iTune: Engineering the Performance of Xen Hypervisor via Autonomous and Dynamic Scheduler Reconfiguration *

Faruk Caglar, Shashank Shekhar and Aniruddha Gokhale
Department of Electrical Engineering and Computer Science
Vanderbilt University, Nashville, TN 37235, USA
{faruk.caglar, shashank.shekhar, a.gokhale}@vanderbilt.edu

ABSTRACT

Despite the widespread use of server virtualization technologies in cloud data centers, such as those provided by the Xen hypervisor, system administrators experience multiple challenges in configuring the hypervisor's scheduler parameters to optimize its performance for applications running in the hypervisor-managed virtual machines. Manually tuning the scheduler's parameters based on generally accepted rules or through trial-and-error approach is a common practice. However, this approach is not effective and efficient particularly when dealing with dynamically changing workload on the host machines, varying CPU resource utilizations, and considering resource overbooking ratios for resource management. To address these problems, particularly in the context of the Xen hypervisor, this paper presents iTune: Engineering the Performance of Xen Hypervisor via Intelligent and Autonomous Xen Scheduler Configurations, which optimizes the Xen's scheduler configuration parameters autonomously through a three phase design comprising: (1) Discoverer, which monitors and saves the resource usage history of the host machines and groups set of related host machine workload, (2) Optimizer, where optimum Xen scheduler configuration parameters for each workload cluster is explored by employing a simulated annealing machine learning algorithm, and (3) Observer, where iTune monitors the resource usage of host machines online, classifies them into one of the categories found in the Discoverer phase, and loads the optimum scheduler parameters determined in the Optimizer phase. Experimental results evaluating iTune shows that it helps to improve the application performance compared to the Xen scheduler's default configuration settings.

Categories and Subject Descriptors

H.4.m [**Information Systems**]: INFORMATION SYSTEMS APPLICATIONS—*Miscellaneous*

General Terms

Experimentation

Keywords

Xen credit scheduler, run-time parameter tuning, cloud computing, optimization

*This work was supported in part by the National Science Foundation CAREER CNS 0845789 and AFOSR DDDAS FA9550-13-1-0227. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of NSF and AFOSR.

1. INTRODUCTION

Server virtualization is an important technology that makes it feasible to support the notion of Cloud computing, where multiple virtual machines (VMs) can be hosted on a single physical server. Virtualization enables Cloud Service Providers (CSPs) to increase the server utilization, and reduce energy consumption, hardware, and maintenance costs in their cloud data centers. Moreover, CSPs achieve additional resource utilization with server consolidation [9, 15] and resource overbooking [4, 20, 13] by packing more VMs on to the host machines.

Virtualization systems usually comprise a scheduling mechanism to share the physical CPU resources between the VMs running on the same host machine. VMs cannot directly access the physical resources; rather a virtual CPU (vCPU) of a VM can only access one of the physical CPU (pCPU) cores when it is scheduled from the run queue of the scheduler based on the enforced scheduling policy. Effective scheduling policies are crucial for effective and efficient scheduling of the physical server resources, which ultimately dictates the application performance running in a VM.

The resulting performance of an application running in a VM is directly impacted by the chosen scheduler configuration [10, 14]. If the scheduler does not operate efficiently and effectively, it yields to: (1) vCPUs not being able to access pCPUs when they need to, (2) a pCPU that is assigned to a vCPU being preempted before it completes its task, or (3) improper and numerous context switches. These in turn lead to performance degradation of the applications running in the VMs. Therefore, it is important to choose the optimal scheduling configurations when dealing with dynamically changing workloads on the host machines, varying CPU utilizations, and resource overbooking ratios adopted by CSPs to overbook the physical resources.

Our literature survey suggests that whenever performance

issues arise, it is common practice among administrators to manually tune the scheduler parameters and adopt a trial-and-error approach. Unfortunately, such approaches tend to address the performance issues under the unrealistic assumption that the overall system dynamics will not change over time thereby resulting in point solutions that yield only a temporary remedy and may not resolve the actual issue. The changing dynamics of workloads on the host machines and resource utilizations preclude any offline decision making of scheduler configuration parameters and manual tuning. These considerations call for an online, autonomous, and self-tuning system for scheduler .

To address these problems, in this paper we propose iTune, which is an intelligent and autonomous self-tuning middleware to optimize the scheduler parameters of the virtualization mechanism. Specifically, we focus on Xen [3], which is a widely used virtualization technology adopted by several prominent CSPs. Xen's hypervisor, which is the control program that manages guest VMs, allows multiple VMs to execute on the same host machine using the most appropriate virtualization mechanism that is available on the given hardware and host operating system. The default scheduler in the Xen hypervisor is a credit-based CPU scheduler, which promotes fair share scheduling among the VMs managed by the hypervisor. The Xen scheduler supports various configuration parameters, such as weight, CPU cap and scheduling timeslice for each VM. These are the parameters that affect how, when, and how long a VM gains access to shared physical resources.

iTune tunes Xen's credit scheduler parameters by dealing with changing workload on the host machine and employing machine learning techniques. iTune comprises a three phase architecture: (1) Discoverer, (2) Optimizer, and (3) Observer. In the Discoverer phase, iTune is trained to cluster host machines based on the workload where workload clusters are determined. The Optimizer phase deals with finding the optimum scheduler parameters. Finally, in the Observer phase, host machines are dynamically profiled and classified into one of the pre-determined categories based on which the credit scheduler parameters are loaded autonomously.

In prior work [21], we have demonstrated a similar approach, however, to automatically tune the configuration parameters and hence the performance of the Hadoop framework executing inside VMs, while in this work we focus on the performance issues that appear in the mechanisms that schedule the VMs themselves. Moreover, although the approach has similarities, the systemic issues related to performance at the Xen hypervisor-level are entirely different from those that exist at the level of the Hadoop framework.

This paper makes the following research contributions:

- We present an intelligent autonomous and self-tuning middleware called iTune to optimize Xen scheduler configuration (See Section 3.3).
- We provide deeper insights into how the Xen's internal scheduler parameters and performance are correlated with each other (See Section 4).
- We show how by employing machine learning algorithms iTune can find the optimum configuration parameters for Xen credit scheduler and self tune it based on varying workload at run-time (See Section 4).

The rest of this paper is organized as follows: Section 2 deals with relevant related work comparing it with our contributions; Section 3 presents the 3-phase systems architecture of iTune in detail; Section 4 validates the effectiveness of iTune; and finally Section 5 presents concluding remarks alluding to future work.

2. RELATED WORK

This section describe relevant related work that optimizes the Xen hypervisor performance, and compares it with iTune.

vSlicer [23] defines two types of virtual machines: CPU-intensive as non-latency sensitive virtual machine (NLSVM) and I/O-intensive as latency sensitive virtual machine (LSVM), and attempts to be fair to both by providing equal total time slot duration with LSVM getting shorter cycles and NLSVM getting longer cycles. However, the approach applies a one-size-fits-all technique for assigning the scheduling parameters. In contrast, we modify the scheduling parameters dynamically by profiling the virtual machines and making the decisions accordingly.

Xu et al. [25] address three sources of latency overhead in data centers: VM scheduling delay, host network queuing delay and switch queuing delay by applying an enhanced *shortest remaining time first* policy. For the VM scheduling delay, they overcome the limitation of the Xen credit scheduler's BOOST mechanism for latency-bound VMs. In contrast, iTune dynamically applies optimized Xen credit scheduler parameters to provide enhanced performance.

