Research Journal of Applied Sciences, Engineering and Technology 9(5): 380-390, 2015
DOI:10.19026/rjaset.9.1417
ISSN: 2040-7459; e-ISSN: 2040-7467
© 2015 Maxwell Scientific Publication Corp.
Submitted: September 22, 2014 Accepted: November 10, 2014 Published: February 15, 2015

Research Article

Workload Known VMM Scheduler for Server Consolidation for Enterprise Cloud Data Center

[1]S. Suresh and [2]S. Sakthivel
[1]Department of Computer Science and Engineering, Adhiyamaan College of Engineering, Hosur-635109,
[2]Department of Computer Science and Engineering, Sona College of Technology, TPTC Main Road, Salem-636005, Tamil Nadu, India

Abstract: This study proposes a novel adaptive meta-heuristics based scheduling policies for provisioning the VCPU resources among competing VM service domains in a cloud. Such provisioning guarantees to Service Level Agreement for each domain, with respect to the diverse workloads on-the-fly. The framework is built on CSIM models and tools, making it easy to understand and configure various virtualization setups. The study demonstrates the usefulness of the framework by evaluating proactive, reactive and adaptive VCPU scheduling algorithms. The paper evaluates how periodic/aperiodic execution of control actions can affect policy performance and speed of convergence. The periodic reactive resource allocation is used as the baseline for analysis and the average response time is the performance metric. Simulation based experiments using variety of real-world arrival traces and synthetic workloads results that the proposed provisioning technique detects changes in arrival pattern and resource demands and allocates resources accordingly to achieve application SLA targets. The proposed model improves CPU utilization and makes the best tradeoff between resource utilization and performance from 2 to 6% comparing with the default VMM scheduler configurations for diverse workloads. In addition, the results of the experiments show that the proposed Weighed Moving Average algorithm combined with the aperiodic policy significantly outperforms other dynamic VM consolidation algorithms in all cases, in regard to the SLA metric due to a substantially reduced level of response time violations and the frequency of algorithm invocation.

Keywords: Cloud computing, cloud workload, server consolidation, server virtualization, simulation, VMM scheduling

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 acts 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 (VMs) 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.

All request is delivered to the 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. Assigning resources to VMs in a static manner solves this issue. However, static allocation becomes inefficient under varying load. The second problem is Quality of Service (QoS). As cloud hosted services and applications are user oriented, QoS has a great impact on the 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., increasing resource utilization effectively and meeting Service Level an Objective (SLOs) in a shared virtualization environment is a great challenge to the scheduler for achieving 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

Corresponding Author: S. Suresh, Department of Computer Science and Engineering, Adhiyamaan College of Engineering, Hosur-635109, Tamil Nadu, India

a better trade off and 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 helps to decrease the level of carbon-dioxide 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 provides certain SLA for an application becomes problematic due to the 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.

Virtualization delivers dramatic resource usage and cost cuttings. However, overall data center efficiency metrics may not meet the requirement of the state-of-art hardware and software advancements for the cloud data centers workloads. Thus, a research was done to evaluate, 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? and How rapidly and at what level is the reconfiguration triggered?

As system level scheduling implementation requires deep understanding on hardware architecture, as well as low level programming and debugging, evaluating new scheduling techniques under various, controllable and repeatable conditions are impossible in real Cloud. This makes qualitative evaluation of scheduling algorithm difficult and requires significant effort. To address this problem, the cloud virtualization environment, is simulated to evaluate the performance of the defined algorithm using CSIM (Schwetman, 2001) simulation toolkit and a C/C++ library that allows assembling a complete virtualization system with flexible configurations.

In summary, the contribution of this study is multifold and our works are as follows:

- Study on server virtualization and its various solutions
- Study on state of the art work in VMM scheduling
- Conducting performance case study that focus on how the performance is affected by the amount of CPU allocation
- 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
- Investigate and evaluate more scheduler configurations and to measure the performance

Background information and definition: This section presents some background information and definitions on server virtualization and its solutions, chip multithreading, combinatorial search techniques and workload forecasting.

