable VCPU on the current physical CPU, the scheduler will try to migrate one runnable VCPU from the

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

other physical CPUs. Each domain is assigned with a (weight, cap) pair. Similarly, the scheduler allocates CPU time proportion (in credit) to each domain according to its weight. All queued VCPUs are sorted by their remaining credit, and the scheduler will select the VCPU that has most credit to run. When the cap is 0, VM receives extra physical CPU (WC-mode), while a nonzero cap (expressed as a percentage) limits the amount of physical CPU time obtained by a VM (NWC-mode). The algorithm uses followers interval for the physical CPU allocation. The priorities (credits) all runnable VMs which are recalculated in the interval, which is mainly in proportion to weight that VMs are assigned by the user. This algorithm can efficiently achieve a global workload balancing on a CMT system when the majority of the workload is not the high concurrent application. However, all these choice come with the burden of choosing the right scheduler and configuring it.

4.3.2 VMware ESX server VCPU scheduling algorithms

The default approach by KVM or Virtual Box hypervisors (Round-Robin algorithm) cause additional synchronization latency for guest virtual machines due to VCPU preemption. In order to eliminate this synchronization latency, VMware applies a co-scheduling algorithm [O. Sukwong, and H. Kim], which uses a concept similar to gang scheduling [33]. Co-scheduling requires that all VCPUs are associated with a VM to be scheduled simultaneously in order for the VM to run. Such an algorithm helps to avoid the synchronization latency, as both the waiting VCPUs and the lock holding VCPU are preempted and resumed at the same time. This "strict" co-scheduling approach, however, introduces a fragmentation problem. A VCPU can only be scheduled after the hypervisor gathers enough resources to execute all other VCPUs in the same VM. However, ESX has several optimizations to improve performance over a naive implementation of co scheduling, which would require even idle VCPUs in a VM to execute. First, ESX is able to detect if a VCPU is executing an idle loop, and in this case ESX does not schedule an idle VCPU to run nor require it to be co-scheduled for active VCPUs to run. Second, ESX uses a technique called relaxed co-scheduling that helps prevent requiring physical CPUs from being idle in order to start running VCPUs in an CMT system. ESX provides three control knobs for CPU allocation to individual VMs: *reservation, limit,* and *shares*. Reservation guarantees a certain minimum CPU allocation expressed in MHz Limit (in MHz) provides an upper bound on the CPU allocation. Share provides a mechanism for proportional allocation during time periods when the sum of the CPU demands of the currently running VMs exceeds the capacity of the physical host.

Having analyzed two major representative virtualization platforms, one can infer that current commercial resource management tools provide only partial solutions to VCPU scheduling problem i.e. it provides resource management capabilities by forcing virtual machines allocation to be within certain limits. In addition, these tools do not address setting these limits with appropriate values for each application, or how they should be changed in case. Thus, a resourceful VMM scheduler is important for increased throughput and decreased response time. Given varying workloads, there is a particular scheduling algorithm that is more efficient at scheduling VM for particular types of workloads. Thus, it is possible to fine tune the VMM scheduler to maximize throughput and minimize response time with specific type of workloads subject to SLA.

5. SYSTEM MODEL

This section presents the proposed self-adaptive resource management model and its working logic. Generally VMM schedulers repeat three steps: workload characterization, performance estimation, and scheduling decision. Thus, the core concept revolves around the idea of building mechanisms into systems that allow for dynamic reconfiguration of VM's VCPU, based on the variations of the workload to achieve a) an improved overall system performance to withstand SLA and b) a better utilization of system resources. To achieve these goals, a computer system needs to be checked regularly. Thus this section shows how the mentioned goals are attained through a combined use of system models that guide heuristic combinatorial search techniques in their exploration of the space of possible configurations. The system model evaluates, predicts, the performance of a system for a given configuration point.

The proposed system algorithm works as follows. Thereby, all the virtual machines are serving the incoming requests; VMM monitors the resource utilization of the various resources, and performance of the system. VMM executes a SAVIMM algorithm, at regular intervals, called Monitoring interval (MI), to determine the best configuration (suitable number of cores) for each VM with the help of a meta-heuristic algorithm. As

Journal of Theoretical and Applied Information Technology
31st October 2014. Vol. 68 No.3

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

a result of running the controller algorithm, reconfiguration commands are generated to instruct the system to change its configuration. To maximize the performance, design issues such as i) fairness in resource sharing among VMs, ii) workload balancing among virtual CPUs in a CMT VM iii) unneglectable cost of the wasted CPU time during the period of synchronization between VCPUs in a VM and iv) adaptiveness for VMs with the different workload properties, should be considered for the scheduler in the VMM. Hence, this model runs continuously to ensure that provisioning goals are met at all times and set the following design goals for resource provisioning approach: *Automation*: All decisions related to provisioning should be made automatically without human intervention; *Adaptation*: The application provisioner should adapt to uncertainties such as changes in workload intensity; *Performance Assurance*: The resource allocation in the system is dynamically varied for ensuring achievement of SLA targets.

