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

Faruk Caglar, Shashank Shekhar, and Aniruddha Gokhale (Senior Member, IEEE)

Abstract—Despite the widespread use of server virtualization technologies in cloud data centers, system administrators experience multiple challenges in configuring the hypervisor's scheduler parameters to optimize its performance. Manually tuning the scheduler's parameters is a common practice, however, this approach is not effective particularly when dealing with dynamically changing workload and resource utilizations on the host machines. This problem becomes even harder if cloud resources are overbooked while hosting both latency-sensitive and batched applications. To address these issues, this paper presents iTune, which is a framework for engineering the performance of a hypervisor via intelligent and autonomous scheduler configurations. Concretely, iTune optimizes the Xen hypervisor's scheduler configuration parameters autonomously through a three phase process 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 are 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 validate our claims.

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

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 [1], [2] and resource overbooking [3], [4], [5] by packing more VMs on the host machines.

Virtualized systems often comprise a scheduling mechanism to share the physical CPU resources among 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 by 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 for the hypervisor [6], [7]. 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. Moreover, orchestrating scheduler configuration becomes even more chaotic when both latency-sensitive and batch applications are collocated on the same host machine. Therefore, it is important to choose the optimal scheduling configurations when dealing with dynamically changing workloads on the host machines, varying utilizations, and resource overbooking ratios adopted by CSPs to overbook the physical resources.

A survey of prior research 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 [8], [9], [10]. 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 the hypervisor 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

- *F. Caglar is with the Department of Computer Engineering, Meliksah University, Kayseri, TURKEY. Work conducted by first author while at Vanderbilt University. E-mail: fcaglar@meliksah.edu.tr*
- *S. Shekhar is with the Department of Electrical Engineering and Computer Science, Vanderbilt University, Nashville, TN, 37235. E-mail: shashank.shekhar@vanderbilt.edu*
- *A. Gokhale is with the Department of Electrical Engineering and Computer Science, Vanderbilt University, Nashville, TN, 37235. E-mail: a.gokhale@vanderbilt.edu*

Manuscript received May 5, 2015; revised September 17, 2015.

virtualization mechanism. Specifically, we focus on Xen [11], 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 a number of configuration parameters, such as weight, CPU cap, and scheduling time slice for each VM. These are the parameters that affect how, when, and how long a VM gains access to shared physical resources.

iTune configures the parameters of Xen's credit scheduler 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. Discoverer and Optimizer are offline phases whereas the Observer phase is online. 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. Furthermore, performance requirements of latency-sensitive applications are categorized and taken into account in the Optimizer phase when numerous application types are collocated on the same host machine in a multi-tenant cloud environment. 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 short, the ultimate goal of iTune is to autonomously optimize all of the scheduler parameters and dynamically reconfigure the scheduling system based on the latency sensitivity requirements of each VM (not individual applications in the VM) on the host machine as the workload changes dynamically.

In prior work [12], we have demonstrated a similar approach, however, that work automatically tuned the configuration parameters and hence the performance of the Hadoop framework executing inside VMs. In contrast, in this work we focus on the performance issues that appear in the mechanisms that schedule the VMs themselves. Thus, although the approach is similar in spirit, the problem domain is different which encounters a different set of challenges than our work in [12]. For example, 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 contributions:

- We provide key insights into how the Xen's internal scheduler parameters and performance are correlated with each other (See Section 3.3).
- We provide options to mark the VMs into one of the following categories: (1) latency-sensitive level-1 (LS-1), (2) latency-sensitive level-2 (LS-2), (3) latency-sensitive level-3 (LS-3), and (4) non-latency sensitive (NLS). iTune ensures delivering the performance requirements of applications in these categories (See Section 3.4).
- We present an intelligent, autonomous and self-tuning middleware called iTune to optimize the Xen scheduler configuration (See Section 3.5).
- We show how, by employing machine learning algorithms, iTune can find the optimum[1] 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 its limitations and future work.

2 Related Work

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

vSlicer [13] 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 CPU cycles and NLSVM getting longer CPU cycles. However, the approach applies a one-size-fits-all technique for assigning the scheduling parameters. In contrast, we provide finer granularity for latency-sensitive VMs and modify the scheduling parameters dynamically by profiling the virtual machines and making the decisions accordingly.

Xu et al. [14] 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 (i.e. one of the queues of earlier version of credit scheduler and has the highest priority) for latency-bound VMs. In contrast, iTune dynamically applies optimized Xen credit scheduler parameters to provide enhanced performance.

Zeng et al. [15] 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 were addressed in the latter version of the Xen credit scheduler, and may be prevented through `rate_limit` Xen configuration 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 different types of applications including the latency-sensitive ones running on physical machines.

Blagodurov et al. [16] propose a method to manage the collocated applications which were classified as critical and non-critical in the virtualized environment. An increase in

1. The optimum word refers to the optimal solution found by simulated annealing algorithm

server utilization and SLA is assured by prioritizing the access to CPU cycles for VMs through utilizing the linux control groups (cgroups) weights. Both static and dynamic weight assignment to the critical and non-critical applications were studied in the paper. The way that classifying applications into two different groups is similar to how iTune classifies the VMs. iTune classifies the critical applications (i.e. VMs) to even further criticality levels.

Padala et al. [17] propose AutoControl, which is a control theory-based, fine-grained resource control system. AutoControl automatically captures the relationship between application performance and the resource allocations. As the demand changes dynamically in virtualized environment, it scales the resource allocations of applications up/down to assure their SLOs. AutoControl does not consider resource-overbooked systems. Rather, AutoControl endeavors to satisfy that total requested resources of all applications are less than or equal to the node capacity. In contrast, iTune configures all the parameters of the scheduler whereas AutoControl only utilizes the cap parameter of the hypervisor's scheduler to throttle the resource allocations to applications. Additionally, iTune targets resource-overbooked environments where latency-sensitive and batch-oriented applications are all hosted together.