Zeng et al. [26] propose some improvements for the first version of Xen's credit scheduler for I/O bound latency-sensitive applications. Three improvements are presented: (1) *load balancing of BOOST domains*, (2) *prevention of premature preemption*, and (3) *dynamic time slice.* Some of the issues such as premature preemption presented were addressed in the second version of the Xen credit scheduler, and may be prevented through `rate_limit` and the new *run queue* logic. iTune shares some similarities with this work such as dynamic time slicing, however, the optimization technique employed by iTune considers all possible solutions whereas the prior work considers only two timeslice values. iTune also optimizes those scheduler parameters that will help all different types of applications including the latency-sensitive ones running on physical machines.

Xentune [14] propose a monitoring tool for the Xen virtual machine monitor (i.e., the hypervisor in this case). Xentune allows monitoring the effects of Xen's credit scheduler parameters on the performance of multimedia applications. Even though this work is close enough to what iTune is trying to accomplish, it differs in that iTune autonomously tunes the scheduler parameters based on VM classification.

RT-Xen 2.0 [22] is a real-time scheduling framework for multicore servers in the Xen virtual machine manager. RT-Xen

2.0 provides two schedulers: (1) global and (2) partitioned. Both schedulers support dynamic and static priorities. RT-Xen itself is a scheduler and targets only the CPU-intensive applications whereas iTune strives to optimize Xen's credit scheduler for different types of applications and does not consider strict real-time requirements in its current form.

Xu et al. [24] illustrate analysis results of different server consolidations with different scheduler configurations. In this work, the effects of timeslice and ratelimit, which were the new parameters proposed in Xen 4.2, were not presented since their work applied to an older version of Xen, specifically version 3.0. There is no dynamic configuration of scheduler based on the changing workload on the host machine whereas iTune utilizes these new parameters and also endeavors to solve the dynamic (re)configuration issue.

Pellegrini et al. [17] target performance optimization for MPI programs running on a multi-core SMP machine by selecting four parameters that have the most impact on the performance. Optimal settings of runtime parameters are predicted by the machine learning algorithms trained offline. To that end, this prior work and iTune are similar in the context of optimizing parameters but operate in different domains.

Cherkasova et al. [8] present comparision of three different CPU schedulers in Xen presenting an analysis of how each CPU scheduler and its parameters affects application performance. The authors limited their work only to the web server with 10KB files. This work helped us to gain insights from Xen's earlier schedulers to design and develop iTune.

3. ITUNE SYSTEM ARCHITECTURE AND DESIGN

This section describes the design and implementation details of iTune. To better understand our design, we first provide an overview of Xen and its credit scheduler[1] along with its configuration options. We then present an intuition behind the iTune approach. Finally we provide details of its design and implementation.

3.1 Overview of Xen and its Credit Scheduler

In this paper we assume a virtualized cloud data center where physical servers employ server virtualization mechanisms. Specifically, in this paper we utilize the Xen hypervisor [3] to virtualize the physical server resources and manage the virtual machines that host applications. The Xen project supports a Type-1 (or bare metal) hypervisor that manages the virtual machines. It is operating system-agnostic, and supports both fully virtualized or hardware assisted (called HVM) and paravirtualized (called PV) guests. The hypervisor is a small layer of software that is executed after a system's bootloader completes its operation. It contains the functionality to manage the CPU, memory and interrupts on the system.

The Xen hypervisor architecture supports the concept of domains, where the domain number zero called Dom0 is a special domain that contains the drivers for the underlying hardware and the necessary toolstack to control the lifecycle of the VMs. A new unprivileged domain (called DomU) is created to instantiate and host a new VM for the guest.

The Xen ecosystem comprises a number of tools and capabilities. For this paper, we leverage *XenMon* [10], which is a tool to monitor the performance of the various domains managed by Xen, and *libvirt,* which is a common, portable, and secure API to manage the lifecycle of domains maintained by the Xen hypervisor.

Xen's hypervisor schedules the CPU resource among the contending VMs hosted in their individual domains (including Dom0) using a *credit scheduler,* which is a proportional fair share and work conserving scheduler built to operate on symmetric multiprocessors. The credit scheduler has recently been made the default scheduler for Xen. It supports a number of configuration parameters whose values can be tweaked to tune the performance and behavior of Xen scheduler and consequently the performance delivered to the VMs. The following are the tunable parameters of Xen's credit scheduler.

- **Weight:** The weight parameter indicates the relative CPU allocation for a domain, which in turn can be translated into credit for each vCPU. The default value assigned to each domain is 256 and the range of values supported are between 1 and 65,535. A weight of 512 implies that the scheduler will allocate twice as much CPU to the domain with a weight of 512 compared to the domain that uses the default weight of 256. The weight parameter is particularly useful when the physical resource is overbooked, and is used as a measure of credit for the domain.

- **Cap:** The cap parameter indicates the maximum amount of a physical CPU (pCPU) that a domain will be able to consume even if other pCPUs are in the idle mode. The default value for cap is 0, which means that there is no cap or upper limit implying that a virtual CPU (vCPU) can access and consume CPU time of other pCPUs as long as they are in the idle mode. This parameter is expressed as a percentage and takes values between 0 to 1,200. For instance, a value of 400 implies the domain can consume up to 4 pCPUs.

- **Rate Limit:** The rate limit parameter specified in milliseconds indicates the minimum amount of CPU time that a VM is allowed to consume before being preempted. The default value is 1,000 and could take values between 100 to 500,000. This parameter allows a VM on a host machine to continue its work without being interrupted until the configured rate limit value is reached.

- **Timeslice:** The timeslice value is the scheduling interval of the credit scheduler specified in milliseconds. It indicates the interval over which the credit of each domain is recomputed. The default value is 30 msec while its range is between 1 and 1,000 msec. The value chosen for timeslice must be higher than the rate limit. The timeslice expiry is one of the events after which the scheduler makes the next scheduling decision and recompute the credits for the domains.

[1]Specifically we consider Xen's credit2 scheduler.

3.2 Intuition Behind the iTune Approach

A resource scheduler, such as the Xen credit scheduler, is a critical component of systems software that manages the resources on cloud platforms. Its design and how it manages the resources dictate the performance delivered to applications hosted in the VMs in individual Xen domains. The scheduler's resource management behavior depends on how it is configured in terms of its parameters, which is the responsibility of the cloud operator managing the platform. The operator is responsible for selecting the right values for the parameters to suit the expected loads on the cloud platform.

This is a hard problem to address because the number of configuration parameters and their available ranges give rise to a total of $65535 \times 1200 \times 499900 \times 1000 = 3.9 \times 10^{16}$ different configuration settings. Relying on the default values of each parameter may not always work well for every application type and workload on a host machine. While a rate limit value less than 1,000 msec could work well for latency-sensitive applications, it might not work well for CPU-intensive applications. Thus, application developers interested in deploying their applications in the virtualized cloud platforms must determine the best configuration settings for their applications. Moreover, they need to determine how these parameters must be changed at runtime as the system dynamics change due to workload and resource availability changes. Addressing these challenges in an automated way so that the operator is relieved of these responsibilities is the focus of this paper.