5.1 The General Control Approach

This section presents the control architecture of the proposed adaptive systems. It describes the system architecture and components, and it also describes how interact with one another. Furthermore, some control decisions regarding workload forecasting, and frequency of control are also discussed. The architecture of the SAIVMM system model is best illustrated in Figure 4.It has five main components namely workload intensity supervisor, workload forecaster, SLA observer, system performance examiner, genetic algorithm guided VCPU regulator. The dynamic balancing component, VCPU regulator of SAVIMM reconfigures the individual VM's VCPU demand based on the resource requirement.

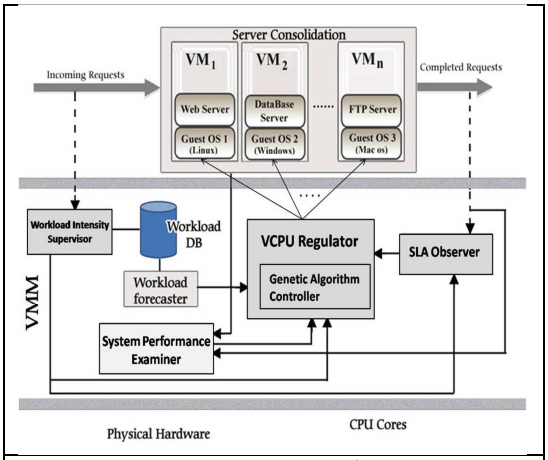

Figure 4: SAIVMM Framework Components

This component performs a re-evaluation of the resource pools, in a regular interval, based on the performance evaluation subject to SLA over a period of time. If there is a big imbalance in the resource pool, then the balancing component will be more aggressive in the balancing process. ***SLA observer*** is a component that computes the measurement required for implementing the control system. It calculates average response time, throughput and resource utilization for each client class on specified time periods. It has a list of completed requests for each client class, which is populated by the resource unit class after servicing the requests. The designer specifies the time interval to calculate the statistics. The generated statistic report is used by the external entities for analysis and makes runtime decisions. Afterwards, the list of request is cleared to accumulate the completed requests till the next sample instance. The SLA observer module uses the average arrival rate of requests obtained in the previous Monitoring Interval (MI), as an estimation of the expected workload intensity for the next MI. This value is then used by the algorithm to compute the SLA value for a given set of configuration parameters. The drawback of this approach is that it overlooks any increasing or decreasing tendency in the workload of the past MI. This results, an inaccurate choice of configuration values. ***System Performance Examiner*** is implemented as a small component that collects utilized data on all system resources (e.g., Disks and CPU) as well as the count of completed requests which allow the component to compute the response time and throughput. The monitor periodically inserts multiple sample requests into the requests that are sent by the client to the server. Two time stamps are used during a sample request is inserted and a response is received. The difference is used as the server side response time and the average response time is considered as the metric at certain point of time. ***Workload Intensity Supervisor and Workload Forecaster*** is the main components that make the algorithm more proactive as opposed to reactive. The use of effective forecasting algorithms enables the controller to acquire a more proactive behavior for the workload intensity value used by the model. It means that the system can make better configuration decisions to accommodate the future workload. To overcome the shortcomings mentioned in SLA module, a module is added responsible for short term workload forecasting. This module keeps a sliding window of N values for the last average arrival rates observed for the last N small sub intervals. Each of the sub intervals