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 a single hardware. VMM, a software layer separates underlying hardware from the software running on top of it and creates a notion of the hardware for a virtual machine. Thus, VMM creates and manages hardware units for the virtual (Smith and Nair, 2005; Suresh and Kannan, 2014a) classified Virtualization Technology into three major approaches, based on the performance and support of the hardware platforms and methods of executing guest OS code with or without hardware access. The first approach is called binary translation, in which the VMM modifies the guest's binary image at runtime to get processor control when guest software attempts to perform a privileged operation. The VMM can then emulate privileged operation and return control to guest software. The second approach is called paravirtualization, in which the guest's kernel is modified to cooperate with the VMM when performing privileged operations. The third one is hardware assisted virtualization, in which the VMM uses processor extensions such as AMD-V and Intel-VT to intercept and emulate privileged operation in the guest.

Scheduling in VMM: Aforesaid, three virtualization approaches infer, VMM is an important entity as it routes any request by the guest to access the hardware. The VMM gives a portion of the full/partial physical machine resources to each guest domain, thus multiple guests must share the available resources. Thus, with diverse workload and demand, it grants each guest OS a limited amount of specified resource. 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.

Chip Multithreading (CMT) and Virtual CPU (VCPU): CMT combines the resources of multiple CPUs in a single host system, in which all the processors behave identical. CMT creates space for any processor; to execute any processes in the system, improving overall system performance. Subsequently, logical domains technology provides flexible assignment of hardware resources to domains, with options for specifying physical resources for a corresponding virtual resource say physical CPU to Virtual CPUs (VCPUs). A domain can have 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. This method avoids the frequent context switches usually VMM implemented by to run several guests on a CPU and to intercept privileged operations. It results an impressive benefit in terms of simplicity and reduction of the overhead.

Dynamic resource allocation: Dynamic resource allocations are quite essential for adaptive computing. Generally, there are two approaches namely reactive resource allocation and proactive resource allocation. Reactive allocation is 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 use fragmented spare resources in a datacenter during load spikes quickly causes host level over provisioning. When predictions are accurate, this scheme provides very good performance. Forecasting can be achieved by applying many techniques (Herbst *et al*., 2013). However, the paper deals with the popular forecasting techniques namely weighted moving averages and exponential smoothing.

Genetic Algorithm (GA): As scheduling belong to combinatorial optimization problems, optimization algorithms are default choice to find optimal solutions. Compared to conventional heuristics, GA (Sivanandam and Deepa, 2008) is 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 and each of them representing a possible solution. The GA, then manipulates the most promising strings searching for improved solutions, through a simple cycle on four stages:

- Creation of a "population" (randomness) of strings
- Evaluation of each string (reproduction)
- Selection of "best" strings using fitness function
- 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. As a result, it maximizes the system utility.

LITERATURE REVIEW

VMM resource allocation, scheduling and analysis of virtualization performance are the most important problems in server virtualization research for 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. Some of the related works are discussed below.

Cherkasova *et al.* (2007) 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 is performance-oriented and does not act in accordance with configured values. They also found that choosing the right parameters for individual virtual machines is crucial in order to get the desired application performance.

Weng *et al.* (2009) 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. Similarly, Watson *et al.* (2010) proposed a probabilistic performance model using quantile regression. They made it possible to predict the response time of the web authentication benchmark depending on the allocated resources at the VM level. However, they did not provide the performance relevant factors explicitly.

Lim *et al.* (2010) proposed a mathematical model to characterize workload using multiple resource usages. In the model, host is characterized as a collection of *m* resource queues. The application is characterized 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. This method assumed, to run multiple applications together and not to take longer time than sequential execution time. But, this is not true in real virtualized environments as severe contention between two VMs may lead to slowdown of more than twice. Jung *et al.* (2010) developed a multi-level resource allocation framework that adapts 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 good performance, transient costs and nominal 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. Similarly Lu *et al*. (2011) used analytical queuing models to quantify the slowdown of virtualized applications in server consolidation scenarios. They concluded that using the total CPU time as distribution factor to derive workload specific virtualization overhead typically results in an uneven estimation at best.

