DYNAMIC RESOURCE MANAGEMENT IN RESOURCE-OVERBOOKED CLOUD DATA CENTERS

By

Faruk Caglar

Dissertation

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in partial fulfillment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

in

Computer Science

August, 2015

Nashville, Tennessee

Approved:

Dr. Aniruddha S. Gokhale

Dr. Douglas C. Schmidt

Dr. Gautam Biswas

Dr. Christopher J. White

Dr. Akos Ledeczi

To my beloved wife Fatma for her patience, encouragement, and support

and

To my daughter Ayse and son Omer.

ACKNOWLEDGMENTS

First and foremost, I would like to express my special gratitude to my advisor Dr. Aniruddha S. Gokhale, for providing me supervision, advice, guidance, and continuous support over the past four years. He made the last four tough years of my Ph.D. studies and research fun with his friendship and humor. I am deeply grateful to him for always being there for research discussions and advice, and so thankful to him for his tireless effort and time he spent on modifications of all my research papers and providing feedback.

I would like to thank Dr. Gautam Biswas, Dr. Douglas C. Schmidt, Dr. Akos Ledeczi, and Dr. Christopher J. White for agreeing to serve on my dissertation committee and their feedback. I am especially grateful to Dr. Gautam Biswas and Dr. John Kinnebrew for discussions we had during the C^3STEM project meetings. I also would like to take this opportunity to thank to Dr. Xenofon Koutsoukos for his suggestions, comments, and discussions at the weekly DDDAS project meetings.

My research has been supported by various agencies and this work would not have been possible without the financial support from them. I would like to thank to the Air Force Research Lab (AFRL), National Science Foundation (NSF), and the Air Force Office of Scientific Research (AFOSR).

I appreciate the feedback I received from the Distributed Object Computing group members: Kyoungho An, Subhav Pradhan, Prithviraj Patil, Shashank Shekhar, Shweta Khare, Yogesh Barve, Shunxing Bao, and Anirban Bhattacharjee. I especially would like to thank Kyoungho An and Shashank Shekhar for devoting time for taking care of our private cluster of host machines and collaborating with me on several papers.

Finally, I would like to acknowledge my parents, Nuh and Zulbiye for providing support. Most importantly, I would like to express my sincere gratitude to my beloved wife Fatma for her help, understanding, patience, and continuous support with two kids.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

I.1 Emerging Trends

Significant volumes of data are generated by a heterogeneous set of sources, e.g., mobile devices, social media, and a number of the sensors surrounding us. It is estimated that in a span of one Internet minute, a hundred hours worth of video are uploaded to YouTube, about four million searches are conducted in Google, and more than three million pieces of content are shared. In the next five years, it is expected that mobile traffic will have grown thirteen times more than the existing mobile traffic and there will be three times more connected devices than the number of people on the Earth [45]. Moreover, scientific experiments such as CERN also generate enormous amounts of data estimated to be about twenty-five petabytes in a year [67]. With the emergence of the Internet of Things (IoT) paradigm, billions of data points are generated and as a result, the volume of this data is getting even larger.

All of this generated data must be processed to extract useful features out of it. This growing, massive amounts of data require more storage and compute resources, which is ultimately provided by the data centers throughout the world and the cloud computing infrastructure. As more and more applications are created, the cloud computing in general and data center in particular have become critical for many projects, enterprises, and research communities. Hence, it will continue to play a crucial role in delivering a variety of services.

It is estimated that 30.2% of computing workloads will be hosted in the public cloud by 2018 rather than on in-house computing resources. Twenty-seven percent of business mailboxes worldwide are in the cloud at present. As of 2012, 38% of businesses had adopted cloud computing with another 29% were making plans to do so. On average, 545

cloud services are used by an organization that embraced cloud services for their business needs. This data emphasizes that cloud computing in general and data centers in particular will increasingly play crucial roles for organizations [12].

Despite the fact that there is a significant momentum towards moving to the cloud, a variety of issues still exist in utilizing the cloud to its fullest potential. For example, energy efficiency, capacity planning, performance management, disaster management, and security are a few major concerns faced by cloud service providers (CSPs) among others. On the one hand, data centers worldwide consume massive amounts of electricity, and growing amounts of electricity will be required as the demand increases. In 2013, 2% of the electricity was used by roughly three million data centers in the US. Moreover, diesel power generators, due to power outages in data centers and power plants, emit millions of tons of carbon [50, 82]. Thus, CSPs must address energy efficiency issues for data centers. A recent initiative by the US Department of Energy (DOE) seeks the data centers to become 20% more energy efficient by 2020 [18]. On the other hand, a survey shows that 64% of cloud customers worry about the poor end-user performance with cloud computing in addition to a 44% reported loss of revenue due to availability, performance, and troubleshooting [69]. Although cloud computing services have been a critical and inseparable part of organizations' every day business requirements and people's daily lives, there remain a range of issues that must be resolved before cloud computing and data centers in particular will be utilized to their fullest extent.

A subset of these challenges are presented and concrete, scientific approaches to resolving them form the key contributions of this dissertation.

I.2 Overview of Research Challenges in Cloud Data Centers

This section provides an overview of the research challenges that are addressed by this dissertation in the context of data center management. To better situate these challenges, a high-level architecture of a cloud data center is depicted in Figure 1. In this architecture,

physical resources such as servers are part of the physical layer which are virtualized by the virtual machine manager (VMM) or the so-called hypervisor in the virtualization layer. Virtualized resources and infrastructure are controlled by the infrastructure management tools in the cloud management layer. Applications and cloud services are executed within the virtualized resource shown on top of the virtualization layer.

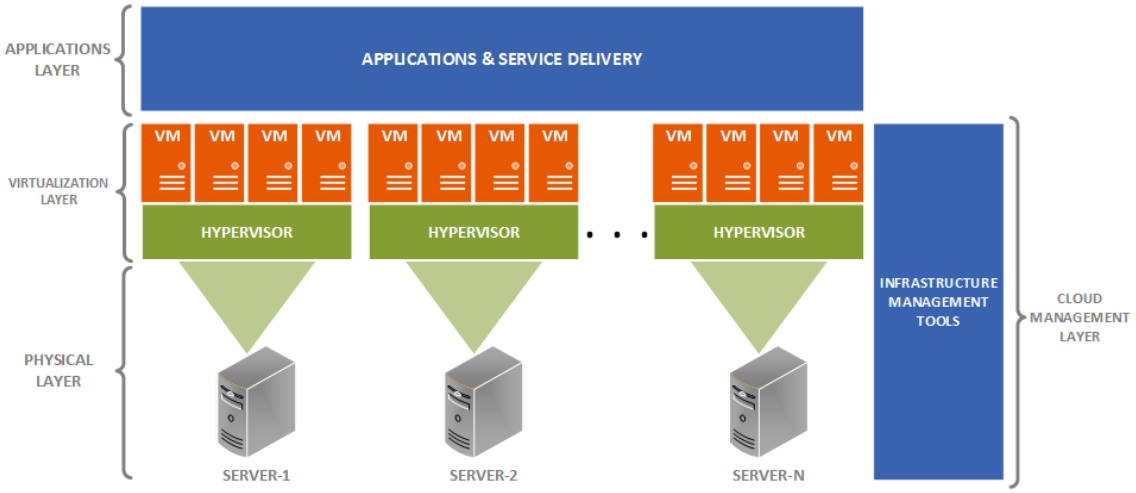


Figure 1: High-level Architecture of a Cloud Data Center

Challenge 1: Autonomous and Dynamic Scheduler Reconfiguration

At the virtualization layer of a data center, hypervisors have a scheduling mechanism to deal with sharing CPU resources among the virtual machines (VMs) and executing the workloads in the VMs. Borrowed Virtual Time (BVT), Simple Earliest Deadline First (sEDF), Credit, and the ESX / ESXi scheduler are a few examples of the schedulers employed by virtual machine managers. Since these schedulers are applicable to many environments and application needs, they are designed to be highly configurable where the chosen parameters for these configurations define how the VMs will be handled and orchestrated, and ultimately the performance delivered to applications hosted in the VMs.

Relying on default values, manually tuning the scheduler's parameters by following

known configuration patterns, using generally accepted rules, and adopting trial-and-error approach, are common practices among the system administrators of the cloud data center. However, these approaches are not effective and efficient, particularly when dealing with dynamically changing workloads on the host machines and varied CPU resource utilizations. Moreover, these non-scientific approaches do not consider the resource overbooking ratios for resource management. Furthermore, often these manual decisions are made offline, which invariably cannot consider the overall system dynamics leading to poor system performance. Therefore, an online, autonomous, and self-tuning system for scheduler configuration is desired.