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

is of length Δ seconds and Δ are chosen so that N x Δ does not exceed the length of a monitoring interval (some minutes).Workload Forecaster is responsible for prediction of request arrival rate. This information helps to compute the resources required for meeting SLA targets and resource utilization goals well in advance. Prediction can be defined based on historical data on resources usage, or statistical models derived from known application workloads. In addition, the particular method to estimate future load, the workload intensity supervisor alerts the workload forecaster and VCPU tuner when service request rate is likely to change. This alert contains the expected arrival rate and it must be issued before the expected time for the rate to change; So that the workload forecaster and VCPU tuner will have time to calculate changes in the system and the application provisioner will have the time to add or remove the required resources. ***Genetic Algorithm guided VCPU Regulator:*** It consists of two components namely VCPU regulator and genetic algorithm controller. VCPU regulator decides the number of VCPUs required meeting the SLA targets, with the help of genetic algorithm. As stated earlier, VCPU regulator finds out the best configuration by collecting response time of the entire VMs. This algorithm takes the desired SLA goals, the arrival and departure processes into account and performs a combinatorial search of the state space of possible configuration points in order to find optimal configuration. The cost function associated with each point in the space of configuration points is the SLA value of the configuration described in section 6.This component considers the system as a network of queues whose model parameters are obtained via workload intensity supervisor and workload forecaster components. The queuing network model considered by the system consists of client and server architecture. Clients in the model are represented by the generated requests, whereas application provisioner and application instances are the processing stations for these requests. Once the VCPU regulator determines the best configuration for the workload intensity levels provided by the various inputs, it sends reconfiguration commands to the appropriate VM. ***Control Considerations:*** The accuracy of the CPU time is scheduled to the virtual CPUs, depending on the time interval that is regarded. Hence in the case to complementing the control approach, it is recommended to take some additional considerations into account when designing and deploying the system. These considerations have a significant impact on the efficiency of the algorithm and on the performance of the entire system. By enabling the algorithm to dynamically regulate the frequency of its invocation over the fixed interval, the overall system performance and stability could be improved. In the case of a sudden surge in the workload, an adaptive controller algorithm responds to the change occurs in the external environment in advance. Figure 5 shows a simple *load adjustable* algorithm that can be used to dynamically vary the length of the monitoring interval. This algorithm sets the length of the MI as a multiple, *MI*, of the smallest possible interval MI_{min}. When the currently measured value of the SLA, SLA_{curr}, is less than or equal to the minimum value of SLA_{min} for the SLA, the monitoring interval is set to its minimum value MI_{min}. Otherwise, the monitoring interval is set to the multiple of MI_{min} according to the relative error between the SLA value,and SLA_{prev}, measured last time the algorithm was activated and the currently measured value of the SLA, SLA_{curr}.

If $SLA_{curr} < SLA_{min}$
do $MI \leftarrow MI_{min}$
else do
$\Omega = | (SLA_{curr} - SLA_{prev}) / SLA_{prev} |$
If $0 \le \Omega \le 0.05$ do $N = 10$
If $0.05 < \Omega \le 0.1$ do $N = 6$
If $0.1 < \Omega \le 0.2$ do $N = 5$
If $0.2 < \Omega \le 0.3$ do $N = 4$
If $0.3 < \Omega \le 0.4$ do $N = 3$
If $0.4 < \Omega \le 0.5$ do $N = 2$
If $\Omega > 0.5$ do $N = 1$
$MI \leftarrow N \times MI_{min}$
Finish

Figure 5: Adaptive Monitoring Interval (MI)

6 DESIGNS AND IMPLEMENTATION OF SYSTEM MODEL

This section presents the design and construction of a simulation framework with appropriate parameters for evaluating VCPU scheduling algorithms. The simulation framework is built by CSIM models and tool that makes the framework easy to understand and configure for various virtualization setups. The simulation model of a system is built as shown in figure 6 and experimented with policies as discussed earlier. The specification of the system is the cluster of the server machines in which each modeled as a multiple server queue. It incorporates necessary assumptions that are required for having a real performance model of cloud centers: (i) Random arrival process (Poisson process) (ii) Incorporates complex user requests by introducing super-tasks; (iii) Captures different delays imposed by cloud

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

centers on user requests; (iv) hyper exponential family distribution of the service time.

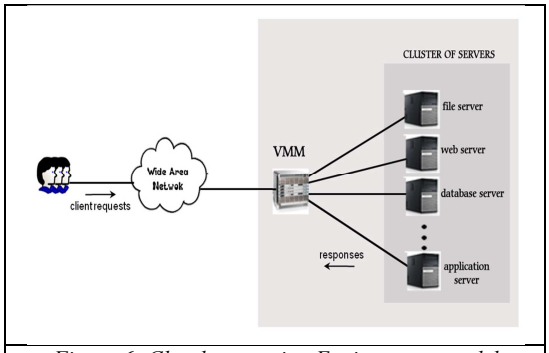

Figure 6: Cloud computing Environment model

Let us assume n virtual machines are consolidated into a m-core physical machine, given as vector VM = $\{vm_1, vm_2, ...vm_n \mid m \geq n\}$.The allocated /available resources for the virtual machines at some point of time is given as vector $R_{cpu} = \{r1_{cpu}, r2_{cpu}, ...rn_{cpu} \mid ri_{cpu} \geq 1\}$.The response time of the virtual machines at some instance ,to meet the SLA is given as vector $SLA_{resp} = \{resp_1, resp_2, \ldots resp_n\}$.Further, resource requirement of the virtual machines can be calculated as follows $ri_{cpu} = (resp_i / \sum_{j=1}^{j=n} respi) * (m-1)$.In real systems, it is common that the execution times of the parallel segments to be lengthened as more processor cores are simultaneously accessing one another. The value (m-1) models the effects of contention for access to multicores.