Huber *et al*. (2011a) proposed a novel approach for self-adaptive resource provisioning in virtualized environments based on online architecture level performance models. They investigated the use of such models as a means for online performance prediction which predicts the effects of changes in user workloads and the effects of particular reconfiguration actions undertaken to obey SLA. By using virtualization techniques, they applied these allocation changes to evaluate the use of such models for online performance prediction. However, this present research, investigates the influences of virtualization on system performance to integrate the gained insights into the proposed performance models are investigated.

Kousiouris *et al*. (2011) used Artificial Neural Network, to predict the impact of the performance of real time scheduling parameters, VM deployment and workload type on the system performance based on measurements using various MATLAB benchmark tests. Similarly, Sethi *et al*. (2012) present a new fuzzy controller for load balancing in cloud computing. For them, the cloud computing requires 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, a generalized technique is applied in this present research so as not to make any guess regarding the CPU speed. Rao *et al*. (2013) proposed a fuzzy controller for allocating virtualized resources with respect to the application response time. The works of Sethi *et al*. (2012) and Rao *et al*. (2013) consider only the response time and its deviation from the SLA value as input parameters to the controller. Instead, this present research 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.

Chia-Ying and Kang-Yuan (2013) proposed a solution for the synchronization problem of a server consolidation by modifying the XEN Credit Scheduler. They added a new priority module to the scheduler in order to avoid the scheduling decisions and for synchronization. This new priority module allows VCPUs to preempt and to run at the next time slice without impacting overall system fairness. Through this way, the threads in the concurrent program can be synchronized. Consequently, proposed scheduler works enhances the performance in concurrent workload greatly by decreasing CPU allocation errors. However, it incurs minor performance drop in a parallel workload due to the extra overhead of finding the most urgent work from other PCPUs. The work by Li and Zheng (2014) on web application scalability originated for static load balancing solution with server clusters, but the dynamic scaling of web applications and energy consumption of resources in virtualized Cloud computing has not been impressively considered by Liu *et al*. (2014).

Thus, this survey proves that earlier works have missed a good opportunity of cost and performance optimization by disregarding cloud workload aware resource allocation at which computer processor technology has progressed. In addition, it also draws a conclusion that, no works exploit the genetic algorithm extensively.

METHODOLOGY

Resource influence on VM and VMM schedulers: As performance provision is the major concern of VMM scheduler in cloud, this section provides:

- A quantitative case study that focus on how the VM performance is affected by adapting a VM's CPU capacity
- Two qualitative case studies of widely popular virtualization VMM schedulers

Performance influencing factors: As virtualization introduces dynamics and increases flexibility, a variety of additional factors can influence the performance of virtualized systems. Having analyzed major representative virtualization platforms, Huber *et al*. (2011b) and Armbrust *et al*. (2010) abstracted a generic performance model of VM performance influencing factors. 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. The number of VMs, Virtual CPUs allocated to a VM and resource over commitment are chief among them.

Performance study:

Impact of CPU allocation: CPU cores are one of the main sources of performance interference. This complexity is demonstrated by executing experiment, targeted at the application level in virtual machine environment by setting CPU limit at different levels. The performance of the virtualized applications (OLTB) are measured while varying the VM's CPU