The key insight behind this work is as follows. When a number of entities compete for a common resource, that particular resource must be scheduled among these competing entities (Dom0 and DomUs in our case). Since the Xen credit scheduler is of the preemptive kind that works on the notion of credit-based proportional fair share, every competing domain for the pCPUs must experience some waiting time in the scheduler's *run queue*. waiting to be scheduled and getting preempted after various kinds of events occur (e.g., timeslice expiry, I/O blocking, etc). The amount of time attributed to waiting in the run queue has a direct impact on the response time, i.e., performance, experienced by the applications in the VMs of the domains. In Section 4.3 we show this correlation in the context of a Xen-based system.

In this work we are not concerned with improving the performance of a single application or an application type; rather we focus on improving the overall system performance, which includes all applications hosted in their respective VMs in the DomUs. Consequently, a key objective for us was to minimize the overall waiting time of the system, which will improve performance. Since we do not intend to replace or redesign the Xen scheduler, the only control variables that we could tweak were the existing scheduler's configuration parameters. Thus, we focus on minimizing the waiting time of the system by tuning the credit scheduler's configuration parameters.

Since workloads and utilizations in a cloud data center can vary over time, a one-size-fits-all set of configuration parameters is not going to suffice. Thus, there is a need to identify distinct *regions of operation* of the system, such that each region of operation varies from the other significantly from the other along one or more dimensions of performance characteristics, implying that the set of configuration parameters for each region will be distinct. Identifying the right number of distinct regions is an important challenge that needs resolution: too little a number will not make much difference to the delivered performance – in some cases even degrading it – when system dynamics change, while too many will complicate the system management and impose unwanted control overhead.

We solve this problem using a three phase approach described in Section 3.3. The first two phases are offline phases that use a combination of machine learning (specifically, k-means clustering and silhouette methods) and optimization (specifically, simulated annealing) to identify the number of regions of operation and the optimal scheduler configurations per region. The third phase is an online phase that periodically detects what region of operation the system currently is executing in, and dynamically updates the scheduler parameters based on the identified region of operation.

3.3 iTune System Architecture

We now present details of iTune. Figure 1 illustrates the system architecture of iTune, which is our intelligent, machine learning-based, autonomous, self-tuning middleware for Xen scheduler configuration. The system architecture of iTune enables a host machine to tune Xen's credit scheduler parameters online and autonomously. iTune is also applicable in situations involving varying workloads.

As can be seen from Figure 1, iTune is deployed in the privileged domain (Dom0) to observe the guest domains (DomUs), and monitor their behaviors. The resource usage information and internal scheduler metrics are collected through a modified XenMon and libvirt library, respectively. These information are stored in a MySQL database for workload clustering and optimum configuration searching. The Encog library [2], which is an open-source machine learning framework, was integrated within iTune to leverage algorithms, such as simulated annealing. To alter the Xen scheduler parameters, the XL toolstack of Xen is utilized. Matlab was also used for various purposes such as classification, correlation analysis, etc.

In our solution, we employ the machine learning algorithms to find the optimum configuration for the credit scheduler with respect to changing workloads. Based on our experimental results, we noticed that the default *cap* parameter consistently provided better results. As a consequence, we did not focus on finding the optimum value for this parameter (which helps reduce our search space). The following phases, which are described in Section 3.4, are followed to tune and load the optimum scheduler configuration.

- **Phase 1:** Resource usage information is logged by our monitoring module and k-means clustering algorithm is utilized to cluster the virtual machines by their resource utilizations.

- **Phase 2:** By running a simulated annealing algorithm, optimum configuration parameters are found for each cluster.

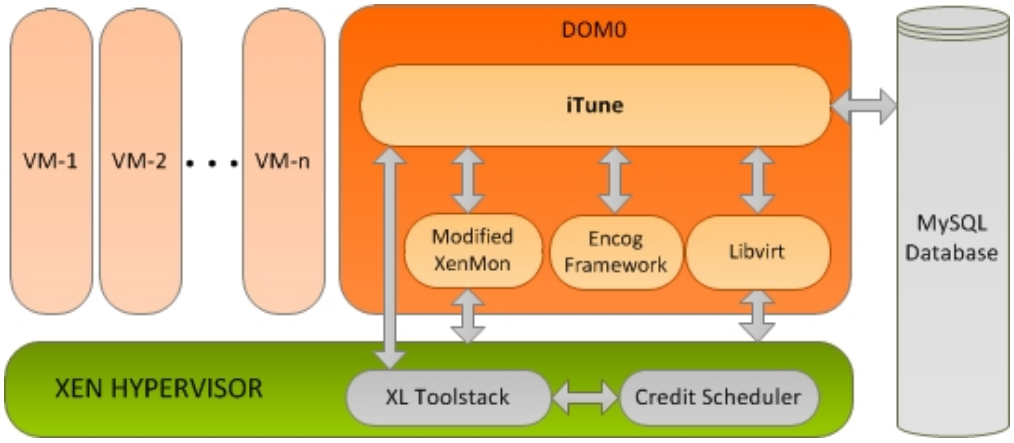


Figure 1: iTune System Architecture

• **Phase 3:** At run-time, the optimum configuration parameters corresponding to the workload on the host are loaded.

3.4 Three Phases of iTune

In this section, each phase of iTune is detailed out. Figure 2 depicts the three distinct phases of iTune which are encoded in the following components: (1) Discoverer, (2) Optimizer, and (3) Observer.

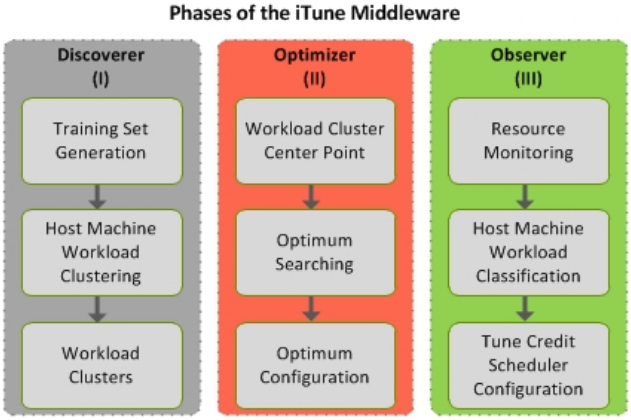


Figure 2: Three distinct phases of iTune

3.4.1 Phase 1: Discoverer Phase

The Discoverer phase is responsible for clustering host machines based on their workload. This phase comprises three steps described below:

Step 1.1: Training Set Generation – In this step, by running a synthetic workload generator we developed (see Section 4.2), the workload trace on a host machine is logged for use in our clustering step. The workload trace of host machine comprises a variety of resource information such as host name, CPU model, CPU frequency, number of cores, number of VMs, CPU utilization, CPU overbooking ratio, memory overbooking ratio, etc. The synthetic workload generator endeavors to mimic real world events and resource usage patterns of an actual data center server.

To that end we used real-world traces available in the Google cluster trace [18] and focused on a certain server (e.g., host id 2596362793 from the trace) and emulated the workload in our private data center for the training step. The Google data center cluster trace was collected during a period of 29 days in May 2011 and a document called *Google cluster-usage traces: format+schema,* which describes the semantics, format, and schema of the trace in detail [18]. This workload consists of substantial data for more than 12,000 heterogeneous physical host machines running 4,000 different types of applications and about 1.2 billion rows of resource-usage data.