Challenge 2: Resource-Overbooking to Support Soft Real-time Applications

Under-utilization, wastage of resources, and inefficient energy consumption are among the traditional issues of crucial importance to data centers. The tools in the cloud management layer in a data center are required to monitor, provision, optimize, and orchestrate the underlying cloud infrastructure resources to remedy these issues. CSPs often overbook their resources by utilizing the tools in the cloud management layer. Overbooking is an attractive strategy to CSPs because it helps to reduce energy consumption and increase resource utilization in the data center by packing more user jobs in a fewer number of resources while improving their profits. Overbooking becomes feasible because cloud users tend to overestimate their resource requirements, utilizing only a fraction of the allocated resources. Without overbooking, resources in a data center will otherwise remain under-utilized.

One common way for the data center vendors to overbook resources is to have a predetermined one-size-fits-all overbooking ratio or a method that will determine the ratio of resource overbooking. Resource overbooking ratios are generally determined sporadically by analyzing the historic resource usage of workloads or following the best practices. Unfortunately, governing cloud resources in this manner may be detrimental and catastrophic

to soft real-time applications running in the cloud. To make systematic and online determination of overbooking ratios such that the quality of service needs of soft real-time systems can be met while still benefiting from overbooking, there is a need for more efficient, effective, and intelligent approaches to overbooking that will ensure good performance for soft real-time applications yet prevent under utilization and also save energy costs.

Challenge 3: Performance Interference Effects on Application Performance

Recall that it is a standard practice for CSPs to overbook physical system resources to maximize the resource utilization and make their business model more profitable. Resource overbooking is usually achieved through the tools in the cloud management layer. However, resource overbooking can lead to performance interference and anomalies among the VMs hosted on the physical resources, causing performance unpredictability for soft real-time applications hosted in the VMs. Moreover, resource overbooking can propagate and trigger faults in other VMs. To address these problems and because workloads of the VMs may change at run time, virtual machine migration between physical host machines and data centers is the generally accepted mechanism.

Choosing the right set of target physical host machines for VM migration decisions plays a critical role in determining the performance and interference effects post migration. Analyzing the performance anomalies that might occur and predicting performance interference and fault before a VM is deployed or migrated on the physical host machines is thus desired and vital for soft-real time applications.

Challenge 4: Power- and Performance-Aware Virtual Machine Placement

As mentioned above, virtual machines are migrated from one physical host machine to another one in the same data center or across the data centers located in different locations due to fault tolerance, balance workload, application performance management concerns,

and eliminate hotspots. Deploying, handling, and migrating VMs in a data center are managed by the tools in cloud management layer.

Apart from the performance interference aspects describe above, power and performance trade-offs are also critical and challenging issues faced by CSPs while managing their data centers. On the one hand, CSPs strive to reduce power consumption of their data centers to not only decrease their energy costs but also to reduce adverse impact on the environment. On the other hand, CSPs must deliver performance expected by the applications hosted in their cloud data centers in accordance with predefined Service Level Objective (SLOs). Not doing so will lead to loss of customers and thereby major revenue losses for the CSPs. Power management and performance assurance are conflicting objectives, particularly in the context of multi-tenant cloud systems where multiple VMs may be hosted on a single physical server. The problem becomes even harder when soft real-time applications are hosted in these VMs.

Solutions to address the virtual machine placement decisions exist. Bin packing heuristics such as first-fit, best-fit, and next-fit are common practices used by cloud management platforms (e.g., OpenNebula, OpenStack, etc.) to deploy VMs in the cloud. However, these solutions do not consider application performance and energy efficiency. To address the aforementioned issues, a power and performance-aware virtual machine placement algorithm is desired.

I.3 Doctoral Research Contributions: Dynamic Resource Management in the Cloud Data Center

Addressing the challenges outlined in Section I.2 requires a systematic and scientific approach that is reusable and easily adopted across different cloud computing platforms. To that end, this doctoral research has designed and validated a holistic set of solutions that can easily be integrated into the existing cloud computing infrastructure fabric. The key distinguishing feature of this research is that each of these solutions defines a concrete

and systematic process that cloud service providers can employ for their cloud platforms. Although our solutions were designed and validated in a private data center virtualized by the Xen hypervisor and managed by OpenNebula cloud management tool, the principles behind the solutions are broadly applicable. Concretely, the research contributions of this doctoral research are as follows:

1. **Addressing Challenge 1 → iTune: Engineering the Performance of Xen Hypervisor via Autonomous and Dynamic Scheduler Reconfiguration** Chapter II presents iTune, which is a middleware that optimizes the Xen hypervisor's scheduler configuration parameters autonomously through a three phase design workflow 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.

2. **Addressing Challenge 2 → iOverbook: Intelligent Resource-Overbooking to Support Soft Real-time Applications in the Cloud** Chapter III describes iOverbook, which is an overbooking strategy that uses a machine learning approach to make systematic and online determination of overbooking ratios such that the quality of service needs of soft real-time systems can be met while still benefiting from overbooking. Specifically, iOverbook utilizes historic data of tasks and host machines in the cloud to extract their resource usage patterns and predict future resource usage along with the expected mean performance of host machines. To evaluate our approach, we have used a large usage trace made available by Google of one of its production data centers.

3. **Addressing Challenge 3 → iSensitive: An Intelligent Performance Interference-aware Virtual Machine Migration Approach** Chapter IV describes the iSensitive, which is a machine learning-based middleware providing an online placement solution where the system is trained using events and lifecycle of a publicly available trace of a large data center owned by Google. Our approach first classifies the VMs based on their historic mean CPU, memory usage, and network usage features. Subsequently, it learns the best patterns of collocating the classified VMs by employing machine learning techniques. These extracted patterns document the lowest performance interference level on the specified host machines making them amenable to hosting applications while still allowing resource overbooking.

4. **Addressing Challenge 4 → iPlace: An Intelligent and Tunable Power- and Performance-Aware Virtual Machine Placement Technique for Cloud-based Real-time Applications** Chapter V presents iPlace, which is a middleware providing an intelligent and tunable power- and performance-aware VM placement capability. The placement strategy is based on a two-level artificial neural network, which predicts (1) CPU usage at the first level, and (2) power consumption and performance of a host machine at the second level that uses the predicted CPU usage. The placement decision (i.e., aptly suited host machine for the VM being deployed) is determined by making the appropriate trade-offs between predicted power and performance values of a host machine.

The rest of this dissertation describes the research challenges above in more detail and discusses our solutions along with substantial empirical validation of the presented ideas.

CHAPTER II

ITUNE: ENGINEERING THE PERFORMANCE OF XEN HYPERVISOR VIA AUTONOMOUS AND DYNAMIC SCHEDULER RECONFIGURATION

II.1 Motivation

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 [29, 53] and resource overbooking [19, 42, 80] by packing more VMs on the host machines.

Virtualization systems usually 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 [36, 52]. 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.

II.1.1 Challenges

Our survey of research literature 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 the hypervisor scheduler.

II.1.2 Solution Approach

To address these problems, in this chapter 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 [13], 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 timeslice for each VM. These are the parameters that affect how, when, and how long a VM gains access to shared physical resources.

iTune configures 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. 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 [84], 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 [84]. 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 chapter 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 II.2.4).

- We present an intelligent autonomous and self-tuning middleware called iTune to optimize Xen scheduler configuration (See Section II.2.6).

- We provide options to mark the VMs into one of the 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). Furthermore, iTune assures to deliver the performance requirements of applications in these categories. (See Section II.2).

- 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 II.3).

II.2 Design and Implementation

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.

II.2.1 System Model and Overview of Xen and its Credit Scheduler

In this chapter, we assume a virtualized cloud data center where physical servers employ server virtualization mechanisms. Specifically, in this chapter, we utilize the Xen hypervisor [13] 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

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

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. In this chapter, we leverage *XenMon* [36], 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. The default value assigned to each domain is 256 and the range of values supported are between 1 and 65,535.

- **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.

- **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. The default value is 1,000 and could take values between 100 to 500,000.

- **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.