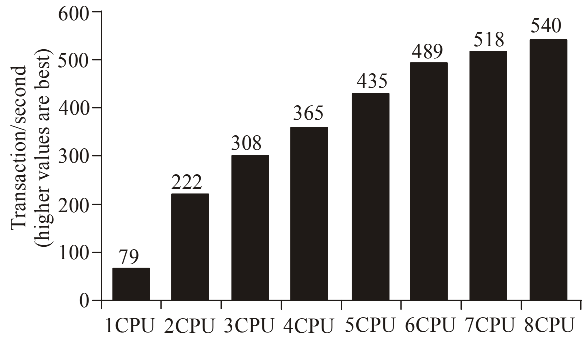

Fig. 1: SysBench database transactions

limit from 1 to 8 cores. All the experiments were conducted on physical hardware configured with AMD FX 8-Core Black Edition FX-9590. Both host and virtual machine are configured with 8 VCPUs and 4 GB RAM, 50 GB HDD with Ubuntu 14.04 LTS (Trusty Tahr). The virtualization solutions considered for the experiment is XEN 5.0. For the XEN machines, virtual NICs use the default bridged network driver. MySQL SysBench (Cillendo and Kunimasa, 2007; Suresh and Kannan, 2014b) is a modular, cross platform and multi-threaded benchmark tool for evaluating OS parameters. These parameters 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 and simultaneous queries to the MySQL database with zero think time. A simple database that fits entirely in memory is used. As a result, these workloads saturated the virtual CPU and generated network activity, with very little disk I/O. For various numbers of threads, the experiment was conducted and the result is given in the Fig. 1. As seen from the graph, the benchmarks workloads behave linearly and the performance slope is different at various CPU allocation ranges. SysBench performance varies almost linearly with CPU allocation. But, at some time, the saturated point is visible because of resource over provisioning. Thus, this data reveal the fact that virtualized workloads can have quite different performance curves with respect to number of CPU allocation. This leads to the conclusions that direct allocation of resources to a VM has an impact on the performance of hosted application and choosing appropriate control knobs to handle resource allocation for a VM is critical to ensure desirable performance and to create a robust model.

State of the art VMM schedulers: VCPU scheduling remains a challenge for Virtualization technologies, especially with hypervisors starting to host Chip Multithreading VMs. As the implemented prototype in this study is generic, it briefly discusses the main features of two popular VMM scheduler's algorithms in specific.

CPU scheduling algorithms in XEN: XEN supports three different types of schedulers (Chisnall, 2007; Cherkasova *et al*., 2007) namely BVT, SEDF and Credit Scheduling. BVT is a proportional share scheduler 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. 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. SEDF is an another algorithm, in which, the domains request a minimum time slice for communication. The request is a tuple 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. 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 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 out by their remaining credit and the scheduler will select the VCPU that has more credit to run. 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.

VMware ESX server VCPU scheduling algorithms: The default Round-Robin approach by KVM or Virtual Box hypervisors cause additional synchronization latency for guest VM due to VCPU preemption. In order to eliminate this synchronization latency, VMware applies a co-scheduling algorithm (Sukwong and Kim, 2011), which uses a concept similar to gang scheduling (Schwiegeishohn and Yahyapour, 1998). 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.

Having analyzed two major representative virtualization platforms, one can infer that current commercial resource management tools provide only partial solutions to VCPU scheduling problem 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.

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:

- An improved overall system performance to withstand SLA
- A better utilization of system resources. To achieve these goals, a computer system needs to be checked regularly