Based on insights gained from the literature [10, 14], our prior work [21], and our analysis results, we decided on the following clustering idea: *workload on the host machines in the cloud can be classified into a set of distinct classes by using metrics, such as CPU utilization, CPU overbooking ratio, and VM count of a host machine.* Intuitively, these are the primary features that affect a hypervisor's scheduling performance. The more the requested CPU cores and deployed VMs on a host, the more is the scheduling latency due to the size of the scheduling run-queue. Critical insights on the effects of these features and how these features may effect the application performance are detailed in Section 4.

To define the performance model of the host machine, we considered the `waiting time` metric provided by XenMon (which is provided as a percentage). The waiting time metric indicates how much time a vCPU spends waiting to run when it needs to access a pCPU. We used the total waiting time percentage of the host machine as the performance indicator. For example, if the waiting time percentage reported is 30, it indicates that the VM waited 30% of the measurement interval of XenMon (which is a configurable parameter). The total waiting time percentage is the sum of waiting time percentages of each VM on the host. The higher the waiting time, the lower the application performance running on the VMs. The waiting time metric is thus crucial since it is directly related to the application performance.

In our performance model, we modified the existing XenMon implementation to log the sum of the metrics it provides. XenMon provides internal scheduler metrics for each VM.

We modified the XenMon.py Python script where we sum each of the metrics for all VMs (except Dom0) and save in a file. This way we can observe the waiting time percentage at host level and not only the VM level. XenMon is able to report a variety of information about the domains and hence the host machine, such as CPU usage, core number, waiting time, blocked time, execution count, etc. A snippet of XenMon log file containing these various reported metrics is illustrated in Figure 3.

blocked(tot)	blocked(%)	blocked/io	waited(tot)	waited(%)	waited/ex
198437150.2	19.844	0	2660.685	0	2359
198437150.2	19.844	0	2660.685	0	2359
8046212.808	0.805	0	697.756	0	746
8046212.808	0.805	0	697.756	0	746
8046212.808	0.805	0	697.756	0	746
8046212.808	0.805	0	697.756	0	746
284079086.6	28.408	0	6473.185	0.001	2277
339476429.7	33.948	0	4455.081	0	2572
339476429.7	33.948	0	4455.081	0	2572
339476429.7	33.948	0	4455.081	0	2572
155912878.5	15.591	0	3731.296	0	2275
155912878.5	15.591	0	3731.296	0	2275
205106558.2	20.511	0	5625.452	0.001	2609

Figure 3: A Snippet from XenMon Log File

The performance model of a host machine for a specified measurement time interval for XenMon can be described by Equation (1).

$$P = \sum_{i=1}^{n} WaitingTime_i \quad (1)$$

where

P : Host machine's performance

n : Total number of the VMs on the host machine

$WaitingTime$: Waiting time percentage of the VM

Step 1.2: Host Machine Workload Clustering – The goal of iTune is to provide an optimum configuration for Xen-enabled hosts in the data center. The configuration of the host machines in the data center is determined based on their cluster numbers that is found in this step. To achieve this goal, the resource usage information of host machines generated by our synthetic workload generator is analyzed and grouped into similar set of objects by utilizing machine learning techniques. This is also called as clustering or classification in machine learning terminology.

To cluster host machines into a set of classes, there exists a number of algorithms such as artificial neural networks, self-organizing map (SOM), and k-means in the literature. K-means [11] is a numerical, unsupervised, and iterative learning algorithm. It generates a fixed number of disjoint, non-hierarchical clusters where the members of the cluster are closer to their own cluster than any other cluster. However, the problem is NP-hard and heuristic algorithms are applied to find locally optimum solutions iteratively. The algorithm divides the data set into k clusters randomly and find centroid for each cluster. It then reassigns the points to the cluster of the nearest centroid. The process is repeated till no more points move, at which point the solution is said to be found. The algorithm is simple and computationally faster compared to other clustering techniques and provide tighter clusters. Thus, we chose to apply it in our approach.

In the k-means algorithm, one of the key requirement is to provide the number of clusters to the algorithm. However, providing a fixed number of clusters with no knowledge about the data would yield an inefficient solution. Hence, the Silhouette method [19] is utilized to determine the right number of clusters in the training set and measure the quality of the clusters. In this method, Silhouette coefficient, which is a measure of how well an observation fits into the assigned cluster, is calculated for different number of clusters. An average value of the coefficients for all the observations within a cluster gives the overall closeness of the points in the cluster to the centroid. A value of one indicates that it fits perfectly into the cluster and zero means it should belong to some other cluster. Silhouette plots are then used to depict the different values for different number of clusters.

In Table 1 the mean silhouette values for each cluster numbers are shown. We determined the optimum number of clusters by looking into these mean silhouette values. The higher the mean silhouette value is, the better the clustering quality. Based on Table 1, the best number of cluster which will be provided to the k-means is defined as **8** with maximum mean silhouette value of roughly 0.90.

Table 1: Silhouette Values of their Associated Clusters

Number of Cluster	Mean Silhouette Value
3	0.8797
4	0.8604
5	0.8723
6	0.8497
7	0.8788
8	**0.8974**
9	0.8859
10	0.7031

Step 1.3: Workload Clusters – In this step, the k-means algorithm is employed for the 8 clusters we determined in the previous step, and the center points found for each cluster are saved. The center points are used in the Observer phase to determine the corresponding cluster number at run-time. These points are rounded and illustrated in Table 2.

3.4.2 Phase 2: Optimizer Phase

In the Optimizer phase, iTune searches for the optimum configuration settings for each workload cluster by running the simulated annealing algorithm. The simulated annealing algorithm is one of the prominent machine learning technique for optimization problems. Genetic algorithms and hill climbing are also some of the techniques utilized to solve optimization problems by the research community. We decided to use simulated annealing because of its ability to not get stuck at a local minima and an assurance to find a statistically global optimum solution comparing to the other

Table 2: Rounded Center Points of Each Cluster

Cluster	CPU Utilization	CPU Overbooking Ratio	VM Count
1	7.95	1.58	5.22
2	179.64	3.60	9.12
3	164.24	3.48	9.66
4	50.18	1.08	10.57
5	203.91	4.04	10.69
6	225.21	4.29	11.15
7	268.48	5.06	13.82
8	84.73	1.91	16.81

optimization techniques. The Optimizer phase comprises three steps, which are: (Step 2.1) Workload Cluster Center Point, (Step 2.2) Optimum Searching, and (Step 2.3) Optimum Configuration.

Step 2.1: Workload Cluster Center Point – The center points found in Step 1.3 are utilized in this step. Recall that the center points consist of CPU utilization, CPU overbooking ratio, and VM count for each clusters. Each center point is representative of all the workload that belongs to that center point (and hence its cluster). Therefore, the optimized configuration for the cluster center point applies to all workloads in that cluster.

To find the optimum configuration for each center points, the workload on the host machine is accommodated to reflect each center point in the clusters found in Discoverer phase, i.e., the host is configured such that its CPU utilization, VM count and CPU overbooking ratio matches that of the center point. The accomodated workload is retained on the host machine until the simulated annealing algorithm finishes its task and finds the optimum configuration of this accomodated workload which is the representative of a cluster center point to find its optimum configuration.