II.2.2 Problem Statement

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 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

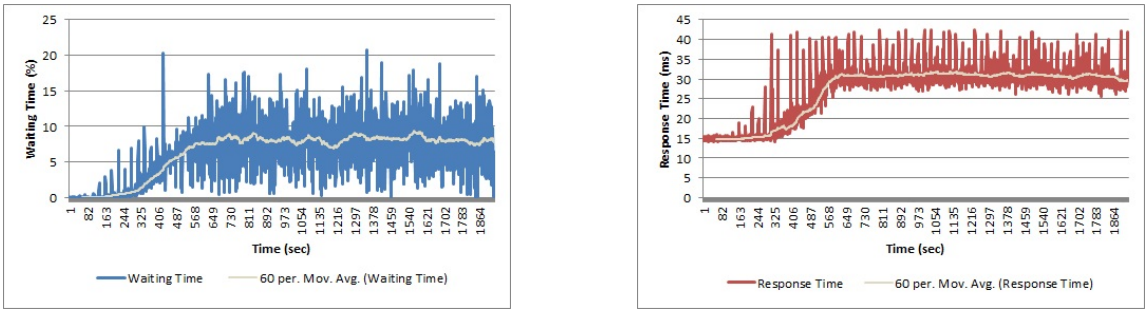
as the system dynamics change due to workload and resource availability changes. Addressing these challenges in an automated way so that the system administrator is relieved of these responsibilities is the focus of this chapter.

II.2.3 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. In Section II.2.4 we show this correlation in the context of a Xen-based system.

II.2.4 Impact of Run Queue Waiting Time on Performance and Resource Utilization

In this section, we present an empirical analysis of how the scheduler waiting time impacts both application performance as well as VM-level resource utilization.



(a) Waiting Time **(b) Apache Web Server Response Time**

Figure 2: Comparison of Web Server Response Time and VM Waiting Time Showing Similar Trend

II.2.4.1 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* [25] 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 [39, 86], 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 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* [3] 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 [26], 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 2.

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.

II.2.4.2 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.

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 3 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

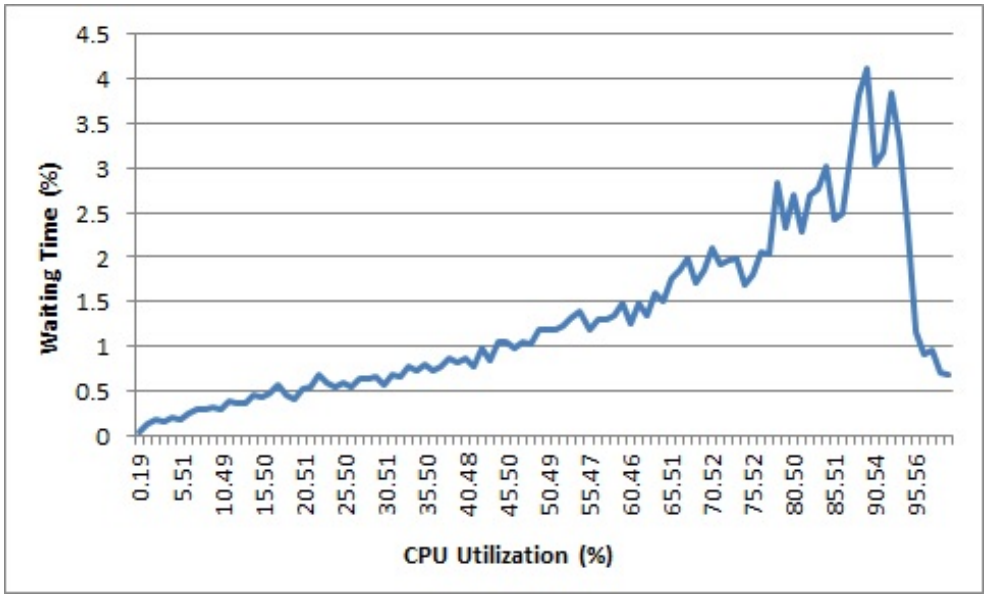


Figure 3: Waiting time vs CPU Utilization for Non-Overbooked Scenario

CPU utilization on waiting time under non-overbooked scenario is low. This also corresponds to the results of application performance experiments shown in Figure 2. 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.

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 4 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.

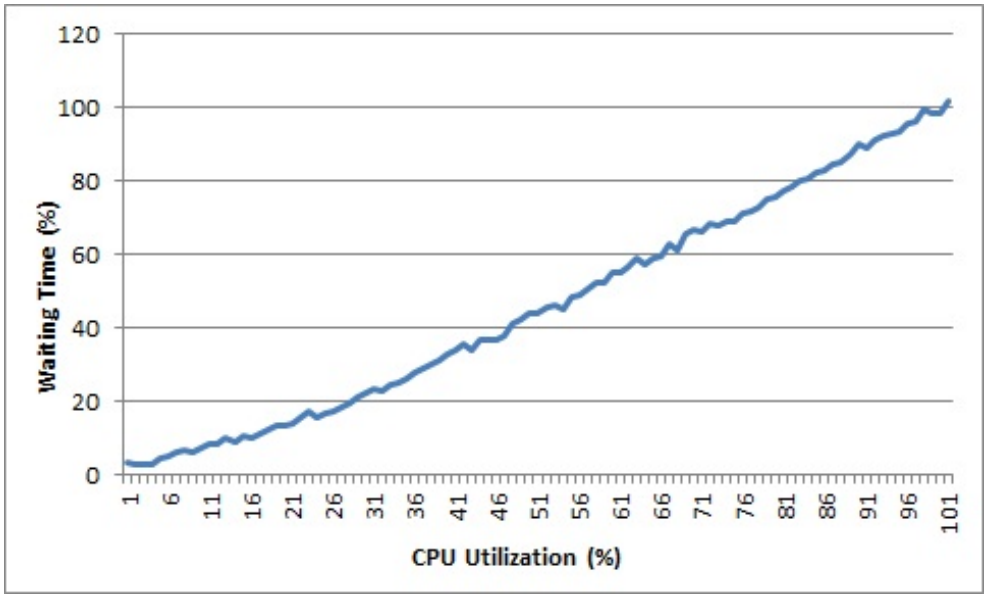


Figure 4: Waiting time vs CPU Utilization for Overbooked Scenario

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 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 5 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.

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

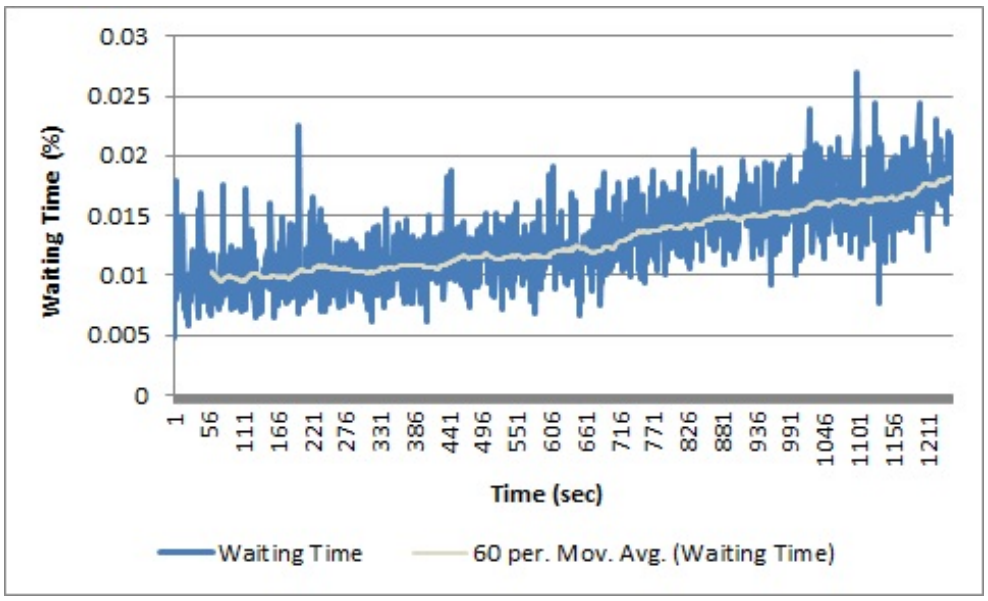


Figure 5: Waiting time vs Memory Utilization for Overbooked Scenario

with step size of about 5KBps increment every minute. The experiment was performed to determine the impact of network utilization on waiting time.