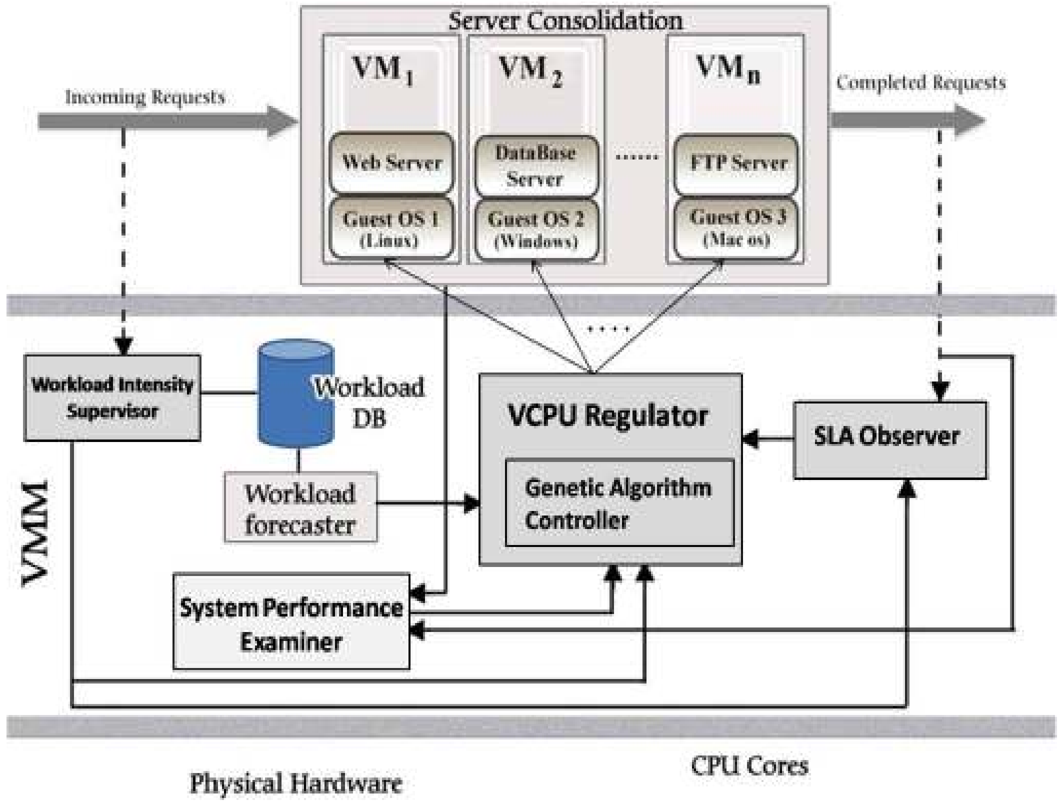

Fig. 2: Proposed system framework components

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 an algorithm, at (a) periodic intervals, called Monitoring Interval (MI), to determine the best configuration (suitable number of VCPUs) for each VM with the help of a meta-heuristic algorithm under varying workload. As a result of running the controller algorithm, reconfiguration commands are generated to instruct the system to change its configuration.

The general control approach: This section presents the control architecture of the proposed adaptive systems. It describes the system architecture and components and how they interact with one another. In addition, some control decisions regarding workload forecasting and frequency of control are also discussed. The architecture of the system model is best illustrated in Fig. 2. It has five main components namely workload intensity supervisor, workload forecaster, SLA observer, system performance examiner, genetic algorithm guided VCPU regulator.

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 (VM) 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. System Performance Examiner is implemented as a component that collects utilized data on all system resources (e.g., 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 are the two main components that make the algorithm more proactive as opposed to reactive. It indicates that the system can make better configuration decisions to accommodate the future workload. Workload Forecaster is responsible for prediction of request arrival rate. It 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 time to add or remove the required resources. Genetic Algorithm guided VCPU Regulator consists of two components namely VCPU regulator and genetic algorithm controller. The dynamic balancing component, VCPU regulator reconfigures the individual VM's VCPU demand based on the resource requirement. 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, the balancing component will be more aggressive in the balancing process. 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. This component considers the system as a network of queues whose model parameters are obtained via workload intensity supervisor and workload forecaster components. 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.

The accuracy of the CPU time is scheduled to the virtual CPUs, depending on the time interval that is regarded. Thus, 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. These considerations have a significant impact on the efficiency of the algorithm and on the performance of the entire system. Furthermore, 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.

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. Let us assume n virtual machines are consolidated into an m-core physical machine, given as vector VM = {vm_1, vm_2, ...vm_n |m≥n}. The allocated/available resources for the virtual machines at some point of time is given as vector $R_{cpu} = \{r1_{cpu}, r2_{cpu},...rn_{cpu} | ri_{cpu} \geq 1\}$. The response time of the virtual machines at some instance, to meet the SLA is given as vector SLA_{resp} = {$resp_1$, $resp_2$, …$resp_n$}. Further, resource requirement of the virtual machines is calculated as $ri_{cpu} = (resp_i / \sum_{j=1}^{j=n} respi) * (m-1)$. Here the value (m-1) models the effects of contention for access to multicores.