Step 2.2: Optimum Searching – The simulated annealing algorithm is run to pinpoint the optimum solution for each cluster centroids found in Discoverer phase. As discussed in Section 3.1, the solution space for the Xen configuration parameters is very large, which makes it unrealistic to keep all possible parameter options in the solution space. Rather, we preferred to pick a range of values within the limits of each parameter (i.e. 100 different values within the range).

Recall that total waiting time percentage in Equation (1) is used as the performance indicator, where waiting time and the application performance are inversely proportional. A simulated annealing algorithm is proposed for finding the configuration with the minimum waiting time percentage. To that end, first, iTune monitors all the VMs on the host machine and collects the resource usage information. Then, it starts the XenMon to log the internal scheduler metrics for a period of four seconds. This period is long enough to collect the total waiting time of the host machine.

To discard the outliers in the data, the median of the waiting time is used instead of mean since median is more resistant to the outliers compared to the mean. Lastly, iTune executes the simulated annealing algorithm from the Encog framework to find the optimum configuration settings for the centroid being optimized. If the point found by the simulated annealing is not the optimum one, iTune continues to retrieve additional resource usage.

For each annealing process, iTune uses four seconds worth of data sampled every 100 ms time interval. The simulated annealing algorithm checks whether the median of the waiting time during this period is improving. This process continues until the configuration parameters are converged to optimum values.

Step 2.3: Optimum Configuration – Optimum configuration for each center point is ultimately found in Step 2.2 and saved into the iTune's configuration library to be used by the Observer phase. A total of eight configuration files are saved in the same folder of iTune, which correspond to the right clusters.

3.4.3 Phase 3: Observer Phase

The final phase of iTune is the Observer phase, which comprises three steps: (1) Resource Monitoring, (2) Host Machine Workload Classification, (3) Tune Credit Scheduler Configuration. The observer phase is employed online by iTune, which continuously monitors the resource usage of the host machine, classifies the host machine into one of the classes based on its profile, and loads the configuration file corresponding to its equivalent class.

Step 3.1: Resource Monitoring – In this step, iTune monitors the resource usage of the host machine along with the VMs on it. This step aims to profile a host machine. This profile data includes CPU utilization, CPU overbooking ratio, VM count, VM state, vCPU count of each VM, memory size, CPU model, etc. All of these metrics are retrieved through the libvirt library and saved into the database at user's choice.

As mentioned before, we collect multiple resource usage information of VMs and the host but only three of these are used in the Observer phase because we are interested in finding which nearest cluster number is closer to the actual workload of the host. These resource usage information are CPU utilization, CPU overbooking ratio, and VM count. iTune's default profiling interval is configured to run every 15 seconds. This parameter along with many other parameters such as the paths, XenMon arguments, simulated annealing start temperature, database connection string, etc. are all configurable and retained in the iTune's application configuration file.

Step 3.2: Host Machine Workload Classification – The methodology followed in this step is classifying the host machine into one of the clusters found in Discoverer phase. The classification is basically performed by computing the Euclidean distance from actual CPU utilization, CPU overbooking ratio, and VM count to each cluster center points found in the Discoverer phase. The classification decision of a host machine is based on the nearest center of a cluster point.

Step 3.3: Tune Credit Scheduler Configuration – After resource monitoring and classifying the host machine's workload, iTune loads the corresponding configuration settings of Xen credit scheduler from the configuration settings library. After loading the configuration file, iTune retains the cluster number of the host machine and does not load the configuration file as long as the cluster number remains same.

4. VALIDATING THE ITUNE APPROACH

This section presents empirical validation of the iTune approach in the context of a dynamically changing real-world data center workload that was obtained by emulating the Google cluster usage trace data [18] in our private data center. Our goal is to validate our hypothesis that the iTune approach of finding the right values for the configuration parameters for the Xen credit scheduler improves performance for VM-hosted applications compared to that due to the default configurations of the Xen scheduler. Recall that our iTune approach emphasizes minimizing the overall waiting time for VMs, and when combined with CPU utilization and resource overbooking ratio as the clustering parameters, we claim to improve performance for applications running in the VMs.

Thus, the validation of iTune requires a two-step process. First, there is a need to demonstrate the relationship between the scheduler waiting time to both application performance and average CPU and memory utilization of the VMs. If only these relationships hold, then the iTune approach can be used to dynamically tune the Xen scheduler parameters. Second, we must show that indeed iTune is able to improve performance for applications by dynamically tuning the Xen scheduler parameter while imposing negligible overhead in the control path.

To that end, this section is organized as follows:

- Description of the experimental setup and operating environment (Section 4.1).
- Description of the experiments conducted to emulate real-world workloads for the training phase of iTune (Section 4.2.
- Validating the relationship between scheduler waiting time and application performance and VM resource utilization (Section 4.3); and
- Comparing the performance improvements observed using the scheduling parameters suggested by iTune against the default Xen scheduler parameters, and overhead of iTune (Section 4.4.

4.1 Experimental Setup

The experiments were conducted in our private data center which is managed by the OpenNebula [16] cloud management software version 3.2.1. For the iTune approach we have assumed a homogeneous data center, where each host of the data center has the hardware and software configuration as described in Table 3.

The iTune engine runs inside Dom0 of the host machine and logs various parameters it needs for analysis and decision making. A python script executes on our cloud management server and invokes OpenNebula APIs to instantiate VMs with different configurations on the experimental host machine. The script also spawns the *lookbusy* [6] synthetic load generator processes on the VMs to generate the desired load.

Table 3: Hardware and Software Specification of the Experiment Host

Processor	2.1 GHz Opteron
Number of CPU cores	12
Memory	32 GB
Hard disk	512 GB
Operating System	Ubuntu 12.04 64-bit
Hypervisor	Xen 4.2.4
Guest virtualization mode	PV (i.e., para virtualized)
Guest Operating System	Ubuntu 11.10 64-bit

4.2 Generating the Training Set

Prior to presenting the actual results, we first describe how the training phase of iTune was conducted. For our experiments, the training data set was generated and utilized in the Discoverer phase of iTune (Step 1.1 of Section 3.4.1). To keep the data set as realistic as possible, we modeled the training session based on the resource and utilization data of one specific host with id 2596362793 chosen from the Google cluster trace logs. This host was chosen since it illustrated interesting variations in resource usage and overbooking ratios.

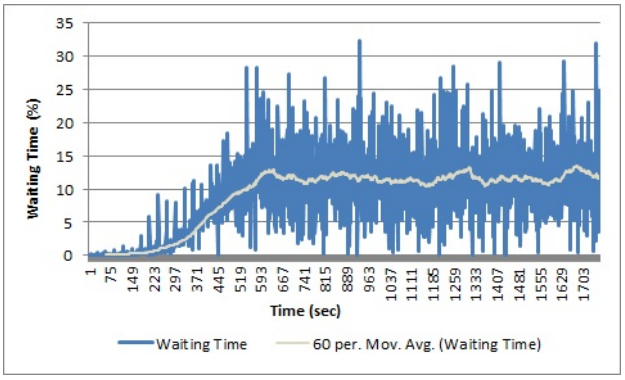
We picked a 12 hour time duration for data generation in which the number of VMs on that host varied from 1 to 18, CPU utilization varied from 0 to 325% and overbooking ratio varied from 0.08 to 6.5. As described in the experimental setup above, we used lookbusy to create these scenarios on the selected host (See Table 3) in our private data center. For this specific scenario, the optimum number of clusters identified by our training data was 8 as shown earlier in Table 1.