Researchers use simulators to evaluate new scheduling techniques under controllable and repeatable condition, which is impossible to reach in real cloud. Simulators are very useful as different setups and different data sets can be used to evaluate existing or proposed solutions as well as to compare their performance. Several simulation approaches for cloud systems have been proposed. Each differs in whether they focus on special applications or allow simulation of cloud systems in common. For an example, the simulation framework MRPerf [34] instruments the discrete event network simulator NS-3 [35] for studying performance and dependability of MapReduce [36]. The framework models network, node, and disk behavior in high aspect and thus allows evaluating the impact of network topology choices and network / node failures, but is limited to applications that use MapReduce. Similarly, the cloudsim toolkit [37] is a discrete event simulation engine provides simple implementation of common entities such as computational resources or users and also allows simulating virtual machines, VM scheduling, simple jobs, network topology, data storage and other useful functionality. However, provided implementations are too simple and it is necessary to extend these entities for more complex requirements. Further, it needs one to have in depth knowledge on cloudsim usage classes and java. In contrast to all the above, CSIM [38, 39] is a simulation model building toolkit, used by C/C++ programmers to implement process oriented, discrete event simulation model. These models mimic the operation of complex systems, to give modelers insight into the dynamic behavior of these systems. Because CSIM models are C/C++ programs, there are virtually no limits to the level of details, degree of complexity and size of the simulation models. CSIM processes are operated in an asynchronous parallel manner, mimicking the behavior of multiple entities which are active at the same time.

A CSIM program models a system as a collection of CSIM processes which interact with each other by using the CSIM structures. The purpose of modeling a system is to produce estimates of time and performance. The model maintains simulated time, so that it can yield insight into the dynamic behavior of the modeled system. In CSIM, entities are represented by processes, and queues are represented by facilities and other simulated resources. In these models, the complete system is represented by a collection of simulated resources and a collection of processes that compete for use of these resources. A major benefit of using a standard programming language to implement simulation models is that these models can be combined easily with other software components. Informally, the followings are the basic constructs of CSIM i) *Processes* - the active entities that request service at facilities, wait for events, etc. ii) *Facilities*- queues and servers reserved or used by processes iii) *Events* - used to synchronize process activities iv) *Mailboxes* - used for inter-process communications v) *Data collection structures* - used to collect data during the execution of a model vi) *Process classes*- used to segregate statistics for reporting purposes vii) *Streams* - flow of random numbers. In CSIM, it is easy to model a CPU and multi-core CPU as a facility and facility_ms respectively. A facility is a server with a queue for the waiting processes. In operation, an arriving process reserves a facility. If the server at the facility is not busy, it is assigned to do the requesting process. If the server is busy, the arriving process is placed in the queue and it is suspended. Normally, when the process is given to the server, it does a hold (service time) and releases

Journal of Theoretical and Applied Information Technology
31st October 2014. Vol. 68 No.3

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

the server at the facility. When this happens the queue is checked; there is a waiting process, and so on. Now a multi-server operates in the same way only when there are multiple servers. The service time is drawn from a probability distribution function exponential (service Time). All the CSIM resources have "inspector functions, which let one to get properties and statistics from the resources. For an example, the mean response time of the server [i] is given by server[i]->resp (). Similarly, the statistics and counters for a resource is cleared by calling the reset () method. The communication among CSIM processes is accomplished via CSIM mailboxes and synchronization of CSIM processes is accomplished via CSIM events.

Based on CSIM, a set of C++ classes, serverVM, VMM, scheduler, client and transactions been developed, which models the basic program and machine components of the system as shown in figure 5 and figure 6. Figure 7 depicts the UML class diagram, model processing units of a computing system. The working logic of the proposed system is given as a sequence diagram in figure 8. This is an open model, where the transactions arrive from outside to be processed with a variable of transactions that circulate among