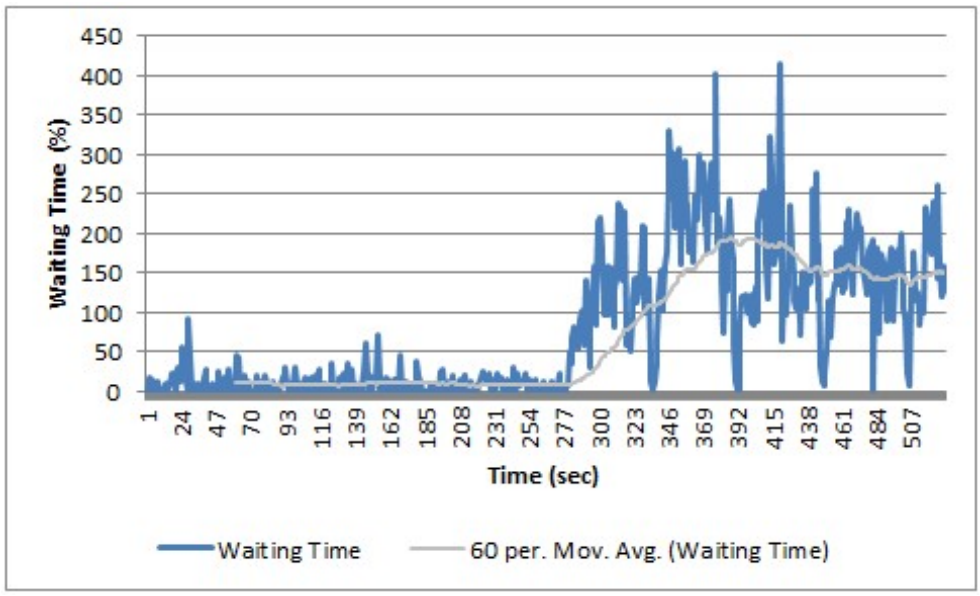


Figure 6: Waiting time vs Network Utilization for Overbooked Scenario

Figure 6 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. Based on these results, for all practical purposes, we will use network utilization as an input parameter in iTune.

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**.

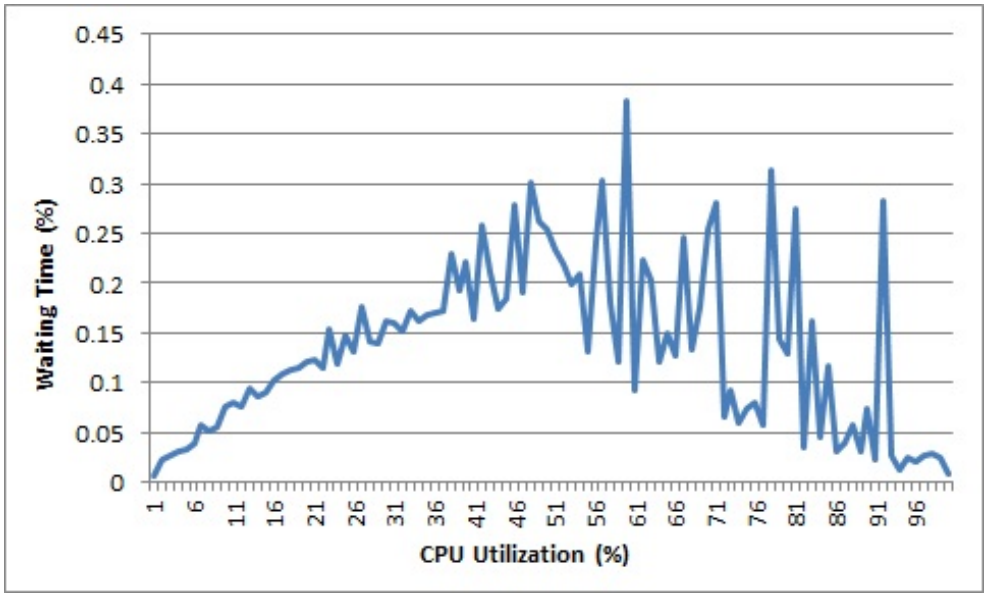


Figure 7: Waiting time vs CPU Utilization for Non-Overbooked Heterogeneous VMs Scenario

Figure 7 represents the result for test **5** that has an analogous trend to test **1**. Similarly, Figure 8 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.

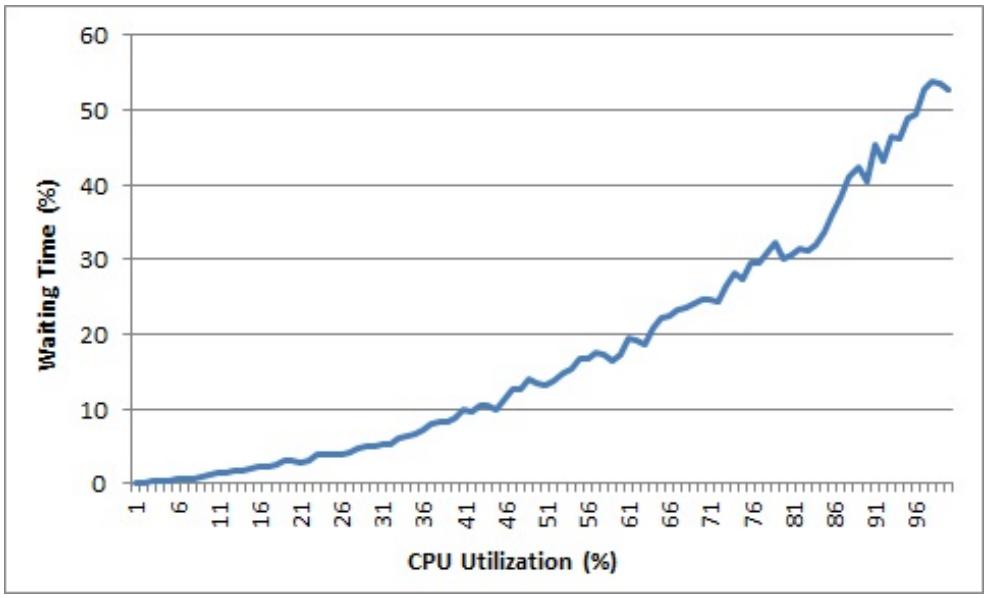


Figure 8: Waiting time vs CPU Utilization for Overbooked Heterogeneous VMs Scenario

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 chapter to *optimize overall waiting time of host machine* which will ultimately satisfy the QoS requirements of the applications running in the VMs.

II.2.5 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 II.2.6. The first two phases are *offline* phases that use a combination of machine learning (specifically, *k-means clustering* [37] and *silhouette* [75] 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.

II.2.6 iTune System Architecture and Implementation

We now present the details of iTune. Figure 9 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 9, 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 [4], 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.

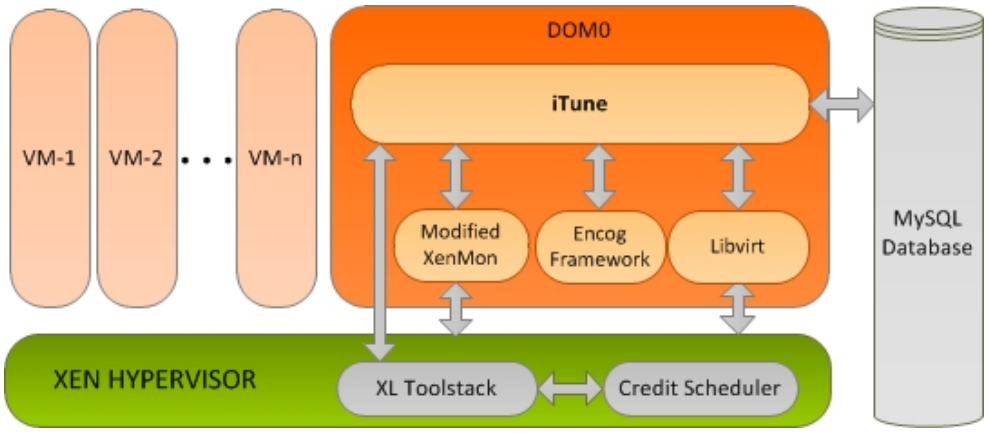


Figure 9: iTune System Architecture

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,

which are described in Section II.2.7, 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.

II.2.7 Three Phases of iTune

Figure 10 depicts the three distinct phases of iTune which are encoded in the following components: (1) Discoverer, (2) Optimizer, and (3) Observer.

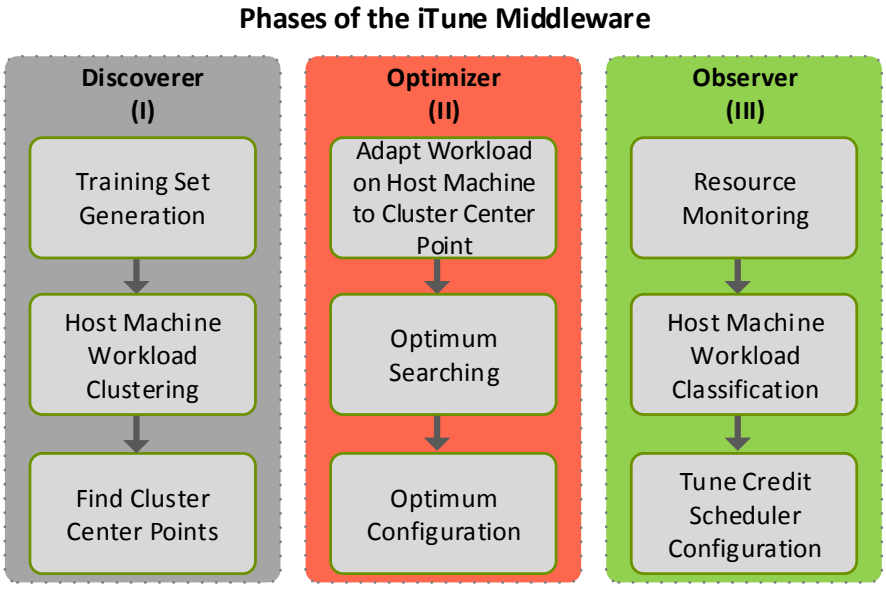


Figure 10: Three distinct phases of iTune

II.2.7.1 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 II.3.2), the workload trace on a host machines 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 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 [74] 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 [74]. 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 [36, 52], our prior work [84], and our analysis results in Section II.2.4, 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 II.3.