4.3 Impact of Run Queue Waiting Time on Performance and Resource Utilization

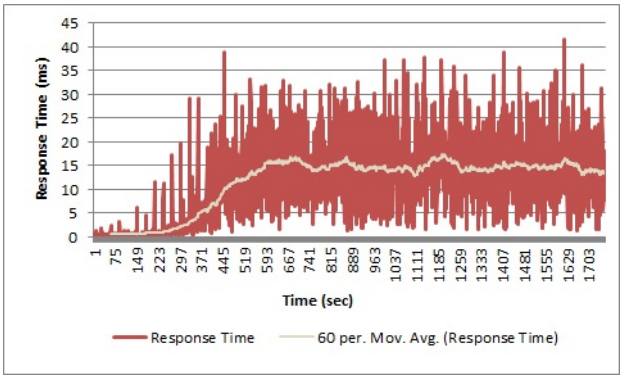
We now present an empirical analysis of how the scheduler waiting time impacts both application performance as well as VM-level resource utilization.

4.3.1 Relationship to Application Performance

The first set of experiments were conducted to identify the relationship between scheduler waiting time for VMs and application performance. For these experiments an overbooked scenario was considered with 24 VMs collocated with our target VM (i.e., the VM that we profiled) on the host machine we used in our study. A system which is overbooked tends to incur higher waiting time and hence this case was chosen. Each VM was allocated one vCPU and 512 MB memory. The workload was generated on each VM using lookbusy with five percent increment every minute in CPU usage from 0 to 100 percent. The five percent increment

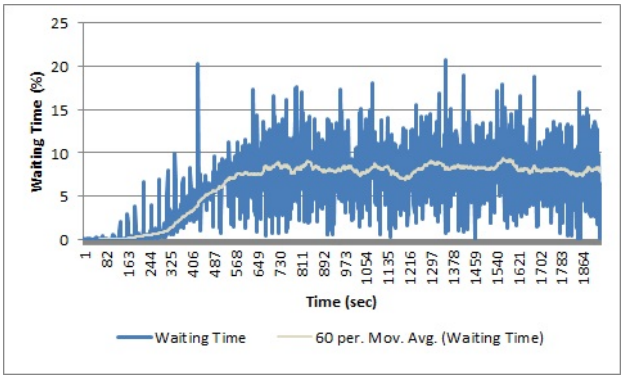


(a) Waiting Time

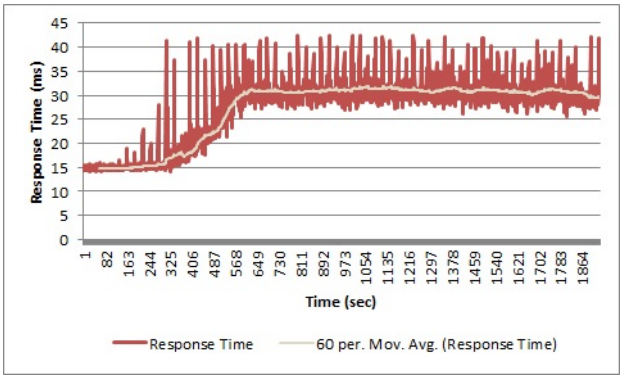


(b) Ping Response Time

Figure 4: Comparison of Ping Response Time and VM Waiting Time



(a) Apache Web Server Average Response Time



(b) Waiting Time

Figure 5: Comparison of Web Server Response Time and VM Waiting Time

was chosen to reduce the number of experiments we had to conduct; yet this value is good enough to capture changes in the system dynamics.

The common methodology for evaluating server performance is to measure the number of requests it serves per unit of time or the average amount of time spent in processing a request. Based on insights from the literature [23, 12], we chose *ping* as the micro benchmarking tool and one of the widely used web server, *Apache* as the macro benchmarking application for our experiments and used their response time as the indicator of the VM performance. We installed Apache web server 2.2.20 on the target VM and *ab - Apache HTTP server benchmarking tool* [1] on another test host residing on the same rack and network switch as the experimental host. *ab* is a popular and easy to use benchmarking tool that we used for measuring average Apache web server response time. The test host was also used as the ping client.

From the test host, ping requests were sent to the target VM at an interval of 100 ms and their response time was recorded for the duration of workload as explained earlier. Similarly, from `ab` tool on test host, client requests were generated for one second durations over 100 connections sending 10,000 requests for a small file of size of 2KB and the average response time was logged. This configuration let us measure the number of requests processed per second, and hence compute the average response time per second. The small size files were chosen as their transfer leads to CPU-bound workload generation [7], which is a good fit for our experiments. We then compared the response times for the two experiments against the waiting time as depicted in Figures 4 and 5.

The correlation values for the VM waiting time and ping response time is 0.46 and for Apache web server is 0.66. These values show that there is a reasonable degree of correlation between the two. Hence our hypothesis to minimize waiting time to improve VM performance is valid.

4.3.2 *Relationship to Resource Utilization*

The next set of experiments were performed to validate the relationship between scheduler imposed waiting time and VM resource utilization under different workload conditions. To that end we conducted five different tests. The tests numbered 1, 2, 3, 4 & 5 were conducted to analyze the relationship of CPU utilization, number of VMs and CPU overbooking ratio with waiting time for various scenarios. Of these, test 3 was conducted to ensure memory utilization does not play a significant role in determining the waiting time.

Test 1: Non-overbooked Case. The first experiment emulates a non-overbooked environment (CPU overbooking ratio 1). In this experiment, 12 VMs were created each having one CPU core and 512 MB memory on the 12 core host. The CPU utilization was incremented gradually from 0 to 100 percent as explained for the application performance experiments. The purpose of the experiment was to measure the waiting time in non-overbooked scenario and later use it to compare with overbooked scenario. We also wanted to verify the analogous relationship between CPU usage and waiting time.

Figure 6 shows the experimental results where the waiting time percentage is proportional to host machine's CPU utilization till $\tilde{9}5\%$. We observe that average waiting time is less than 5% for all the CPU utilization levels in this test. This shows that the impact of CPU utilization on waiting time under non-overbooked scenario is low. This also corresponds to the results of application performance experiments shown in Figures 4 and 5. We also noticed that waiting time starts dropping after $\tilde{9}5\%$ of CPU utilization. After analyzing the trace log, we observed that the average context switch between cores within the measurement interval before and after this utilization level is 48% and 22%, respectively. This means that the CPU pinning is performed by the credit scheduler at $\tilde{9}5\%$ CPU utilization level on a non-overbooked host and the results after this break-even point can be discarded from our consideration.

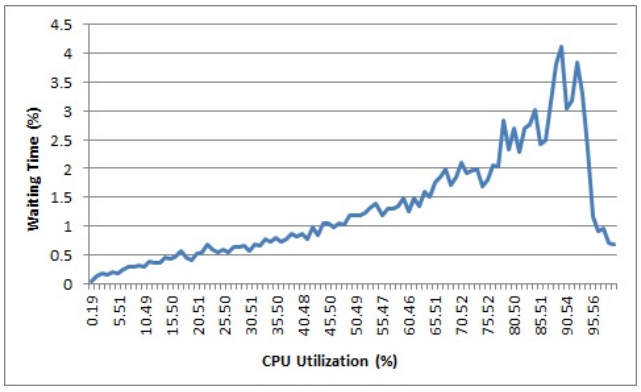


Figure 6: Waiting time vs CPU Utilization for Non-Overbooked Scenario

Test 2: Overbooked Case. The second experiment is similar to test **1** but with overbooking ratio of 2, i.e. 24 VMs each having a single vCPU were created on the 12 core host. Figure 7 shows the result where the waiting time is significantly higher than the non-overbooked test scenario and it exceeds 100% at close to 100% CPU utilization. The results are consistent with our premise that overbooking increases the scheduler waiting time thereby degrading the application performance. The waiting time also increases near linearly with CPU utilization depicting higher degree of correlation and substantiates the application of CPU utilization percentage as an input parameter to iTune.