Xentune [7] proposes 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 comes close to what iTune is trying to accomplish, it differs in that iTune allows classifying VMs based on their latency sensitivity levels.

RT-Xen 2.0 [18] is a real-time scheduling framework for multicore servers using Xen. 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.

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 along with its configuration options. We then present an intuition behind the iTune approach that is based on the impact of run queue waiting time on performance and resource utilization. Finally we provide details of its design and implementation.

3.1 System Model and 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 [11] to virtualize the physical server resources and manage the virtual machines that host applications.

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* [6], which is a tool to monitor the performance of the 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 physical 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.
- **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.
- **Rate Limit:** The rate limit parameter specified in microseconds indicates the minimum amount of CPU time that a VM is allowed to consume before being preempted.
- **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.

3.2 Problem Statement

Resource schedulers, such as the Xen credit scheduler, are critical components of systems software that manage the resources on cloud platforms. Their design and how they manage the resources dictate the performance delivered to applications hosted in the platforms. Specifically, the Xen scheduler's resource management behavior depends on how it is configured in terms of its parameters, which is the responsibility of the cloud operator. 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 roughly $65535 \times 1200 \times 499900 \times 1000 = 3.9 \times 10^{16}$ different configuration settings for a 12 CPU host machine. 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 microseconds 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 system administrator is relieved of these responsibilities is the focus of this paper.

3.3 Intuition Behind the iTune Approach

The key insight behind our solution approach 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*. 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.

We now show this correlation in the context of a Xen-based system by presenting an empirical analysis of how the scheduler waiting time impacts both application performance as well as VM-level resource utilization.

3.3.1 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* [19] with five percent increment every minute in CPU usage from 0 to 100 percent. The five percent increment 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 [13], [20], we chose *ping* as the micro benchmarking tool and one of the widely used web server, *Apache HTTP server* as the macro benchmarking application for our experiments and used their response times 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* [21] 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, using `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 2KB and the response times were logged. This configuration allows us to 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 [22], which is a good fit for our experiments. We then compared the response times for the two experiments with their *waiting time*. Comparison between web server response time and VM waiting time is depicted in Figure 1.

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 strong degree of correlation between the two. Hence, our hypothesis to minimize waiting time to improve VM performance is valid.

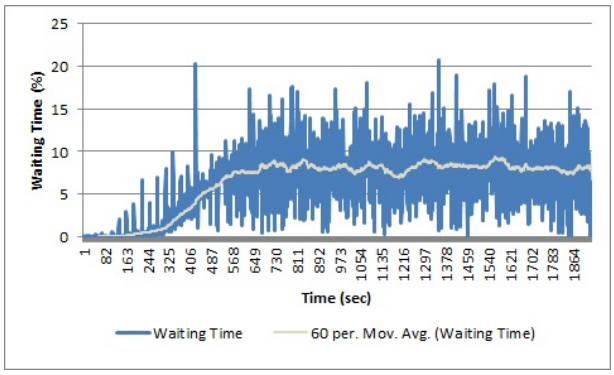
3.3.2 Relationship to Resource Utilization

The next set of experiments were performed to validate the relationship between the scheduler imposed waiting time and VM resource utilization under different workload conditions. To that end we conducted six different tests. The tests numbered 1, 2, 3, 4, 5 & 6 in the following sub sections were conducted to analyze the relationship of CPU utilization, network 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.

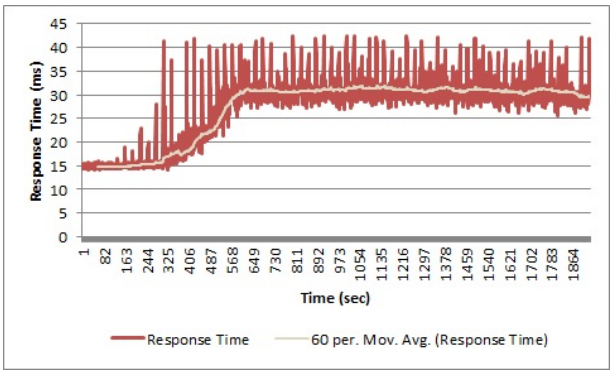
3.3.2.1 **Test 1: Non-overbooked Case**: The first experiment emulates a non-overbooked environment (CPU overbooking ratio 1). In this experiment, 12 VMs were created each having one CPU core and 512 MB memory on the 12 core host. The CPU utilization was incremented gradually from 0 to 100 percent as explained earlier 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 2 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 Figure 1. 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 among the CPU 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.

3.3.2.2 **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 3 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



(a) Waiting Time



(b) Apache Web Server Response Time

Fig. 1: Comparison of Web Server Response Time and VM Waiting Time Showing Similar Trend

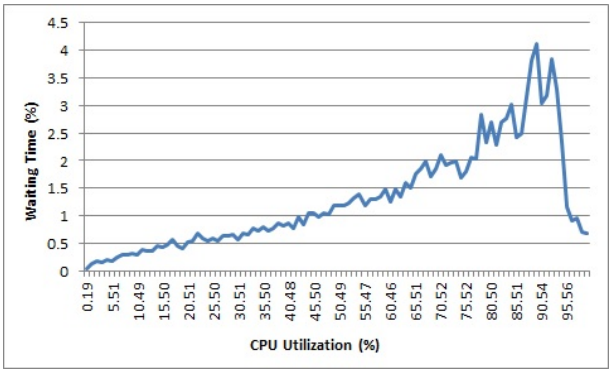


Fig. 2: Waiting time vs CPU Utilization for Non-Overbooked Scenario

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.