The simulation model of a system is built as the cluster of the four server machines in which each cluster is modeled as a multiple server queue. It incorporates necessary assumptions that are required for having a real performance model of cloud centers:

- Random arrival process
- Incorporates complex user requests
- Captures different delays imposed by cloud centers on user requests
- Hyper exponential family distribution of the service time

As mentioned earlier, the simulation framework is built by CSIM models and tool, used by C/C++ programmers to implement process oriented, discrete event simulation model. 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. 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 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 lets one to get properties and statistics from the resources. For 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 (used for inter-process communications) and synchronization of CSIM processes is accomplished via CSIM events (used to synchronize process activities).

Based on CSIM, a set of C++ classes, server VM, VMM, scheduler, client and transactions have been developed, which models the basic program and machine components of the system as shown in Fig. 2 and system specification. This is an open model, where the transactions arrive from outside to be processed with a variable of transactions that circulate among 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; 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 (Fig. 2). 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 periodic/aperiodic 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 is considered to dynamically vary the length of the monitoring interval that is dynamically determined. 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 to implement the required simulation depending on their requirements.

RESULTS AND DISCUSSION

This section involves performance evaluation of a range of VCPU provisioning policies.

One of the key areas to be looked at in the cloud is how a cloud service provider handles various capacity requirements in different time zones at a given time. In situations like these, cloud service providers must consider all these geo-specific business requirements and design the service model in which there are none or the least possible bottlenecks or resource conflicts. Thus, to make experiments reproducible, it is important to rely on a set of input traces to reliably generate the workload, which would allow the experiments to be repeated as many times as necessary. Thus, it is also important to use diverse workload traces collected from a real system such as slowly varying (ITA, 1998), Large variations (NLANR, 1995) and artificially generated quickly varying (synthetic) workload, as this would help to reproduce a realistic scenario. Furthermore, this implies that the overall system workload is composed of multiple independent heterogeneous applications, which also corresponds to a cloud environment. In our implementation testbed, the four servers can together handle at most 200 requests per second; thus, we scale the arrival traces such that the maximum request rate into the system is 200 req/sec. Further, we scale the duration of the traces to 2 h.

Due to the space constraints, only consolidated servers average response time for workload aware reactive and proactive algorithm with fixed and variable time interval is given in Table 1. For synthetic workload, 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 min 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). 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. Similarly, for Weighted Moving Average case, it gives good response time for both MI = {1, 2} (Table 1) with least deviation of 0.5 and 34% execution frequency reduction. This is due to the this forecasting algorithm, which exploits the workloads characteristics of constant intensity for quite

Table 1: Comparison of various workload aware algorithms for the fixed monitoring interval

			Response time (min) for various workloads		
Model	Time (min)	Frequency of algorithm invocation	Quickly varying (synthetic)	Slowly varying	Large variations
Reactive	1	120	0.3967	0.3786	0.3831
Proactive (ExpS)			0.3899	0.3739	0.3790
Proactive (WMA)			0.3810	0.3783	0.3779
Reactive	2	60	0.3998	0.3816	0.3862
Proactive (ExpS)			0.3821	0.3768	0.3825
Proactive (WMA)			0.3826	0.3825	0.3781
Reactive	3	40	0.4119	0.3907	0.3960
Proactive (ExpS)			0.3890	0.3797	0.3916
Proactive (WMA)			0.4090	0.3858	0.3820
Reactive	4	30	0.4292	0.4104	0.4151
Proactive (ExpS)			0.4207	0.4090	0.4119
Proactive (WMA)			0.4256	0.4009	0.4071
Reactive	5	24	0.4315	0.4215	0.4240
Proactive (ExpS)			0.4190	0.4108	0.4129
Proactive (WMA)			0.4002	0.4047	0.4036
Reactive	6	20	0.4521	0.4456	0.4472
Proactive (ExpS)			0.4421	0.4229	0.4277
Proactive (WMA)			0.4456	0.4233	0.4289

Table 2: Comparison of various workload aware algorithms for the adaptive monitoring interval

Model	Frequency of algorithm invocation (synthetic/slowly varying /large variations))	Response time (min) for various workloads		
		Quickly varying (synthetic)	Slowly varying	Large variations
Reactive (base)	38/27/32	0.3883	0.3743	0.3876
Proactive (ExpS)	26/17/23	0.3751	0.3712	0.3794
Proactive (WMA)	22/19/20	0.3579	0.3727	0.3614

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 proposed 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, an MI itself adjust to the workload strength can be more effective than a fixed MI. Table 2 gives the comparative results of the adaptive MI algorithms.