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 (II.1). The performance and waiting time are inversely proportional and the goal is to minimize the cumulative waiting time of host

machine in Equation (II.1) in Section II.2.7.2.

$$P = \sum_{i=1}^{n} WaitingTime_i \tag{II.1}$$

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 [37] 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 [75] 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

II.2.7.2 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 – The simulated annealing algorithm is run to pinpoint the optimum solution for each cluster centroids found in the Discoverer phase. As discussed in Section II.2.1, 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 (II.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.

II.2.7.3 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. All of 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 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 cluster number changes, iTune loads the new cluster settings if that new cluster number persists for at least after five consecutive execution of Step 3.2. Resource monitoring time interval at Step 3.1 (i.e. 15 secs) and the number of consecutive execution here are trade-off values between application overhead and resource usage consistency (i.e. mechanism to eliminate sudden resource usage spikes).

II.3 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 [74] 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.

To that end, this section is organized as follows:

- Description of the experimental setup and operating environment (Section II.3.1).

- Description of the experiments conducted to emulate real-world workloads for the training phase of iTune (Section II.3.2).
- Comparing the performance improvements observed using the scheduling parameters suggested by iTune against the default Xen scheduler parameters, and overhead of iTune (Section II.3.3).

II.3.1 Experimental Setup

The experiments were conducted in our private data center which is managed by the OpenNebula [56] 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.

II.3.2 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 II.2.7.1). 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 [74]. 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* [6] open-source testing and benchmarking framework, (2) *lookbusy* synthetic load generator processes, (3) *netperf* [5], (4) *sysbench* [7], and (5) *httperf* [62] 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.

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.

II.3.3 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 11, 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 11. 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 11.

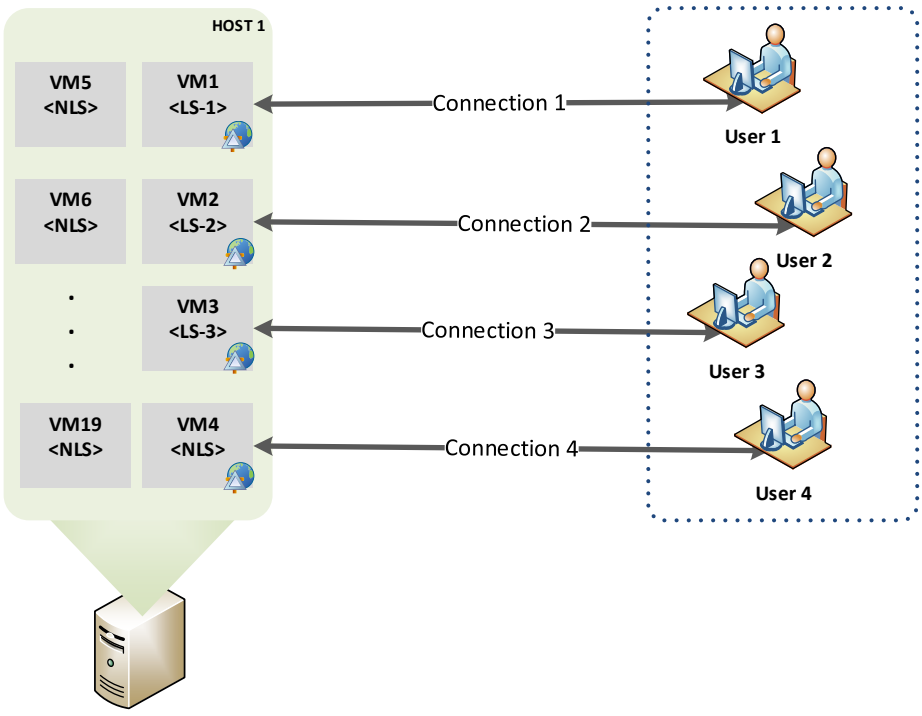


Figure 11: Illustration of iTune's Validation Environment

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.

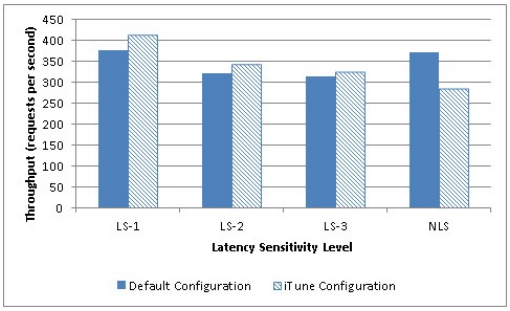
Table 3: Default and iTune Optimized Configuration Parameter Values

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

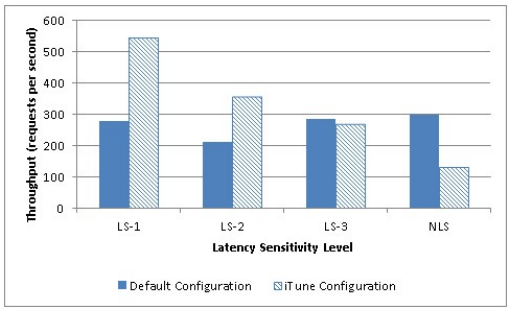
II.3.3.1 Use Case 1: Apache Web Server

The evaluation results with the Apache web server is presented in this section. Figure 12 shows the comparison of Apache web server's throughput in four different VMs (Shown as VM1, VM2, VM3, VM4 in Figure 11) with different latency sensitivity levels under 250 and 500 concurrent users load. Throughputs in Figure 12 were measured under Credit Scheduler's default configuration and when iTune loaded optimum scheduler configuration.

As can be seen in Figure 12a and Figure 12b, 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.



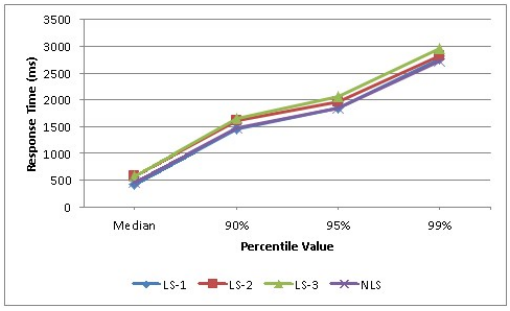
(a) Under 250 Concurrent Users



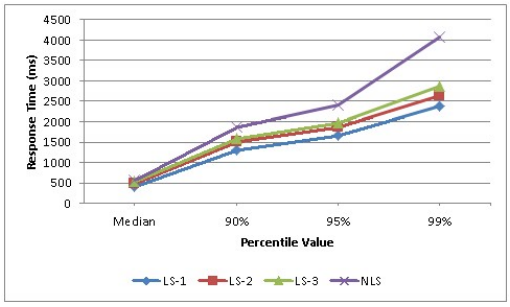
(b) Under 500 Concurrent Users

Figure 12: Comparison of Web Server Throughput Under 250 and 500 Concurrent Users Load

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.



(a) Default Configuration



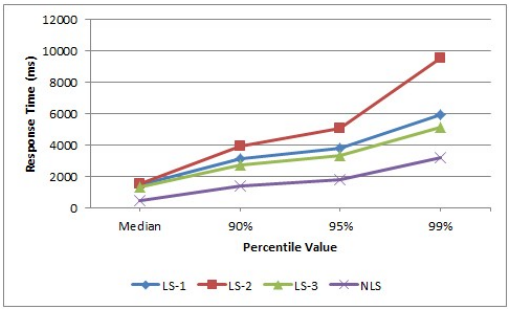
(b) iTune Configuration

Figure 13: Comparison of Web Server Response Time Under 250 Concurrent Users Load

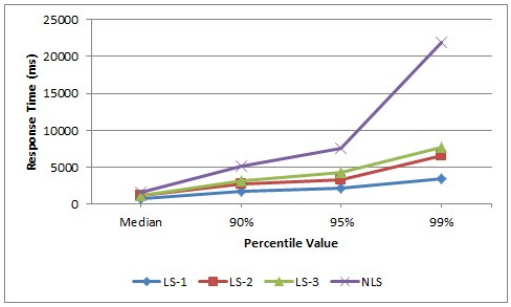
When throughputs in Figure 12a and Figure 12b are compared, it is clearly seen that iTune again provides clearer distinction under 500 concurrent users load. The reason for better distinction under 500 concurrent users load comparing to the 250 concurrent users load is because of high CPU demand under 500 concurrent users, so iTune is favoring LS-1 and LS-2, and is more effective as parallelism increases.