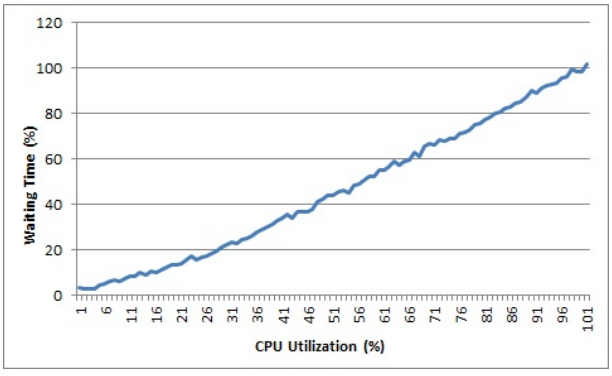


Fig. 3: Waiting time vs CPU Utilization for Overbooked Scenario

3.3.2.3 **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 of each VM from 0 to 1,200 MB with step size of 60MB every minute. Ballooning technique employed by the virtual machine managers allows to overbook physical memory by dynamically adjusting the portion used by a guest VM. Therefore, to be able to observe the impact of memory utilization realistically, the ballooning technique was enabled on the host machines. The experiment was performed to determine the impact of memory utilization on waiting time. Figure 4 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 as long as the memory utilization does not exceed the host capacity.

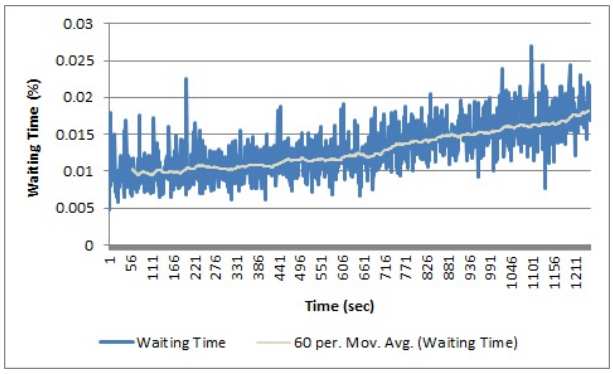


Fig. 4: Waiting time vs Memory Utilization for Overbooked Scenario

3.3.2.4 **Test 4: Network Utilization Case**: This test has the same configuration as tests **2** and **3** but instead of incrementing CPU utilization in test **2** or memory utilization as in test **3**, the *ping* tool was used to increment the network utilization of each VM from 17 KBps to 256 KBps with step size of about 5KBps increment every minute. The experiment was performed to determine the impact of network utilization on waiting time.

Figure 5 illustrates that the waiting time due to network utilization is very high, reaches up to 200% at the host level. Initially, waiting time was not that high and less than 25%. Since the system was overbooked and as the network IO usage of each VM was increased and VMs started to require more CPU time to handle network packets ultimately causing waiting time to increase dramatically after some point.

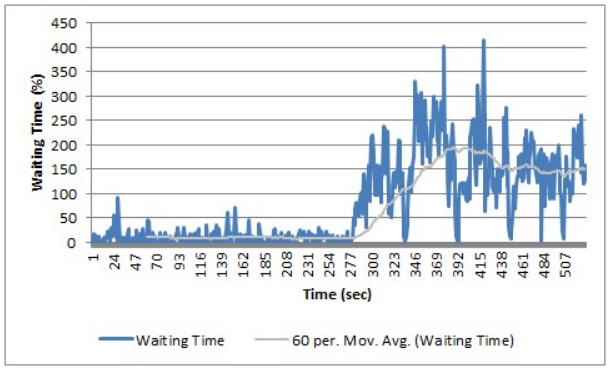


Fig. 5: Waiting time vs Network Utilization for Overbooked Scenario

Based on these results, for all practical purposes, we will use network utilization as an input parameter in iTune.

3.3.2.5 **Test 5 & 6: 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 **5** 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 **6** had 12 VMs, four each with one, two and three vCPUs, respectively, for a total of 24 vCPUs as in overbooked test **2**.

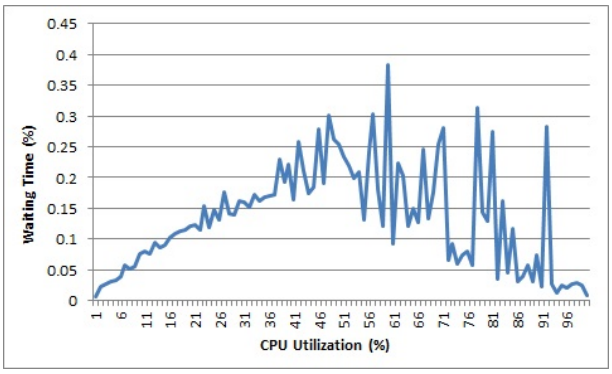


Fig. 6: Waiting time vs CPU Utilization for Non-Overbooked Heterogeneous VMs Scenario

Figure 6 represents the result for test **5** that has an analogous trend to test **1**. Similarly, Figure 7 shows the result for test **6**, comparable to trend of test **2**. However, the waiting time is nearly ten times less for test **5** than test **1** having half the number of VMs in non-overbooked scenario and twice less for test **6** 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.

The empirical insights and proof shown in this section guided us to utilize the *waiting time* metric for online tuning of scheduler parameters. Moreover, it became the main goal of this paper to *optimize overall waiting time of host machine* which will ultimately satisfy the QoS requirements of the applications running in the VMs.

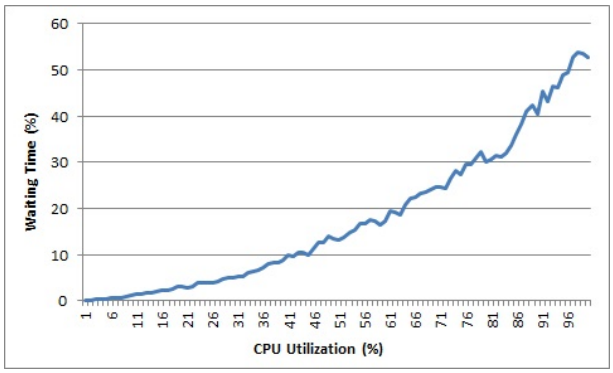


Fig. 7: Waiting time vs CPU Utilization for Overbooked Heterogeneous VMs Scenario

3.4 iTune Solution Approach

In this work, we are concerned with providing the performance assurance to applications running in the VMs marked as *LS-1*, *LS-2*, *LS-3*, and *NLS* which may be translated into *best*, *better*, *good*, and *best effort*, respectively. Furthermore, we also focus on improving the overall system performance compliant with these performance-level descriptors of their respective VMs. Consequently, a key objective for us is to assure the performance delivered to the VMs associated with their performance-level descriptor, and 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 nor is an offline one time setting of the configuration parameters. Thus, there is a need to identify distinct *regions of operation* of the system, such that each region of operation varies 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 3.5. The first two phases are *offline* phases that use a combination of machine learning (specifically, *k-means clustering* [23] and *silhouette* [24] 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.