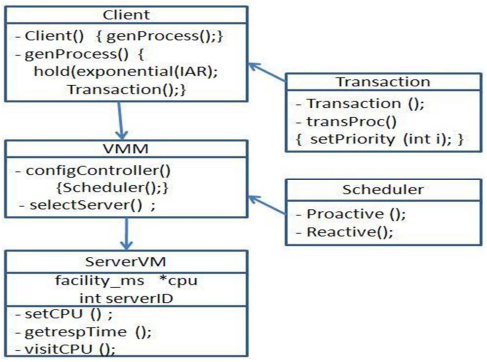

Figure 7: UML Class Diagram of the model

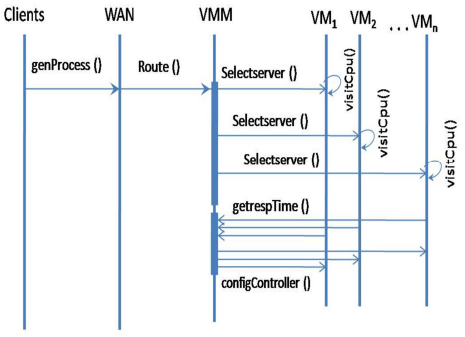

Figure 8 :UML Sequence Diagram of the model

the clients and the servers via internet. The sim process creates the model in which the activities start with the instantiation of the genProcess method in the Client class where each client has its unique id. The generator holds (delay) an exponentially distributed interarrival interval and generates a new transaction using genProcess that runs "forever". When genProcess creates a new transaction, it selects the server first and the transaction will visit using client class id. In this model, the servers are an array of server objects, and each server has a CPU resource, with multiple servers (think of each of these CSIM servers as a core). Each transaction notes its start time and visits the cpu on the serverVM object. When a transaction completes, it records its response time (the interval between its start time and its completion). The selectServer and configController method models the VMM as if it is in the proposed system (figure 5). The configController is elaborated more. For example, scheduler () functions have two different methods, reactive and proactive for resource allocation. Each scheduler needs to implement its scheduling policy and needs to register itself to this interface. Users can set the scheduler option during compile time by passing the parameter value of scheduler () and the scheduler implements the required resource allocation decisions. Based on the chosen controller, the sim process executes configController that in turn invokes appropriate scheduler which allocates sufficient resources. For instance, if the decision is to maintain 2 and 3 resource units for A and B client classes respectively, this component implements these decisions until the next decision is made. It has the access to the queue instances of each client class, resource units and other state variables. In regular interval, it executes the scheduler algorithm. Here, the design decision of centralized scheduler has been taken instead of each class taking the responsibility of scheduling. This is because, it is easy to track and validate the resource utilizations compared to a distributed algorithm. In addition, arbitrary size of controller intervals are considered to dynamically vary the length of the monitoring interval that is dynamically determined using the algorithm in Figure 6. In each control period, the VMM scheduler computes a weight for every VM. The weights are then truncated to integers and given to the VMs to set the number of VCPUs to be used for the next interval. Finally, the inspector function reports, statement of each server resources usage, and summary of the response times for all of the transactions. The user can use the given classes

Journal of Theoretical and Applied Information Technology
31st October 2014. Vol. 68 No.3

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

to implement the required simulation depending on their requirements.

7. EXPERIMENTS AND EVALUATION

This section, presents the result of an extensive evaluation of the proposed workload aware adaptive scheduler approach based on synthetic workloads. Due to the space constraints, only selected experiments are presented. Thus, only consolidated servers average response time for workload aware reactive and proactive algorithm with fixed and variable time interval is given. Table 1 indicates that the effective response time differs from the MI for both proactive and reactive. As far as reactive algorithm is concerned, it gives good results for MI is 1 minute (Table 1), and for all other cases not withstands with SLA.This is because the resource allocations are carried out precisely at the time interval, and in all other cases no such precision is maintained. Subsequently, for this case there is no improvement in their frequency of execution. Likewise, for the Exponential Smoothing case the effective response time is 0.3821 when MI=2 (Table 1 and Figure 9).

Table 1:Comparison of Various Workload Aware Algorithms for the Fixed Monitoring Interval

S.No	Model	Time (minutes)	Frequency of algorithm invocation	resp time (minutes)
1	Reactive	1	100	0.3967
2	Proactive(ExpS)			0.3899
3	Proactive(WMA)			0.3810
4	Reactive	2	50	0.3998
5	Proactive(ExpS)			0.3821
6	Proactive(WMA)			0.3826
7	Reactive	3	33	0.4119
8	Proactive(ExpS)			0.3890
9	Proactive(WMA)			0.4090
10	Reactive	4	25	0.4292
11	Proactive(ExpS)			0.4207
12	Proactive(WMA)			0.4256
13	Reactive	5	20	0.4315
14	Proactive(ExpS)			0.4190
15	Proactive(WMA)			0.4002
16	Reactive	6	17	0.4521
17	Proactive(ExpS)			0.4421
18	Proactive(WMA)			0.4456

This is because, the Exponential Smoothing forecasted the workload, predetermined resource allocation that made the server ready for maintaining SLA. In addition, as the frequency of execution time is saved 50%, steeled to the service time. For the same case, when the MI=3 (Table 1) close result with 50% execution frequency improvement with a 2% deviation in their net response time can be seen.