Figure 13a and Figure 13b 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 13b. Additionally, iTune also provided better response time values than default configuration for LS-1, LS-2, and LS-3.

Figure 14a and Figure 14b 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.



(a) Default Configuration



(b) iTune Configuration

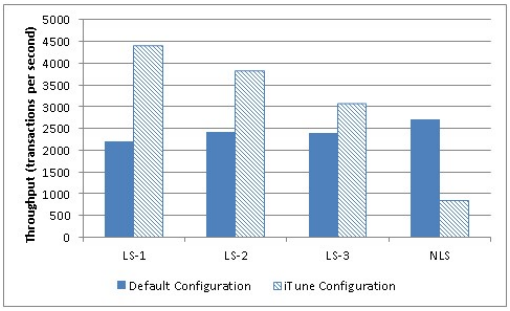
Figure 14: Comparison of Web Server Response Time Under 500 Concurrent Users Load

II.3.3.2 Use Case 2: Netperf

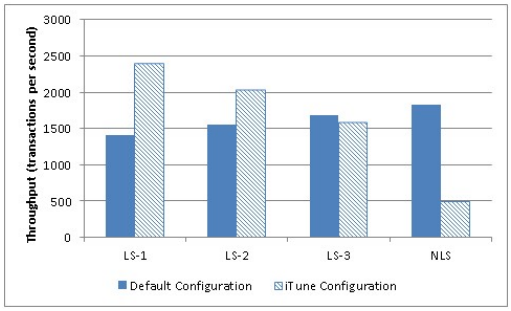
The evaluation results with the Netperf benchmark is presented in this section. Same set up is utilized as in use case 1 in Figure 11.

Figure 15 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 15 show the same trend with the Apache web server test. Figure 15a 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 15a, 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 15a and Figure 15b 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.



(a) Under 6 Concurrent Users



(b) Under 12 Concurrent Users

Figure 15: Comparison of Netperf Throughput Under 6 and 12 Users Load

II.3.3.3 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 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.

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%

II.4 Related Work

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

vSlicer [86] 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. [87] 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. [88] 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. [17] propose a method to manage the collocated applications which were classified as critical and non-critical in the virtualized environment. An increase in 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 chapter. 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. [68] 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 [52] 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 [85] 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.

II.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 chapter 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.
- 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 [19], and also power-performance trade-offs in data centers [21]. 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`.

CHAPTER III

IOVERBOOK: INTELLIGENT RESOURCE-OVERBOOKING TO SUPPORT SOFT REAL-TIME APPLICATIONS IN THE CLOUD

III.1 Motivation

Resource overbooking [1, 15, 59, 60] is a common practice adopted by Cloud Service Providers (CSPs) to increase resource utilization of the servers in a data center and reducing the number of physical servers that are powered on. The outcome for the CSPs is a profitable business model and lower energy bills due to lesser number of servers being used. Resource overbooking entails committing more resources, such as CPU and memory, than are actually available on the physical host machines to the applications – in our case more virtual machines (VMs) that host user applications – that are packed onto physical servers than can actually fit. The resource overbooking technique is a feasible option for CSPs to adopt because cloud users often tend to overestimate the resource requirements for their applications; in reality they use just a fraction of the requested (and hence allocated) resources.

This claim can be validated by observing the dynamics of a production data center whose usage trace is made available by Google Inc [74]. After analyzing the actual CPU usage, the host machine CPU capacity, and the requested CPU capacity of the several host machines in the cluster trace data, we observed that the actual CPU usage of a task is much lower than the allocated amount of CPU, which clearly indicates that users overestimate their resource needs.

Without overbooking, this situation yields very low resource utilizations in data centers, which is detrimental to the CSP as well as to the environment. It is estimated that in Google's data centers, the resource utilization is maintained between 40-60% whereas this percentage is around 7-25% in other data centers [73].

To enable overbooking the servers of data centers that support virtualization, most well-known hypervisors, such as Xen, KVM, and VMware ESX Server support a configuration option for resource overbooking ratios. Even the cloud infrastructure software that manages the cloud platforms, such as OpenNebula, OpenStack, and Eucalyptus allow overbooking. For example, OpenStack has a feature for allowing up to 16:1 and 1.5:1 CPU and memory overbooking ratios, respectively. A 16:1 CPU overbooking ratio means that one physical CPU (pCPU) core can be overbooked by up to sixteen virtual CPU (vCPU) cores. Techniques, such as transparent page sharing, memory ballooning, memory compression, and swapping to disk are some of the methods that hypervisors utilize to make memory overbooking possible [83].

The resource overbooking approach adopted by CSPs tends to be suitable for enterprise applications where most jobs are of the batch processing type and for whom throughput is more important. However, as more applications with soft real-time requirements, such as airline reservations, video streaming (e.g., Netflix), real-time stream processing, and massive open online courses, get hosted on the cloud, resource overbooking may cause significant jitter giving rise to unpredictable performance, which is not acceptable for this class of applications. Moreover, in accordance with the Service Level Agreements (SLA) between the CSP and the customer, service providers have to assure certain performance requirements, such as response time and availability, which is hard to assure without a systematic approach to resource overbooking.

At one end of the spectrum of overbooking possibilities lies lower overbooking ratios, which can result in high satisfaction for cloud users, but can be detrimental to CSPs who would not be effectively and economically utilizing their resources. At the other end of the spectrum exist higher and arbitrary overbooking ratios, which might result in CSPs utilizing their resources effectively thereby saving on energy costs and making their services more profitable, however, the soft real-time systems hosted in the cloud will suffer from not

receiving their desired quality of service (QoS) due to the high resource contention and interference caused by overbooking [64, 71, 79, 89].

III.1.1 Challenges

At one end of the spectrum of overbooking possibilities lies lower overbooking ratios, which can result in high satisfaction for cloud users, but can be detrimental to CSPs who would not be effectively and economically utilizing their resources. At the other end of the spectrum exist higher and arbitrary overbooking ratios, which might result in CSPs utilizing their resources effectively thereby saving on energy costs and making their services more profitable, however, the soft real-time systems hosted in the cloud will suffer from not receiving their desired quality of service (QoS) due to the high resource contention and interference caused by overbooking [64, 71, 79, 89].

The key challenge lies in systematically identifying effective overbooking ratios which will make the right trade-offs in meeting these conflicting objectives. Note that since cloud data centers are made up of heterogeneous machines, a single overbooking ratio may not be effective. Finally, since the cloud environment is highly dynamic, an offline computation of overbooking ratios is not applicable. In the current state of the art, system administrators determine overbooking strategies for their data centers by analyzing the workloads of the VMs through resource monitoring applications or by basing their decisions on earlier studies [1] on optimum overbooking ratios for CPU, memory, and disk . However, none of these contemporary approaches might be appropriate for all the CSPs because of the workload heterogeneity and the risks of errors due to human involvement. These limitations call for an online and autonomous solution.

III.1.2 Solution Approach

To address these limitations, this chapter presents *iOverbook*, which provides an autonomous, online and intelligent, performance-aware overbooking strategy for heterogeneous and virtualized datacenters hosting soft real-time applications. iOverbook autonomously forecasts asymmetric overbooking ratios, *i.e.*, an overbooking ratio per host machine in the data center, by carefully considering the historic resource usage of the applications and not jeopardizing the performance requirements of the soft real-time systems. Specifically, it predicts the mean CPU and memory usage of the physical host machine a future specified time interval – in our case an hour – by utilizing historic resource usage patterns along with some other features, such as CPU capacity, memory capacity, and requests for CPU and memory, and employing machine learning algorithms. Overbooking ratios for the next hour for CPU and memory are then computed based on a mathematical formula. iOverbook continues to adjust these ratios till they converge to a precise value, which will assure certain QoS levels for the hosted applications. The prediction window then slides to the next hour. Note that resource overbooking can cause performance interference and affect VM placement, however, these challenges are investigated in our ongoing work [20].

The research contributions in this chapter are summarized below:

- It presents an intelligent and autonomous, performance-aware overbooking strategy for each host machine in heterogeneous virtualized datacenters that satisfies soft real-time application QoS (See Section III.3).
- Through experimental validations, it analyzes how resource utilization levels can be improved and power consumption reduced in the cloud data centers by utilizing iOverbook (See Section III.4).