3.5 iTune System Architecture and Implementation

We now present the details of iTune. Figure 8 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 8, iTune is deployed in the privileged domain (Dom0) to observe the guest domains (DomUs), and monitor their behavior. 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 [25], 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.

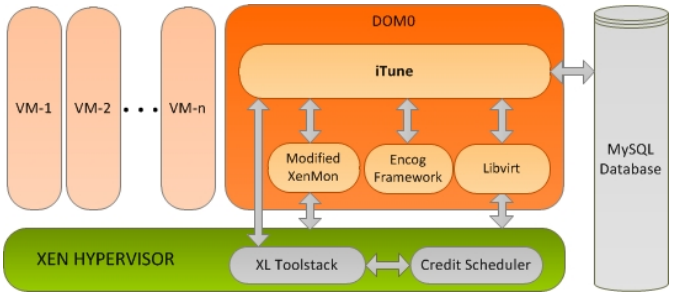


Fig. 8: iTune System Architecture

In our solution, we employ the machine learning algorithms to find the optimum configuration for the credit scheduler with respect to changing workloads. The following phases are followed to tune and load the optimum scheduler configuration.

- **Phase 1 (*Offline*):** 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 (*Offline*):** By running a simulated annealing algorithm, optimum configuration parameters are found for each cluster.
- **Phase 3 (*Online*):** At run-time, the optimum configuration parameters corresponding to the workload on the host are loaded based on identified cluster.

Figure 9 depicts the three distinct phases of iTune which are encoded in the following components: (1) Discoverer, (2) Optimizer, and (3) Observer.

3.5.1 Phase 1: Discoverer Phase (*Offline*)

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 machines is logged

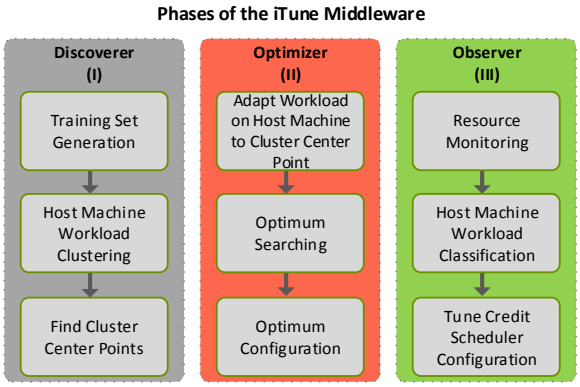


Fig. 9: Three distinct phases of iTune

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 and network 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 [26] 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 [26]. 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 [6], [7], our prior work [12], and our analysis results in Section 3.3, 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, network utilization, CPU overbooking ratio, and VM count of a host machine.* 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 average waiting time percentages of each VM on the

host. The higher the waiting time, the lower the application performance running in the VMs.

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. 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.

The performance model of a host machine for a specified measurement time interval for XenMon can be described by Equation (1). The performance and waiting time are inversely proportional and the goal is to minimize the cumulative waiting time of host machine in Equation (1) in Section 3.5.2.

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

where

P : Host machine's performance
n : Total number of the VMs on the host machine
$WaitingTime$: Average 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 in the literature such as artificial neural networks, self-organizing map (SOM), and k-means in the literature. The K-means [23] algorithm is simple and computationally faster compared to other clustering techniques and provides tighter clusters. Thus, we chose to apply it in our approach.

In the k-means algorithm, one of the key input is 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 [24] is utilized to determine the right number of clusters in the training set and measure the quality of the clusters. In this method, the 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. We determined the optimum number of clusters by looking into the mean silhouette values for 3,4,5,6,7,8,9, and 10 clusters. The higher the mean silhouette value is, the better the clustering quality. The best number of clusters which will be provided to the k-means is defined as **5** with maximum mean silhouette value of roughly 0.84.

Step 1.3: Find Cluster Center Points – In this step, the k-means algorithm is employed for the 5 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 illustrated in Table 1.

TABLE 1: Rounded Center Points of Each Cluster

Cluster	CPU Utilization	CPU Overbooking Ratio	VM Count	Network Utilization (MBps)
1	91.8	5.71	15.60	24.28
2	19.5	1.30	10.17	0.54
3	21.12	2.28	10.96	109.90
4	59.06	4.82	12.73	22.18
5	31.43	4.26	12.57	1.39

3.5.2 Phase 2: Optimizer Phase (**Offline**)

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

Step 2.1: Adapt Workload on Host Machine to Cluster Center Point – The center points found in Step 1.3 are utilized in this step. Recall that each center point consists of CPU utilization, CPU overbooking ratio, VM count, and network utilization 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 the Discoverer phase, i.e., the host is configured such that its CPU utilization, VM count, CPU overbooking ratio, and network utilization matches that of the center point. The accommodated workload is retained on the host machine until the simulated annealing algorithm finishes its task and finds the optimum configuration of this accommodated workload which is the representative of a cluster center point to find its optimum configuration.

Step 2.2: Optimum Searching – Simulated annealing is run to pinpoint the optimum solution for each cluster centroids found in the 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. Hence, we preferred to pick a range of values within the limits of each parameter (i.e. 100 different values within the range).

Recall that the 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.