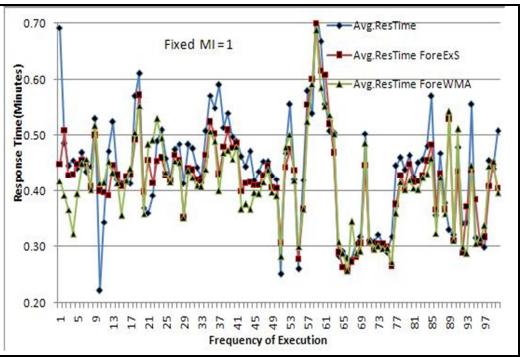

Figure 9: SAVIMM Response Time for the fixed Monitoring Interval *of 1 minute*

Similarly, for Weighted Moving Average case it gives good response time for both MI= {1, 2} (Table 1 and Figure 10) with least deviation of 0.5% and 34% execution frequency reduction. This is because as mentioned earlier this forecasting algorithm exploits the workloads characteristics of constant intensity for quite a while before changing significantly. Thus for the fixed MI, forecasting algorithm provides good results especially for WMA. As far as MI is concerned the accuracy of the CPU time is scheduled to the virtual CPUs depending on the time interval that is regarded. That is, if the MI is too small and the workload amount is relatively steady, the SAVIMM algorithm will be executed too frequently with little or no effect. At the same time, if the MI is too large and the workload amount varies very quickly, the controller will not run effectively. Thus, a MI that adjusts itself to the workload strength can be more effective than a fixed MI. Table 2 gives the comparative results of the adaptive MI algorithms. Figure 11 shows all algorithms work quite nicely and withstand the SLA over fixed MI.

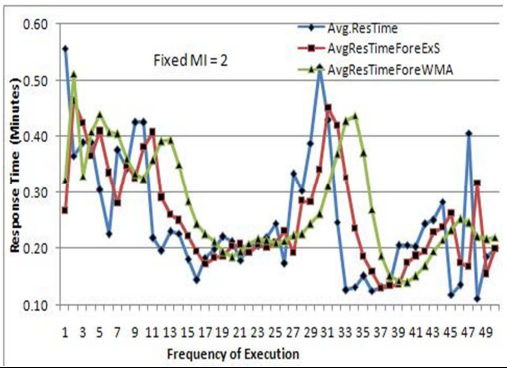

Figure 10: SAVIMM Response Time for the fixed Monitoring Interval of 2 minute

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

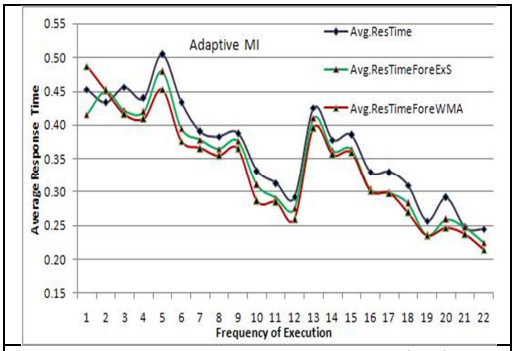

Figure 11: SAVIMM Response Time for the Adaptive Monitoring Interval

Table 2: Comparison of Various Workload Aware Algorithms for the Adaptive Monitoring Interval

S.No	Model	Frequency of algorithm invocation	resp time (minutes)
1	Reactive	38	0.3883
2	Proactive(ExpS)	26	0.3751
3	Proactive(WMA)	22	0.3579

Limitations: Though this flexible simulation framework aids the evaluation of VCPU scheduling algorithms, the framework at this current state is still primitive that needs improvements in order to (i) include other resource requirements, such as memory, network bandwidth, and (ii) represent more synchronization mechanisms (iii) try adaptive MI algorithm with different values (iv) use a variety of real-world arrival traces to generate the request rate.

8. CONCLUSION AND FUTURE RESEARCH

Virtualization is becoming an increasingly important technology in large part because it offers the promise of allowing more efficient use of computing resources. However, the behavior of a VM often differs significantly from a physical system, leading to performance degradation for applications running in a VM. Furthermore, virtualization is a rapidly evolving field, as hardware support continues to be added and adopted to bring the performance of virtualized systems closer to that of native execution. Hence, this paper, propose novel adaptive meta-heuristics based scheduling policies for provisioning the VCPU resources among competing VM service domains in a cloud. The objective of such provisioning is to guarantee to budge to SLA for each domain, with respect to the diverse workloads on-the-fly. The framework is built upon CSIM models and tool, making the framework easy to understand and configure for various virtualization setups. We demonstrate the usefulness of the framework by evaluating three VCPU scheduling algorithms: proactive, reactive and adaptive. We evaluated how periodic and aperiodic execution of control actions can affect policy performance and speed of convergence. Simulation based experimental results using synthetic workload models indicated that the proposed provisioning technique can detect changes in workload intensity (arrival pattern and resource demands) that occur over time and allocate resources accordingly to achieve application SLA targets. In addition the results of the experiments have shown that the proposed Weighed Moving Average algorithm combined with the adaptive MI policy significantly outperforms other dynamic VM consolidation algorithms; In regard to the SLA metric due to a substantially reduced level of response time violations and the frequency of algorithm invocation.