III.2 Related Work

This section compares related work synergistic to our work. Predicting future resource usage of VMs based on historic data is a significant aspect of resource overbooking for which our work has leveraged machine learning as a technique. Our decision was based on the observation that machine learning-based approaches have been widely used in different domains for forecasting the future. For example, in the energy domain, [35] predicts future usage of electrical consumption and [66] predicts hot water production, respectively. In the grid and cloud domain, [44] predicts future workload and [60] predicts resource utilization patterns.

Moreno et al. [60] presented a neural network-based overallocation strategy to increase the energy efficiency in data centers and satisfy performance requirements of real-time applications. The mechanism presented in that work predicts the customer's resource utilization based on historic data and computes the amount of resources that will be allocated to a VM by employing cost-benefit analysis and an overallocation algorithm. The work in that paper differs from our work in that it does not provide per-host resource overbooking ratios as we do. However, the forecasting of resource consumption has similarity to our work.

Tomas and Tordsson [81] proposed a cloud computing management framework comprising admission control for horizontal elasticity (*i.e.*, whether to accept more VMs) and scheduling techniques for vertical elasticity (*e.g.*, CPU, memory, and bandwidth). Additionally, they assumed that no SLA violations occur if the used capacity is within the bounds of the physical host machine. This might not always be the case due to the resource contention and interference effects. Our work differs from this work in two ways. First, we provide asymmetric overbooking ratios for a specified timing window (*e.g.*, next one hour). Second, we take many parameters, such as the number of VMs on the host machine and mean CPU usage, into account to precisely predict the performance when overbooked. This significantly alleviates the performance interference problem.

Our earlier work [76] developed a model predictive algorithm for workload forecasting based on which an autonomous framework for resource autoscaling for the cloud was developed. This work was also based on insights gained from usage traces; in that case from the Soccer World Cup of 1998. Although the goals of our previous and current work are performance assurance, the previous work focused on deciding how many resources are needed for a specific application and how to proactively scale them up or down based on prediction of the incoming workload. The end objective was to trade-off performance with the price the customer pays for using cloud resources. In the current work, we take a CSP-centric viewpoint where the objective is to pack as many VMs on the physical resources as possible to maximize resource utilization while being cognizant of application performance requirements.

In the context of supporting real-time applications, Zhang et al. [89] proposed CPI^2 to improve the performance of latency-sensitive jobs when they experience performance interference. CPI^2 detects CPU performance interference incidents by automatically identifying jobs that cause the issue, and optionally shields victim jobs by throttling the triggering task. The authors prove that CPI (cycles-per-instruction) is a good metric to represent application response time. Using these insights, we have used the multiplicative inverse of CPI (*i.e.* instruction per cycle or IPC) as the key metric to measure the performance of tasks and develop our algorithms.

The technique we have presented in this chapter was made possible after gaining deep insights from a usage trace of a production data center released by Google [74]. We have leveraged the findings from recent analyses of this trace [54, 57, 73] that provide deep insights on workload characteristics, task classification, statistical profile and actual resource utilization.

III.3 iOverbook System Architecture and Design

Figure 16 depicts the architecture of iOverbook, which is our intelligent, machine learning-based approach for online determination of effective overbooking ratios for the machines of a data center. Specifically, we focus on the CPU and memory overbooking ratios for each individual host machine within a specified future time interval. Since the online computation of effective overbooking ratios must assure the performance of soft real-time applications, we require an understanding of how the resources are currently utilized and the properties of existing applications so that we can predict the resource usage for a future specified time interval. Once we know this information, we can determine how much overbooking is feasible and whether it is acceptable for soft real-time applications or not.

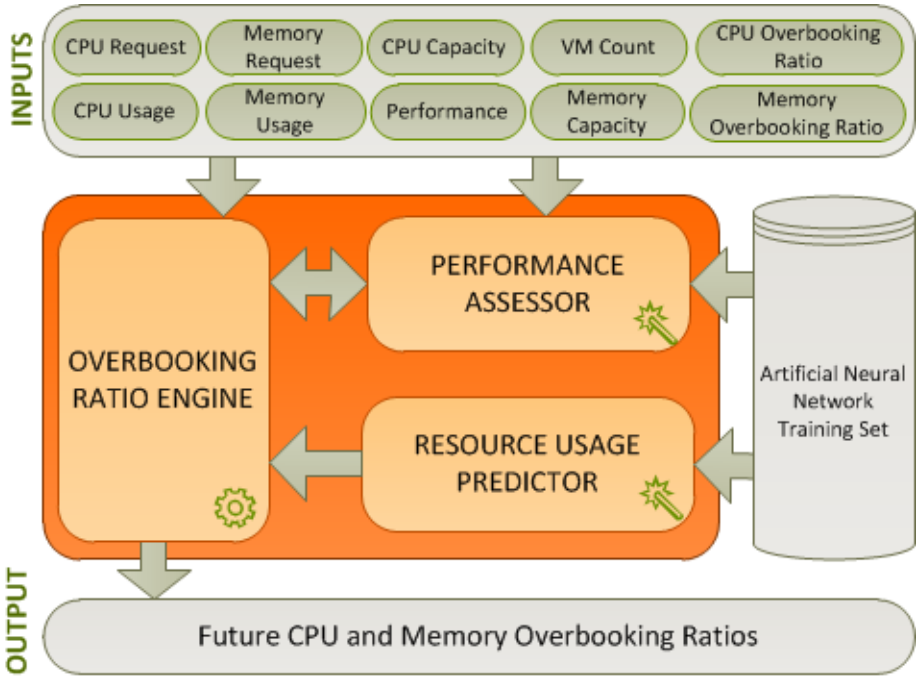


Figure 16: iOverbook System Architecture

These responsibilities motivated a three stage design for iOverbook, which comprises:

(1) a resource usage predictor, (2) an overbooking ratio prediction engine, and (3) a performance assessor. The resource usage predictor and performance assessor components retrieve historic data from a training set repository to train their internal neural networks. iOverbook utilizes mean CPU and memory request, mean CPU and memory usage, mean performance, mean VM count, mean CPU and memory capacity, and CPU and memory overbooking ratios as input parameters. Section III.3.1 justifies the choice of these parameters. We have showcased how iOverbook predicts the overbooking ratios for a time window of one hour, however, this property is tunable. The rest of this section explains the three components of iOverbook.

Google Inc. has released a data center cluster trace 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 [74]. 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. We utilized all 29 days (*i.e.* 696 hours) of data to gain the overall insights and train internal neural networks of iOverbook. To avoid overfitting in the artificial neural networks (ANNs) of iOverbook, the noisy data in the training set was cleared out and numerous sets of training data were provided for generalized training.

Most inputs in Figure 16 are obtained via collecting usage information of the resources. The overbooking ratio inputs are computed using Equation (III.2) for each host machine in

the cluster.

$$TotalResourceAllocated = \sum_{i=0}^{n} ResourceAllocated_i \tag{III.1}$$

$$OverbookingRatio = \frac{TotalResourceAllocated}{Capacity} \tag{III.2}$$

where

$TotalResourceAllocated$: Total amount of resources allocated to all the tasks on host machine

n : Total number of the tasks

$ResourceAllocated$: Size of allocated CPU or memory

$Capacity$: Resource capacity of host machine

III.3.1 Resource Usage Predictor

The purpose of the resource usage predictor is to predict the mean CPU and memory usage of the host machine within the next hour (or the specified time interval). A two layer, feed forward ANN is employed for prediction. ANNs have a powerful ability to model and generalize both linear and non-linear relationships between input and output, and only a hidden layer is sufficient to make any prediction [51]. The sliding window mean CPU and memory resource usage data, and mean CPU and memory requests along with the host machine's resource capacity are the extracted features that are provided to the resource usage predictor.

The structure of the ANN is depicted in Figure 17. The Levenberg-Marquardt back-propagation algorithm is employed for training the ANN. The topology of the ANN for predicting mean CPU and memory usage within the next specified time interval – in our case one hour – is shown in the mathematical formulation of the ANN below.