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 the annealing process by modifying the scheduler parameters and retrieving additional resource usage information as described next.

For each annealing process, iTune uses four seconds worth of data sampled every 100 ms time interval. The simulated annealing algorithm checks whether the sum of the average waiting time for each VM during this period is improving. Recall that VMs marked as *LS-1*, *LS-2*, *LS-3*, and *NLS* must acquire *best*, *better*, *good*, and *best effort* performance, respectively. Therefore, the following rules are also assured during the annealing process.

- **Rule 1**: Assure that weight values of each latency sensitivity levels complies with LS-1>LS-2>LS-3>NLS
- **Rule 2**: Set *Cap* value of the highest available latency sensitivity level VM to 0 and rest of the latency sensitivity levels to their vCPU values.

This process continues until the configuration parameters converge to optimum values.

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

3.5.3 Phase 3: Observer Phase (**Online**)

The final phase of iTune is the Observer phase, which comprises three steps: (1) Resource Monitoring, (2) Host Machine Workload Classification, (3) Tuning the 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, network utilization, CPU overbooking ratio, VM count, VM state, vCPU count of each VM, memory size, CPU model, etc. These metrics are retrieved through the libvirt library and saved into the database.

As mentioned before, we collect multiple resource usage information of VMs and the host but only four of these are used in the Observer phase because we are interested in finding which nearest cluster number is closest to the actual workload of the host. These resource usage information are CPU utilization, CPU overbooking ratio, VM count, and network utilization. iTune's default profiling interval is configured to run every 15 seconds. This profiling interval (i.e. heartbeat rate) parameter helps iTune to adjust the heartbeat rate for resource monitoring in the Discoverer and Observer phases. Lower profiling intervals may fail to catch the fluctuations at the workload and may cause frequent reconfigurations. Therefore, 15 seconds was preferred for this configuration parameter. 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, VM count, and network utilization 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 reload the configuration file as long as the cluster number remains same. When the cluster number changes, iTune loads the new cluster settings if that new cluster number persists for at least after five consecutive executions of Step 3.2. The resource monitoring time interval at Step 3.1 (i.e. 15 secs) and the number of consecutive executions monitored for persistence are trade-off values between application overhead and resource usage consistency (i.e. mechanism to eliminate sudden resource usage spikes).

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 [26] 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 assures performance differences between VMs with different latency-sensitivity levels and improves overall 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 showing that indeed iTune is able to provide differentiated performance gains between four different latency sensitivity levels and improve performance for applications by dynamically tuning the Xen scheduler parameter while imposing negligible overhead in the control path.

4.1 Experimental Setup

The experiments were conducted in our private data center which is managed by the OpenNebula [27] cloud management software version 4.6.2. 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 2.

TABLE 2: 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

The iTune engine runs inside Dom0 of the host machine and logs various parameters it needs for analysis and decision making. A python script named as *synthetic workload generator* executes on our cloud management server and invokes OpenNebula APIs to instantiate VMs with different configurations on the experimental host machine.

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 3.5.1). 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 [26]. 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 25, average CPU utilization of host machine varied from 0 to 100%, average network IO usage between 0MBps to 14MBps, and overbooking ratio varied from 0.25 to 9.17.

The *synthetic workload generator* running in the Dom0 spawns: (1) the applications in the *phoronix test suite* [28] open-source testing and benchmarking framework, (2) *lookbusy* synthetic load generator processes, (3) *netperf* [29], (4) *sysbench* [30], and (5) *httperf* [31] in the VMs to generate the desired workload on the selected host in our private data center. These benchmark applications comprises CPU-bound, network-bound, memory-bound, and disk-bound type applications to represent workload realistically. We think that the diversity of the applications used from all of these benchmarking frameworks is enough to represent real-world workloads. However, to make the training set more and more realistic various other type of applications such as tightly-coupled parallel applications [32], multimedia applications [33], etc. can also be utilized.

For this specific scenario, the optimum number of clusters identified by our training data was 5 with the highest mean silhouette value of 0.84.

4.3 Application Performance Improvement using iTune

The experiments in this section were conducted to validate the effectiveness of the iTune framework where we compare the performance differences between VMs with different latency sensitivity levels as well as the improvement of applying our approach over the default one.

In this set up, we created a random workload having 19 VMs, each using CPU varying between 10% and 60% on a host machine in our private data center. To illustrate a host machine in a real data center, VMs were created with varying workloads and applications randomly selected from benchmark suites. The validation of iTune was conducted through sending concurrent web requests from four clients to Apache web server and Netperf application in two seperate test cases. In Figure 10, the set up to validate the iTune is illustrated. There are four VMs marked as ***LS-1***, ***LS-2***, ***LS-3***, and ***NLS***. These VMs host Apache web server having the identical configuration and requests are sent from the clients marked as ***User 1***, ***User 2***, ***User 3***, and ***User 4*** in Figure 10. Initial experiments showed us that originating all four requests from a single server, single VM, or four separate VMs on the same server produce inconsistent and unfair test results between different test cases. Therefore, for fair testing practices and consistent results, each client/user sending requests are originated from four different non-virtualized bare metal servers as illustrated in Figure 10.

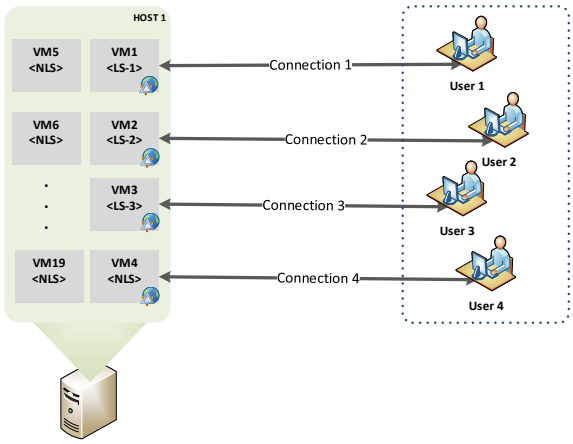


Fig. 10: Illustration of iTune's Validation Environment

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. We validated iTune with the performance evaluation of two different applications: (1) Apache web server (Use Case 1) and (2) Netperf (Use Case 2).