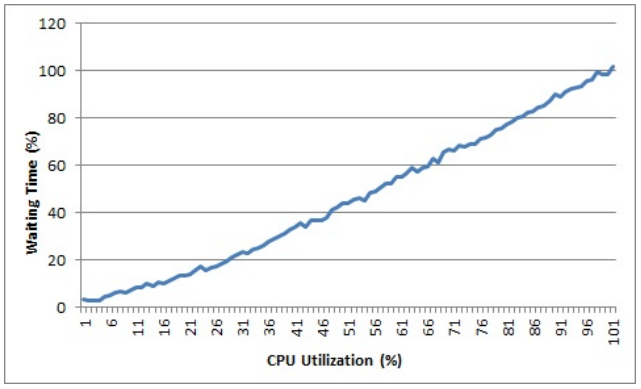


Figure 7: Waiting time vs CPU Utilization for Overbooked Scenario

Test 3: Memory Utilization Case. This test has the same configuration as test **2** but instead of incrementing CPU utilization, the lookbusy task was used to increment the memory utilization from 0 to 1,200 MB with step size of 60MB every minute. The experiment was performed to find the impact of memory utilization on waiting time. Figure 8 illustrates that the waiting time due to memory utilization is very low, less than 0.03% in this case, even though the trend shows an increase (though much slowly) with memory utilization. Based on these results, for all practical purposes, we do not use memory utilization as an input parameter in iTune.

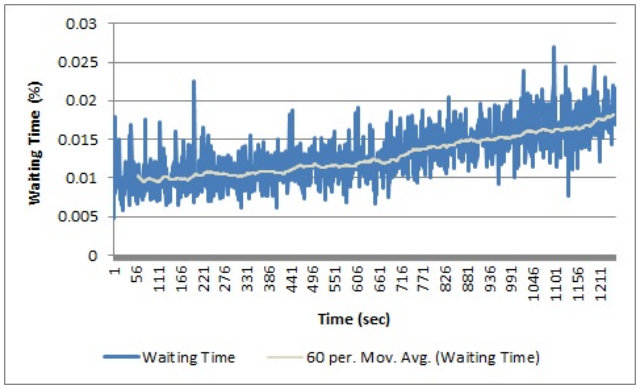


Figure 8: Waiting time vs Memory Utilization for Overbooked Scenario

Test 4 & 5: Heterogeneous VMs. These tests were conducted to verify that the trends of tests **1** and **2** hold even for host configurations having heterogeneous set of VMs and to show that the number of VMs on a host has high impact on waiting time. Test **4** had 6 VMs, two each with one, two and three vCPUs, respectively, for a total of 12 vCPUs similar to the non-overbooked test **1**. Test **5** had 12 VMs, four each with one, two and three vCPUs, respectively, for a total of 24 vCPUs as in overbooked test **2**.

Figure 9 represents the result for test **4** that has an analogous trend to test **1**. Similarly, Figure 10 shows the result for test **5**, comparable to trend of test **2**. However, the

waiting time is nearly ten times less for test **4** than test **1** having half the number of VMs in non-overbooked scenario and twice less for test **5** than test **2** in overbooked scenario. This supports our hypothesis of using number of VMs and CPU overbooking ratio as the input parameters to iTune.

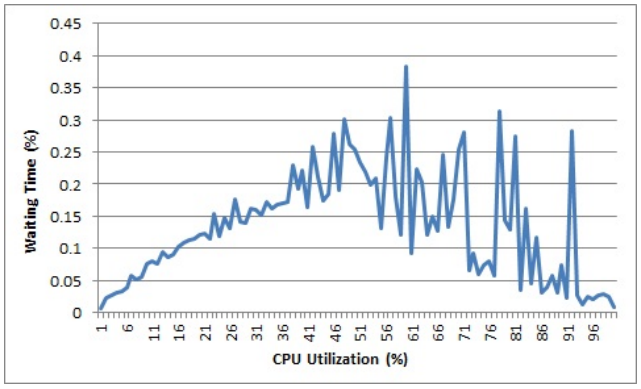


Figure 9: Waiting time vs CPU Utilization for Non-Overbooked Heterogeneous VMs Scenario

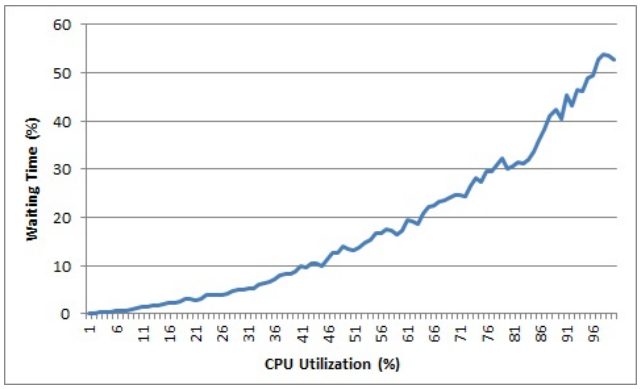


Figure 10: Waiting time vs CPU Utilization for Overbooked Heterogeneous VMs Scenario

4.4 Application Performance Improvement using iTune

The final set of experiments were conducted to validate the effectiveness of the iTune framework where we compare the performance improvement of applying our approach over the default one. In this setup, we created a random workload having 21 VMs, each using 45% CPU on a host machine in our private data center. The resource monitoring step of the Observer phase provided 2.25 CPU overbooking ratio and 94.15% host CPU usage. The next step of the iTune was host machine classification that resulted in classifying the host to one of the clusters. Subsequently, the corresponding Credit Scheduler configuration was loaded and results were obtained. The overhead of using iTune in the runtime phase is negligible.

We ran the experiments for one minute duration for both default and iTune optimized parameters and compared the results. Table 4 shows the results obtained for different validation parameters. We observed an overall waiting time improvement of 19.6% for the experimental host. We also

Table 4: Validation Results

Setup	Xen Default	iTune	Improvement (%)
Host Waiting Time	7.50 %	6.03 %	19.60
VM Waiting Time	1.89 %	1.26 %	33.33
VM Ping Response Time	1.17 ms	0.80 ms	31.62
VM Application Response Time	21.30 ms	17.56 ms	17.56

validated the performance gain for one of the representative virtual machine picked from the host. It showed an average waiting time improvement of 33.33% during the experiment, and ping and Apache web server response time improvement of 31.62% and 17.56%, respectively. Hence, the experiments validate the iTune approach for Xen Credit Scheduler autonomous configuration.

5. CONCLUSIONS

Systems software is often complex. One reason for its complexity stems from its high degree of flexibility that stems from the need to make it more widely applicable. The system flexibility often manifests in the form of configuration parameters can be used to tweak the system behavior. The Xen scheduler is an example of systems software, which is used to schedule CPU resources for the virtual machines it manages in a cloud data center. Such flexibility offered by systems software can become overwhelming for operators; without appropriate tool support, operators often have to resort to default values to get their system to work, which may not provide the best performance, or resort to trial-and-error-based approaches, which have no scientific basis.