In the case of real workload traces, for slowly varying traces, exponential smoothing works much better. For the fixed time interval even for the MI = {1, 2, 3}, there is no huge performance variance in it. Comparing with the reactive model (0.3786 min), exponential smoothing (0.3739 min) gives 12% improvement. In the case of adaptive MI, exponential smoothing (0.3712 min) gives 8% improvement over reactive model (0.3743 min). Comparing with weighted moving average (0.3727 min), it gives merely 3% improvement. Exponential smoothing impact due to adaptive MI (merely 17 times algorithm invocation) is around 7%. This is all because of its forecasting characteristics best suited for the slowly varying workload nature. Surprisingly, reactive model works equivalent to weighted moving average in some cases (e.g., for MI = 1, 2). This is all because of the workload characteristics.

For highly varying workload, apparently WMA once again works much better. Especially with adaptive MI (only 20 times of algorithm invocation), it gives 7% improvement over the reactive model. Comparing with exponential smoothing, it gives 5%. This is because of its forecasting characteristics and suitability for the workload traces. Table 2 shows all the algorithms work quite nicely and withstand the SLA with adaptive MI over fixed MI.

CONCLUSION

This study proposed a novel system using meta-heuristic combinatorial search techniques that automatically regulates the VMM CPU scheduler for cloud, considering diverse geo-specific business requirements. It evaluated the usefulness of the framework for proactive, reactive and adaptive VCPU scheduling algorithms for varying real-world and synthetic workload traces. 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. In addition, the results of the experiments have shown that the Weighed Moving Average algorithm combined with the adaptive MI policy significantly outperforms other dynamic VM consolidation algorithms in all cases.

Regarding possible future works, first, the proposed system can be evaluated in a real Cloud infrastructure like Open Stack. Second, the system can be evaluated for more complex workload models such as slowly varying, 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, the work can be extended to cover the allocations of the resources like main memory and, network I/O. Fourth, multiple alternative forecasting methods in parallel can be applied. Finally, the model can be extended to support adaptive scheduling techniques in addition with the resource allocation for the diverse workloads.

REFERENCES

Armbrust, M., A. Fox, R. Griffith, A.D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica and M. Zaharia, 2010. A view of cloud computing. Commun. ACM, 53(1): 50-58.

Cherkasova, L., D. Gupta and A. Vahdat, 2007. Comparison of the three CPU schedulers in Xen. SIGMETRICS Perform. Eval. Rev., 35(2): 42-51.

Chia-Ying, T. and L. Kang-Yuan, 2013. A modified priority based CPU scheduling scheme for virtualized environment. Int. J. Hybrid Inform. Technol., 6(2).

Chisnall, D., 2007. The Definitive Guide to the Xen Hypervisor. 1st Edn., Prentice Hall Press, Upper Saddle River, NJ, USA.

Cillendo, E. and T. Kunimasa, 2007. Linux Performance and Tuning Guidelines. Redpaper, IBM.

Herbst, N.R., N. Huber, S. Kounev and E. Amrehn, 2013. Self-adaptive workload classification and forecasting for proactive resource provisioning. Proceeding of the 4th ACM/SPEC International Conference on Performance Engineering (ICPE'13). Czech Republic, Prague.

Huber, N., B. Fabian and K. Samuel, 2011a. Model-based self-adaptive resource allocation in virtualized environments. Proceeding of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS '11), pp: 90-99.

Huber, N., M. Quast, M.V. Hauck and S. Kounev, 2011b. Evaluating and modeling virtualization performance overhead for cloud environments. Proceeding of the International Conference on Cloud Computing and Services Science (CLOSER 2011). SciTePress, Noordwijkerhout, Netherlands, pp: 563-573.