For the Apache web server case, we ran the experiments for two minutes resulting in about 41K data points which was sufficient enough to make decisions. For the Netperf case, we ran the experiments for five times, each test for a duration of two minutes and averaged the results out for both default and iTune optimized parameters.

Observer phase of iTune detected actual load on host machine was close to **Cluster 3** in Table 1 at Use Case 1 and Use Case 2. Therefore, the optimum configuration for **Cluster 3** was loaded autonomously. Table 3 shows the default and iTune optimized configuration values. Cap values in this Table for the VMs marked as LS-2, LS-3, and NLS is 100 times their vCPU count which means that these VMs cannot utilize more vCPU than the one assigned to them even if there is idle pCPUs available. The overhead of using iTune in the runtime phase is negligible.

TABLE 3: Default and iTune Optimized Configuration Parameter Values

Configuration Name	Default	iTune
Timeslice (ms)	30	34
Ratelimit (μs)	1000	1600
Dom0 Weight	256	62208
Dom0 Cap	0	0
VM<*LS-1*>Weight	256	43008
VM<*LS-1*>Cap	0	0
VM<*LS-2*>Weight	256	18176
VM<*LS-2*>Cap	0	100*vCPU
VM<*LS-3*>Weight	256	2560
VM<*LS-3*>Cap	0	100*vCPU
VM<*NLS*>Weight	256	256
VM<*NLS*>Cap	0	100*vCPU

4.3.1 Use Case 1: Apache Web Server

The evaluation results with the Apache web server is presented in this section. Figure 11 shows the comparison of Apache web server's throughput in four different VMs (Shown as VM1, VM2, VM3, VM4 in Figure 10) with different latency sensitivity levels under 250 and 500 concurrent users load. Throughputs in Figure 11 were measured under Credit Scheduler's default configuration and when iTune loaded optimum scheduler configuration.

As can be seen in Figure 11a and Figure 11b, since there is no latency differentiation between VMs in the default Credit Scheduler configuration, VMs marked as LS-1, LS-2, LS-3, or NLS latency sensitivity levels does not guarantee the same level of throughput between different experiments. At each experiments a different VM may receive the best performance. There is no assurance for a VM to get the best performance.

However, in the case of iTune, the optimum scheduler configuration is loaded autonomously and VMs marked as LS-1, LS-2, LS-3, and NLS gain the best, better, good, and best effort througputs, respectively. Recall that the experiments were conducted five times and average of throughput values were taken. iTune configuration provided the same trend between different latency sensitivity levels whereas there is no way for default configuration to achieve this.

When throughputs in Figure 11a and Figure 11b are compared, it is clearly seen that iTune again provides clearer distinction under 500 concurrent users load. The reason for better distinction under 500 concurrent users load comparing to the 250 concurrent users load is because of high CPU demand under 500 concurrent users, so iTune is favoring LS-1 and LS-2, and is more effective as parallelism increases.

Figure 12a and Figure 12b show the comparison of web server response time for all four latency sensitivity levels in median, 90, 95, and 99 percentiles metrics under 250 concurrent users load. Response times along all four metrics are close to each other and LS-3 gain the best response time at 99 percentile under default configuration. In all four metrics, iTune assured best, better, good, and best response times for LS-1, LS-2, LS-3, and NLS as shown in Figure 12b. Additionally, iTune also provided better response time values than default configuration for LS-1, LS-2, and LS-3.

Figure 13a and Figure 13b also show the comparison of web server response time for all four latency sensitivity levels in median, 90, 95, and 99 percentiles metrics under 500 concurrent users load. Again, iTune assured the different performance gains between latency sensitivity levels and clear response time differences in all four metrics. Additionally, iTune provided better response time values than default configuration for LS-1 and LS-2.

4.3.2 Use Case 2: Netperf

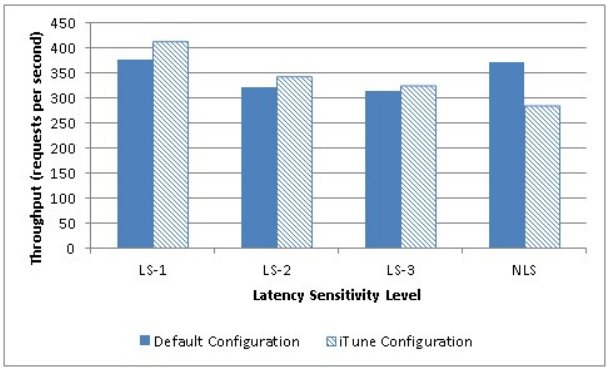
The evaluation results with the Netperf benchmark is presented in this section. Same set up is utilized as in use case 1 in Figure 10.

Figure 14 shows the comparison of Netperf throughput (i.e. complete transactions exchanged per second) under 6 and 12 users load. At each user load, consecutive TCP request/response tests are sent to the Netperf server known as Netserver for a period of two minutes synchronously, one transaction at a time.

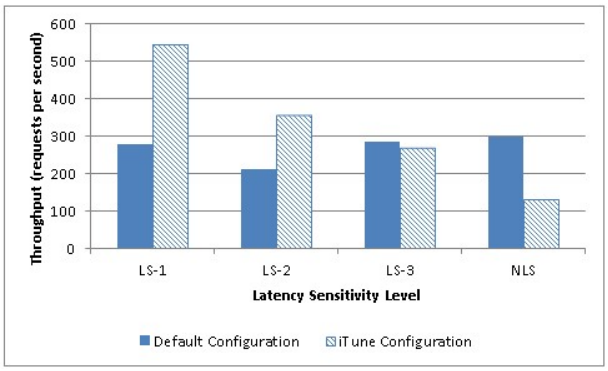
The Netperf test results in Figure 14 show the same trend with the Apache web server test. Figure 14a shows the comparison of complete transactions per second between Netperf clients on four bare metal servers and Netserver applications on four test VMs. As can be seen in Figure 14a, iTune configuration provided necessary performance differences between latency sensitive applications versus no clear differentiation under default configuration. Netperf throughput test under 6 and 12 concurrent users load in Figure 14a and Figure 14b show the similar performance trend where default configuration does not provide performance difference and iTune configuration always assured best, better, good, and best effort level of throughput for LS-1, LS-2, LS-3, and NLS latency sensitivity level VMs.