Regarding possible future works, first we plan to evaluate the proposed system in a real Cloud infrastructure like Open Stack. Second is the investigation of more complex workload models such as slowly varying, quickly varying (synthetic), big spike, dual phase variations which are drawn from real-world traces; that will leverage these workload models with increasing server consolidation ratio. Third apart from CPU sharing we would to like extend to cover the allocations of the resources like main memory, network I/O. Fourth, we can use multiple alternative forecasting methods in parallel; select which method to trust based on its accuracy in recent time horizons. Finally we will extend the model to support adaptive scheduling techniques in addition with the resource allocation for the diverse workloads.

REFERENCES

[1]. Makhija V, Herndon B, Smith P, Roderick L, Zamost E and Anderson J, "VMmark: A Scalable Benchmark for Virtualized Systems", *Technical Report VMware-TR-2006-002*, Sep 2006.

[2]. James E .Smith, Ravi Nair, "Virtual Machines: Versatile Platforms for Systems and Processes", ISBN: 1-55860-910-5, Elsevier.

[3]. Sukwong O, Kim H, "Is co-scheduling too expensive for SMP VMs?", *In Proc. of the 6th conf. on Computer systems*, ACM, 2011.

[4]. Suresh.S, Kannan.M, "A Study on System Virtualization Techniques", *International Conference on HI-TECh Trends In Emerging*

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Computational Technologies (ICECT2014), Tamilnadu, India, February 20- 21, 2014.

[5]. Herbst N.R,Huber N,Kounev S, and Amrehn E, "Self-Adaptive Workload Classification and Forecasting for Proactive Resource Provisioning", *In Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering (ICPE 2013)*, Prague, Czech Republic, April 21-24. 2013.

[6]. Sivanandam S.N, Deepa S.N, "Introduction to Genetic Algorithms", *Springer*, ISBN 978-3-540-73189-4, 2008.

[7]. Goldberg D. E, "Genetic Algorithms in Search", *Optimization & machine Learning,* Addison-Wesley, 1989.

[8]. Cherkasova L,Gupta D, and Vahdat A, "Comparison of the three CPU Schedulers in XEN", *SIGMETRICS Perform. Eval. Rev*., Vol.35, no.2, pp.42-51, 2007.

[9]. Xue Liu, Xiaoyun Zhu, Pradeep Padala, Zhikui Wang, and Sharad Singhal, "Optimal Multivariate Control for Differentiated Services on a Shared Hosting Platform", *Proceedings of the 46th IEEE Conference on Decision and Control (CDC'07)*, December 2007.

[10]. O.Tickoo., R.Iyer., R.Illikkal., and D. Newell, "Modeling Virtual Machine Performance: Challenges and Approaches", In HotMetrics, 2009.

[11]. Ongaro D, Cox A. L., and Rixner S, "Scheduling I/O in Virtual Machine Monitors", *Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments(VEE '08)*,ACM,pp.1-10, New York, USA,2008.

[12]. Padala P, Hou K.Y, Shin K.G, Zhu X, Uysal M, Wang Z, Singhal S, and Merchant A, "Automated Control of Multiple Virtualized Resources", *Proceedings of EuroSys '09*, ACM, 2009.

[13]. Weng C., et al., "The Hybrid Scheduling Framework for Virtual Machine Systems," *Proc. of the 2009 ACM SIGPLAN/SIGOPS intl'conf. on Virtual execution environments*, New York, NY, USA, 2009.

[14]. Chieu T. C, Mohindra A, Karve A. A, and Segal A, "Dynamic Scaling of Web Applications in a Virtualized Cloud Computing Environment", *Proceedings of the 6th International Conference on e-Business Engineering (ICEBE'09)*, 2009.

[15]. Watson B. J, Marwah M, Gmach D, Chen Y, Arlitt M, and Wang Z, "Probabilistic Performance Modeling of Virtualized Resource Allocation," *in ICAC*, 2010.

[16]. Jung G, Hiltunen M, Joshi K, Schlichting R, and Pu C, "Mistral: Dynamically Managing Power, Performance, and Adaptation Cost in Cloud Infrastructures", *In ICDCS*, 2010.

[17]. Lim S.H, Huh J.S, Kim Y.J, Shipman G. M, and Das C. R, "A Quantitative Analysis of Performance of Shared Service Systems with Multiple Resource Contention", *Technical report*, 2010, [Online]. Available: http://www.cse.psu.edu/research/publications/tech-reports/2010/cse-10-010.pdf.

[18]. Gong Z, Gu X, "Pac: Pattern-Driven Application Consolidation for Efficient Cloud Computing", *18th Annual IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'10)*, pp.24–33, August 2010.