ITA, 1998. The Internet Traces Archives: WorldCup98. Retrieved from: http://ita.ee.lbl.Gov/html/contrib/WorldCup.html.

Jung, G., M. Hiltunen, K. Joshi, R. Schlichting and C. Pu, 2010. Mistral: Dynamically managing power, performance and adaptation cost in cloud infrastructures. Proceeding of IEEE 30th International Conference on Distributed Computing Systems (ICDCS, 2010), pp: 62-73.

Kousiouris, G., T. Cucinotta and T. Varvarigou, 2011. The effects of scheduling, workload type and consolidation scenarios on virtual machine performance and their prediction through optimized artificial neural networks. J. Syst. Software, 84(8): 1270-1291.

Lim, S.H., J.S. Huh, Y.J. Kim, G.M. Shipman and C.R. Das, 2010. A quantitative analysis of performance of shared service systems with multiple resource contention. Technical Report. Retrieved from: http://www.cse.psu.edu/research/publications/tech-reports/2010/cse-10-010.pdf.

Liu, Z., W. Qu, W. Liu, Z. Li and Y. Xu, 2014. Resource preprocessing and optimal task scheduling in cloud computing environments. Concurr. Comp-Pract. E., DOI: 10.1002/cpe.3204.

Lu, L., H. Zhang, G. Jiang, H. Chen, K. Yoshihira and E. Smirni, 2011. Untangling mixed information to calibrate resource utilization in virtual machines. Proceeding of the 8th ACM International Conference on Autonomic Computing, pp: 151-160.

NLANR, 1995. National Laboratory for Applied Network Research. Anonymized access logs. Retrieved from: ftp://ftp.ircache.net/Traces/.

Rao, J., Y. Wei, J. Gong and C.Z. Xu, 2013. QoS guarantees and service differentiation for dynamic cloud applications. IEEE T. Network Serv. Manage, 10(1).

Schwetman, H., 2001. CSIM19: A powerful tool for building system models. Proceeding of the 2001 Winter Simulation Conference, pp: 250-255.

Schwiegeishohn, U. and R. Yahyapour, 1998. Improving first-come-first-serve job scheduling by gang scheduling. In: Feitelson, D.G. and L. Rudolph (Eds.), JSSPP'98. LNCS 1459, Springer-verlag, Berlin, Heidelberg, pp: 180-198.

Sethi, S., A. Sahu and S.K. Jena, 2012. Efficient load balancing in cloud computing using fuzzy logic. IOSR J. Eng., 2(7): 65-71.

Sivanandam, S.N. and S.N. Deepa, 2008. Introduction to Genetic Algorithms. 2nd Edn., Springer-Verlag, New York.

Smith, J.E. and R. Nair, 2005. Virtual Machines: Versatile Platforms for Systems and Processes. Morgan Kaufmann [Imprint], San Diego.

Sukwong, O. and H.S. Kim, 2011. Is co-scheduling too expensive for SMP VMs? Proceeding of the 6th conference on Computer Systems (EuroSys '11), pp: 257-272.

Suresh, S. and M. Kannan, 2014a. A study on system virtualization techniques. Proceeding of the International Conference on HI-TECh Trends in Emerging Computational Technology (ICECT, 2014). Virudhunagar, Tamilnadu, India.

Suresh, S. and M. Kannan, 2014b. A performance study of hardware impact on full virtualization for server consolidation in cloud environment. J. Theor. Appl. Inform. Technol., 60(3).

Watson, B.J., M. Marwah, D. Gmach, Y. Chen, M. Arlitt and Z. Wang, 2010. Probabilistic performance modeling of virtualized resource allocation. Proceeding of the 7th International Conference on Autonomic Computing (ICAC'10), pp: 99-108.

Weng, C., Z. Wang, M. Li and X. Lu, 2009. The hybrid scheduling framework for virtual machine systems. Proceeding of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments. New York, USA.