4.3.3 Host-level Performance Improvement

Table 4 shows the sum of average waiting time of each VMs on the host machine and were obtained while experimenting Use Case 1 and Use Case 2. Use Case 1 and Use Case 2 validates iTune at VM-level, but do not show whether there is improvement on the rest of the VMs on the same host machine. Therefore, we showed the overall waiting

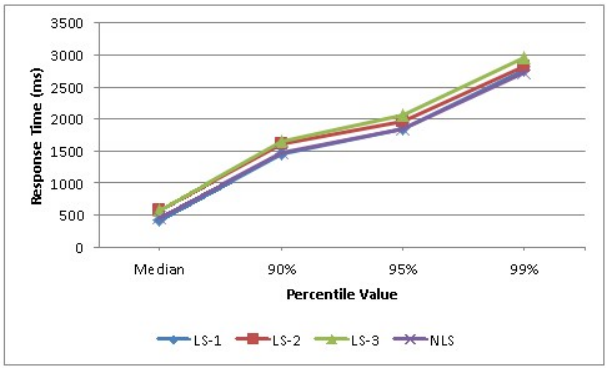


(a) Under 250 Concurrent Users

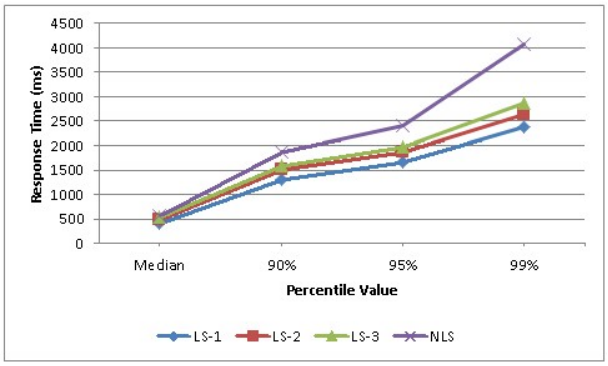


(b) Under 500 Concurrent Users

Fig. 11: Comparison of Web Server Throughput Under 250 and 500 Concurrent Users Load

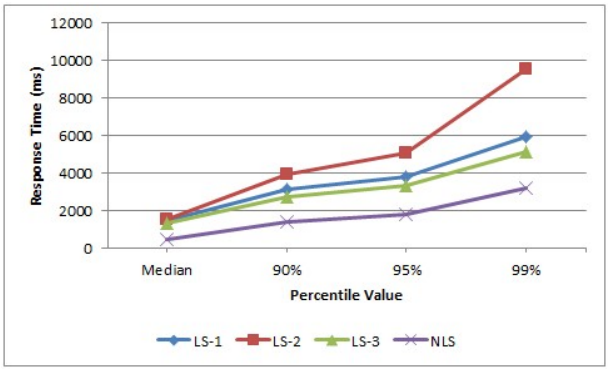


(a) Default Configuration

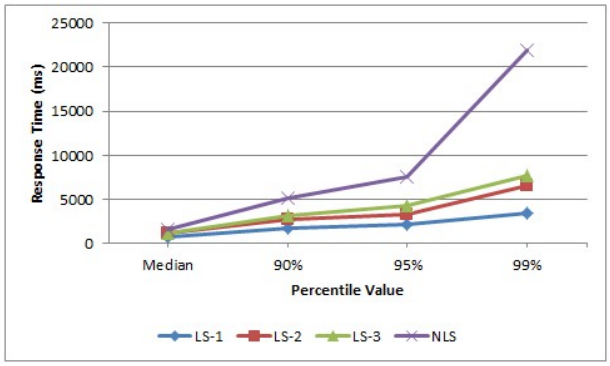


(b) iTune Configuration

Fig. 12: Comparison of Web Server Response Time Under 250 Concurrent Users Load



(a) Default Configuration



(b) iTune Configuration

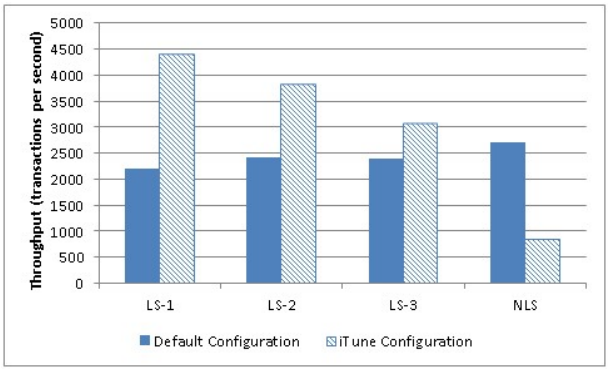
Fig. 13: Comparison of Web Server Response Time Under 500 Concurrent Users Load

time improvement to get a holistic view of perfomance improvement at the host-level.