[19]. Li K, "Optimal Load Distribution for Multiple Heterogeneous Blade Servers in a Cloud Computing Environment", *Parallel and Distributed Processing Workshops and PhD Forum, 2011 IEEE International Symposium* , pp.948–957, 2011.

[20]. Huber N, Quast M. von, Hauck M, and Kounev S, "Evaluating and Modeling Virtualization Performance Overhead for Cloud Environments", *Proceedings of the International Conference on Cloud Computing and Services Science (CLOSER 2011)*, SciTePress, pp. 563 - 573. Noordwijkerhout, Netherlands, May 7-9, 2011.

[21]. Kousiouris G, Cucinotta T, and Varvarigou T, "The Effects of Scheduling, Workload Type and Consolidation Scenarios on Virtual Machine Performance and their Prediction through Optimized Artificial Neural Networks", *Journal of Systems and Software*, Vol. 84, no. 8, 2011.

[22]. Lu L, Zhang H, Jiang G, Chen H, Yoshihira K, and Smirni E, "Untangling Mixed Information to Calibrate Resource Utilization in Virtual Machines", *International Conference on Autonomic Computing*, 2011.

[23]. Nikolaus Huber et.al., "Model-based Self-Adaptive Resource Allocation in Virtualized Environments", *SEAMS '11*, May 23-24, 2011.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

[24]. Guenter B, Jain N, and Williams C, "Managing Cost, Performance, and Reliability Tradeoffs for Energy-Aware Server Provisioning", *Proceedings IEEE INFOCOM*, pp. 1332-1340, 2011.

[25]. Sethi S, Sahu A, and Jena S. K, "Efficient Load Balancing in Cloud Computing using Fuzzy Logic", *IOSR Journal of Engineering,* vol. 2,no. 7, pp. 65-71, 2012.

[26]. Kraft S, Casale G, Krishnamurthy D, Greer D, and Kilpatrick P, "Performance Models of Storage Contention in Cloud Environments", *SoSyM*, 2012.

[27]. Rao J, Wei Y, Gong J, and Xu C.Z., "QoS Guarantees and Service Differentiation for Dynamic Cloud Applications", *IEEE Transactions on Network and Service Management,* Vol. 10, no. 1, March 2013.

[28]. Chia-Ying Tseng and Kang-Yuan Liu, "A Modified Priority Based CPU Scheduling Scheme for Virtualized Environment", *International Journal of Hybrid Information Technology*, Vol. 6, no. 2, March, 2013.

[29]. Armbrust M, Fox A, Griffith R, Joseph A. D, Katz R, Konwinski A, Lee G, Patterson D, Rabkin A, Stoica I, and Zaharia M, "A view of Cloud Computing", *Communication of ACM*,Vol.53,pp.50–58, April 2010.

[30]. Eduardo Ciliendo,Takechika Kunimasa , "Linux Performance and Tuning Guidelines", *Redpaper*,IBM,July 2007.

[31]. Suresh S, Kannan M, "A Performance Study of Hardware Impact on Full Virtualization for Server Consolidation in Cloud Environment", *Journal of Theoretical and Applied Information Technology(JATIT)*, Vol. 60, no.3, 28th February 2014.

[32]. Chisnall D, "The Definitive Guide to the XEN Hypervisor", Prentice Hall, 2007.

[33]. Schwiegeishohn U and Yahyapour R, "Improving First-Come-FirstServe Job Scheduling by Gang Scheduling", *Job Scheduling Strategies for Parallel Processing*, Springer Berlin, Heidelberg, 1998.

[34]. Wang G, Butt A. R, Pandey P, and Gupta K, "A Simulation Approach to Evaluating Design Decisions in MapReduce Setups", *International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS)*, IEEE,pp.1–11,2009.

[35]. Various authors, "The Network Simulator NS-3",[Online].Available:http://www.isi.edu/nsnam/ ns/.(Retrieved in January 2014).

[36]. Dean J and Ghemawat S, "Mapreduce: Simplified Data Processing on Large Clusters", *Communication of ACM*, Vol.51,pp.107–113, January 2008.

[37]. Buyya R, Ranjan R, and Calheiros R. N, "Modeling and Simulation of Scalable Cloud Computing Environments and the CloudSim Toolkit: Challenges and Opportunities", *Proceedings of the 7th High Performance Computing and Simulation (HPCS 2009) Conference*, 2009.

[38]. Edwards G and Sankar R, "Modeling and Simulation of Networks using CSIM", *Simulation*, Vol.58, no.2, pp.131-136, 1992.

[39]. Schwetman H, "CSIM19: A Powerful Tool for Building System Models", *Proceedings of the 2001 Winter Simulation Conference*, pp.250-255, 2001.