To address such concerns, this paper presented iTune, which is an Intelligent and Autonomous Self-Tuning Middleware for Xen Scheduler Configuration. iTune comprises three phases named Discoverer, Optimizer, and Observer. The Discoverer phase is responsible for generating resource usage history of host machines and workload clustering. The optimum scheduler configuration parameters are searched in the Optimizer phase. The Observer is the final phase which monitors resource usage, classifies host machine workload at run-time, and loads the optimum scheduler parameters. iTune employs k-means and simulated annealing machine learning algorithms for host machine workload clustering and Xen's credit scheduler parameter optimization.

The following observations can be made about iTune:

- Although iTune has currently been demonstrated in the context of the Xen credit scheduler, the approach has broader applicability and can be used for other systems software.
- Although we have identified 8 as the number of regions of operation (i.e., clusters) for our training set, this number was derived solely based on a specific workload

pattern. It is possible that for a different kinds of expected workload, the number of identified clusters may be different. Hence we suggest that CSPs first apply iTune to their expected workloads.

- It is possible that the workload patterns themselves may differ during different times of the years, and hence it may be necessary to switch between one set of clusters to another. This dimension of work is part of our future work.
- Our recent work has explored the dimensions of resource overbooking to conserve server-side resources [4], and also power-performance trade-offs in data centers [5]. It is possible that the objectives of these efforts and iTune may conflict with each other. Our future work will explore trade-offs along these dimensions.

All scripts, source code, and experimental results of iTune are available for download from `www.dre.vanderbilt.edu/~caglarf/download/iTune`.

6. REFERENCES

[1] ab - apache http server benchmarking tool. `http://httpd.apache.org/docs/2.2/programs/ab.html`, 2014.

[2] Encog Machine Learning Framework. `http://www.heatonresearch.com/encog`, 2014.

[3] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., and Warfield, A. Xen and the Art of Virtualization. In *ACM SIGOPS Operating Systems Review* (2003), vol. 37, ACM, pp. 164–177.

[4] Caglar, F., and Gokhale, A. iOverbook: Managing Cloud-based Soft Real-time Applications in a Resource-Overbooked Data Center. In *The 7th IEEE International Conference on Cloud Computing (CLOUD' 14)* (Anchorage, AL, USA, June 2014), IEEE, p. 10.

[5] Caglar, F., Shekhar, S., and Gokhale, A. iPlace: An Intelligent and Tunable Power- and Performance-Aware Virtual Machine Placement Technique for Cloud-based Real-time Applications. In *17th IEEE Computer Society Symposium on Object/component/service-oriented real-time distributed Computing Technology (ISORC '14).* (Reno, NV, USA, June 2014), IEEE.

[6] Carraway, D. Lookbusy–a synthetic load generator. `http://www.devin.com/lookbusy/`, 2014.

[7] Cherkasova, L., and Gardner, R. Measuring cpu overhead for i/o processing in the xen virtual machine monitor. In *USENIX Annual Technical Conference, General Track* (2005), vol. 50.

[8] Cherkasova, L., Gupta, D., and Vahdat, A. Comparison of the three cpu schedulers in xen. *SIGMETRICS Performance Evaluation Review 35*, 2 (2007), 42–51.

[9] Corradi, A., Fanelli, M., and Foschini, L. Vm consolidation: a real case based on openstack cloud. *Future Generation Computer Systems 32* (2014), 118–127.

[10] Gupta, D., Gardner, R., and Cherkasova, L. Xenmon: Qos monitoring and performance profiling tool. *Hewlett-Packard Labs, Tech. Rep. HPL-2005-187* (2005).

[11] Hartigan, J. A., and Wong, M. A. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics) 28*, 1 (1979), 100–108.

[12] Heo, J., and Singaravelu, L. Deploying Extremely Latency-sensitive Applications in vSphere 5.5: Performance Study. Tech. rep., VMware, Inc., 2013. `www.vmware.com/files/pdf/techpaper/latency-sensitive-perf-vsphere55.pdf`.

[13] Householder, R., Arnold, S., and Green, R. On cloud-based oversubscription. *arXiv preprint arXiv:1402.4758* (2014).

[14] Lee, M., Krishnakumar, A., Krishnan, P., Singh, N., and Yajnik, S. Xentune: Detecting xen scheduling bottlenecks for media applications. In *Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE* (2010), IEEE, pp. 1–6.

[15] Lee, S., Panigrahy, R., Prabhakaran, V., Ramasubrahmanian, V., Talwar, K., Uyeda, L., and Wieder, U. Validating heuristics for virtual machines consolidation. *Microsoft Research, MSR-TR-2011-9* (2011).

[16] Milojičić, D., Llorente, I. M., and Montero, R. S. Opennebula: A cloud management tool. *IEEE Internet Computing 15*, 2 (2011), 0011–14.

[17] Pellegrini, S., Wang, J., Fahringer, T., and Moritsch, H. Optimizing mpi runtime parameter settings by using machine learning. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface*. Springer, 2009, pp. 196–206.

[18] Reiss, C., Wilkes, J., and Hellerstein, J. Google cluster-usage traces: format+ schema. *Google Inc., White Paper* (2011).

[19] Rousseeuw, P. J. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics 20* (1987), 53–65.

[20] Tomas, L., and Tordsson, J. An autonomic approach to risk-aware data center overbooking. *Cloud Computing, IEEE Transactions on PP*, 99 (2014), 1–1.

[21] Wu, D., and Gokhale, A. A Self-Tuning System based on Application Profiling and Performance Analysis for Optimizing Hadoop MapReduce Cluster Configuration. In *20th Annual IEEE International Conference on High Performance Computing (HiPC '13)* (Bengaluru, India, Dec. 2013), IEEE, pp. 89–98.

[22] Xi, S., Wilson, J., Lu, C., and Gill, C. RT-Xen: Towards Real-time Hypervisor Scheduling in Xen. In *Proceedings of the International Conference on Embedded Software (EMSOFT)* (2011), ACM, pp. 39–48.

[23] Xu, C., Gamage, S., Rao, P. N., Kangarlou, A., Kompella, R. R., and Xu, D. vslicer: latency-aware virtual machine scheduling via differentiated-frequency cpu slicing. In *Proceedings of the 21st international symposium on High-Performance Parallel and Distributed Computing* (2012), ACM, pp. 3–14.

[24] Xu, X., Shan, P., Wan, J., and Jiang, Y. Performance evaluation of the cpu scheduler in xen. In *Information Science and Engineering, 2008. ISISE'08. International Symposium on* (2008), vol. 2, IEEE, pp. 68–72.

[25] Xu, Y., Bailey, M., Noble, B., and Jahanian, F. Small is better: avoiding latency traps in virtualized data centers. In *Proceedings of the 4th annual Symposium on Cloud Computing* (2013), ACM, p. 7.

[26] Zeng, L., Wang, Y., Shi, W., and Feng, D. An improved xen credit scheduler for i/o latency-sensitive applications on multicores. In *Cloud Computing and Big Data (CloudCom-Asia), 2013 International Conference on* (Dec 2013), pp. 267–274.