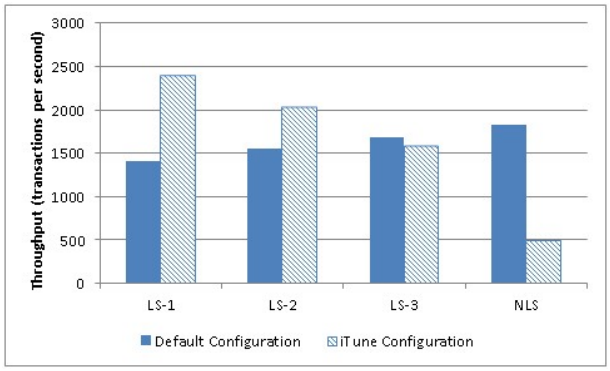
As seen in Table 4, overall waiting time improvement of 41.51% and 52.45% were gained for the experimental host. In other words, this means that the waiting time improvement at the host-level is reflected as application-level performance improvement residing on the VMs. Hence, the experiments validate 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



(a) Under 6 Concurrent Users



(b) Under 12 Concurrent Users

Fig. 14: Comparison of Netperf Throughput Under 6 and 12 Users Load

TABLE 4: Host-level Improvement Results

Setup	Xen Default	iTune	Improvement (%)
Total of Average Waiting Time of all VMs (Apache)	20006%	11701%	41.51%
Total of Average Waiting Time of all VMs (Netperf)	7493%	3563%	52.45%

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 to optimize the scheduler parameters of the hypervisor. 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. Unlike the other two phases, the Observer phase is online and 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. iTune also provides four different latency sensitivity levels to the VMs.

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 5 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 in the Google cluster trace. 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 to overcome this limitation.
- 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 other limitation of iTune and part of our future work.
- Our recent work has explored the dimensions of resource overbooking to conserve server-side resources [3], and also power-performance trade-offs in data centers [34]. 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.

ACKNOWLEDGMENTS

This work was supported in part by the US NSF 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.

REFERENCES

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

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

[3] F. Caglar and A. Gokhale, "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: IEEE, Jun. 2014, pp. 538–545.

[4] L. Tomas and J. Tordsson, "An autonomic approach to risk-aware data center overbooking," *Cloud Computing, IEEE Transactions on*, vol. 2, no. 3, pp. 292–305, July 2014.

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

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

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

[8] VMware, "Best Practices for Performance Tuning of Latency-Sensitive Workloads in vSphere VMs," VMware, Inc., Tech. Rep., 2013, www.vmware.com/files/pdf/techpaper/VMW-Tuning-Latency-Sensitive-Workloads.pdf.

[9] "Xen - credit scheduler," http://wiki.xen.org/wiki/Credit_Scheduler, November 2015.

[10] "Oracle - how to tune scheduler parameters," https://docs.oracle.com/cd/E19044-01/sol.containers/817-1592/rmfss.task-21/index.html, November 2015.

[11] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the Art of Virtualization," in *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5. ACM, 2003, pp. 164–177.

[12] D. Wu and A. Gokhale, "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: IEEE, Dec. 2013, pp. 89–98.

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

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

[15] L. Zeng, Y. Wang, W. Shi, and D. Feng, "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.

[16] S. Blagodurov, D. Gmach, M. Arlitt, Y. Chen, C. Hyser, and A. Fedorova, "Maximizing server utilization while meeting critical slas via weight-based collocation management," in *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*. IEEE, 2013, pp. 277–285.

[17] P. Padala, K.-Y. Hou, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, and A. Merchant, "Automated control of multiple virtualized resources," in *Proceedings of the 4th ACM European conference on Computer systems*. ACM, 2009, pp. 13–26.

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

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

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

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

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

[23] J. A. Hartigan and M. A. Wong, "Algorithm as 136: A k-means clustering algorithm," *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, vol. 28, no. 1, pp. 100–108, 1979.

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

[25] "Encog Machine Learning Framework," http://www.heatonresearch.com/encog, July 2014.

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

[27] D. Milojičić, I. M. Llorente, and R. S. Montero, "Opennebula: A cloud management tool," *IEEE Internet Computing*, vol. 15, no. 2, pp. 0011–14, 2011.

[28] "The phoronix test suite," http://www.phoronix-test-suite.com/, March 2015.

[29] "Netperf - a network performance benchmark," http://www.netperf.org, March 2015.

[30] "Sysbench - system performance benchmark," https://launchpad.net/sysbench, March 2015.

[31] D. Mosberger and T. Jin, "httperfa tool for measuring web server performance," *ACM SIGMETRICS Performance Evaluation Review*, vol. 26, no. 3, pp. 31–37, 1998.

[32] H. Chen, S. Wu, S. Di, B. Zhou, Z. Xie, H. Jin, and X. Shi, "Communication-driven scheduling for virtual clusters in cloud," in *Proceedings of the 23rd international symposium on High-performance parallel and distributed computing*. ACM, 2014, pp. 125–128.

[33] S. K. Barker and P. Shenoy, "Empirical evaluation of latency-sensitive application performance in the cloud," in *Proceedings of the first annual ACM SIGMM conference on Multimedia systems*. ACM, 2010, pp. 35–46.

[34] F. Caglar, S. Shekhar, and A. Gokhale, "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: IEEE, Jun. 2014, pp. 48–55.

Dr. Faruk Caglar received the M.S. degree in Computer and Information Systems Engineering from the Tennessee State University, Nashville, TN, USA, in 2008, and M.S. and PhD degrees in Computer Science from the Vanderbilt University, Nashville, TN, USA. He currently teaches at the Department of Computer Engineering, Meliksah University, Kayseri, TURKEY. His research interests include cloud computing, big data, and machine learning.

Shashank Shekhar is a Computer Science PhD student and graduate research assistant at at Vanderbilt University's Institute for Software Integrated Systems, Nashville, TN. His research focuses on developing adaptive resource management algorithms for Cloud and Internet of Things (IoT) devices in cloud. He received his B.E. in computer science and engineering from Manipal Institute of Technology, Manipal, India.

Dr. Aniruddha S. Gokhale is an Associate Professor in the Department of Electrical Engineering and Computer Science, and Senior Research Scientist at the Institute for Software Integrated Systems (ISIS) both at Vanderbilt University, Nashville, TN, USA. His current research focuses on developing novel solutions to emerging challenges in mobile cloud computing, real-time stream processing, publish/subscribe systems, and cyber physical systems.