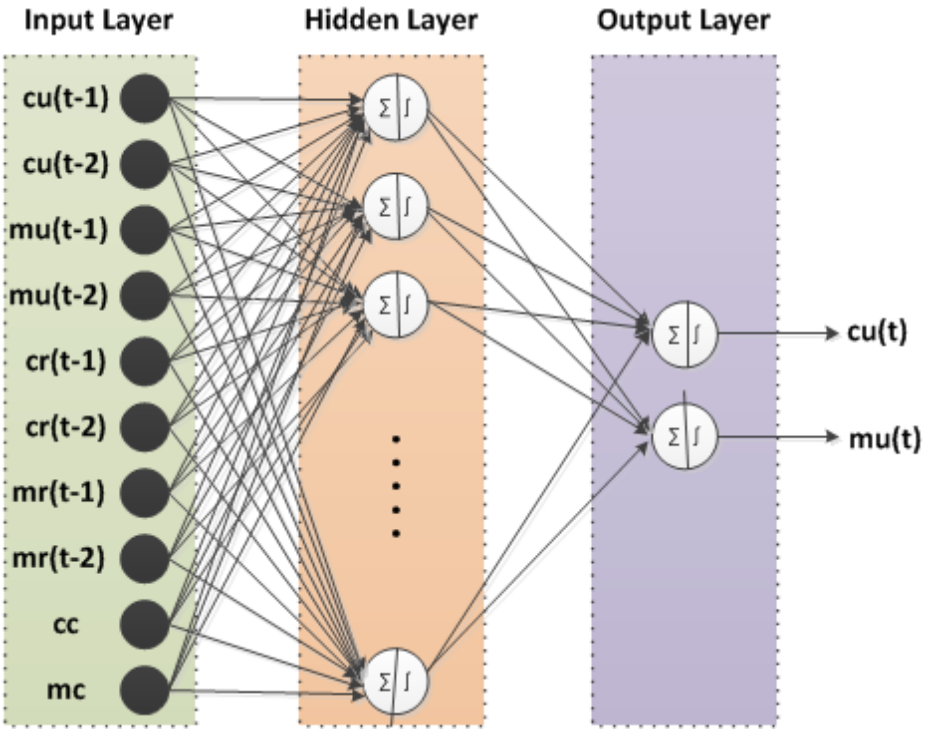


Figure 17: Structure of the Resource Usage Prediction Artificial Neural Network

$$\text{Input Layer} : cu(t-1), cu(t-2), mu(t-1), mu(t-2),$$
$$cr(t-1), cr(t-2), mr(t-1), mr(t-2),$$
$$cc, mc$$

$$\text{Hidden Layer} : 23 \text{ neurons}$$

$$\text{Activation Function} \quad \text{(in hidden layer)}$$
$$: \text{Tangent Sigmoid}$$

$$\text{Output Layer} : cu(t), mu(t)$$

$$\text{Transfer Function} \quad \text{(in output layer)}$$
$$: \text{Pure Linear}$$

where

$t =$ The predicted hour

$cu(t-1) =$ Mean CPU usage at hour $t-1$

$$cu(t-2) = \text{Mean CPU usage at hour } t-2$$

$$mu(t-1) = \text{Mean memory usage at hour } t-1$$

$$mu(t-2) = \text{Mean memory usage at hour } t-2$$

$$cr(t-1) = \text{Mean CPU request at hour } t-1$$

$$cr(t-2) = \text{Mean CPU request at hour } t-2$$

$$mr(t-1) = \text{Mean memory request at hour } t-1$$

$$mr(t-2) = \text{Mean memory request at hour } t-2$$

$$cc = \text{CPU capacity of the host machine}$$

$$mc = \text{Memory capacity of the host machine}$$

$$cu(t) = \text{Mean CPU usage at hour } t$$

$$mu(t) = \text{Mean memory usage at hour } t$$

The reason behind utilizing these input parameters for resource usage prediction is that they are the most common factors affecting the CPU and memory usage of a host machine. CPU and memory capacity are also provided to the ANN due to the heterogeneity of data center machines, which help convey better correlation between input and output.

For testing and experimentation, 40 of the host machines which have the highest mean CPU usage in the 29 days usage of the entire cluster trace are utilized (*i.e.* sufficient number of host machines have been used for the experimental study). The idea behind this filtering is to study only those host machines which hosted more compute-intensive tasks.

The best performance of the ANN was produced with 23 neurons in the hidden layer using a trial-and-error approach with the mean squared error value (MSE), which is the averaged squared difference between inputs and outputs, of 0.0001. The regression (R) value, which is the correlation between inputs and outputs, is 0.9. The generated MSE and R values indicate that the resource usage predictor predicts outputs with a negligible error

value, and that the output parameters of the ANN are very well correlated with its input parameters.

The selection of the activation function made in the hidden layer and output layer are based upon the ANN type (*e.g.*, back propagation dictates an activation function in the hidden layer that provides a derivative), desired output value constraints, and on trial-and-error performance results of the ANN.

The predicted CPU and memory usage values along with the actual usage values for each host machine are illustrated in Figure 18. The training ANN involved using 695 hours of the cluster trace except the 696th hour. The prediction was made for the 696th hour. As seen in Figure 18, the predicted resource usage value follows the actual usage values well enough because of the decent MSE and R values.

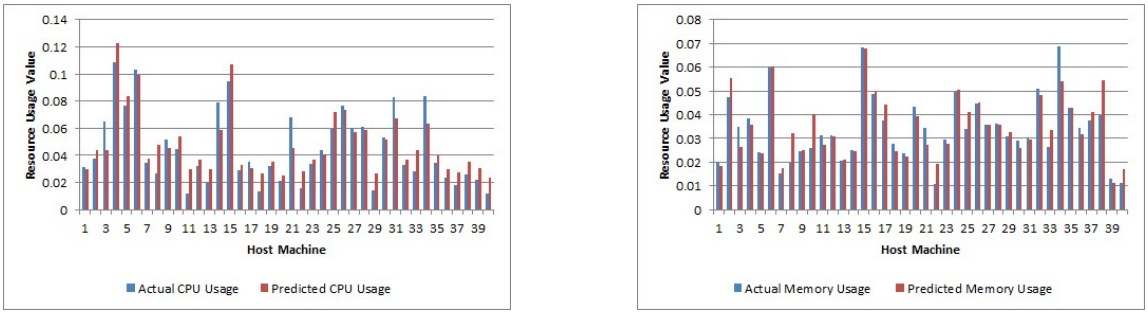


Figure 18: Actual and Predicted Hourly Mean CPU and Memory Usage Value Comparison

The resource usage predictor can also predict the resource demand when flash crowds occur by refining itself through learning new resource usage data. This challenge has already been addressed and discussed in our earlier work [76].

III.3.2 Overbooking Ratio Prediction Engine

After the resource usage predictor predicts the CPU and memory usage for the next one hour time window, the overbooking ratio prediction engine computes the CPU and memory

overbooking ratios per machine, and hands it to the performance assessor. The performance assessor component predicts the performance by using these new overbooking ratios and hands it back to the overbooking ratio prediction engine. This two way communication between the overbooking ratio prediction engine and performance assessor iterates until the predetermined convergence values (calculated manually from the historic data in trace) are satisfied. These convergence values are justified in Section III.4. The details of the computation are shown in Equation (III.4). A discrete step size called "SecuritySlack" is used by our iterative approach to converge on an acceptable overbooking ratio.

$$SecuritySlack(t) = Capacity(t) \times SecurityPercentage \tag{III.3}$$

$$OverbookingRatio(t) = \lambda \times \frac{Capacity(t) - SecuritySlack(t)}{PredictedUsage(t)} \tag{III.4}$$

$$ResourceRequest(t) = OverbookingRatio(t) \times Capacity$$

where

t : The predicted hour

λ : Elastic capacity to converge the best ratio if predicted performance is too high

$Capacity(t)$: Resource capacity (e.g. CPU and memory) of a host machine at hour t

$SecurityPercentage(t)$: Elastic capacity on a host machine to converge the best ratio at hour t

$OverbookingRatio(t)$: CPU and memory overbooking ratios at hour t for a host machine

III.3.3 Performance Assessor

The performance assessor component is responsible for predicting the performance thereby providing an assurance that the new overbooking ratios computed by the overbooking ratio engine do not violate the SLAs. The performance assessor uses the instruction per cycle (IPC[1]) as the performance metric which means that the higher the value, the better the performance. The SLA violation is checked based on the historic maximum performance values (max) on a per-host basis. These threshold values are derived from the tracelog, however, a domain expert may also decide and assign these values.

If the machine under consideration's predicted IPC is greater than the maximum observed IPC in the trace of the same host machine reached in the cluster, iOverbook assumes that SLA will not be violated thereby providing performance assurances to soft real-time applications. If the predicted IPC violates the SLA, iOverbook does not allow overbooking that particular host machine. In Section III.4, this SLA violation logic is elaborated upon by taking the standard deviation of the same type of host machines into account for tighter and more realistic performance results.

The structure of the performance predictor ANN is similar to the resource usage predictor ANN in Figure 17, however, with different inputs and outputs. The topology of this ANN for predicting IPC is provided in the mathematical formulation below. It is considered that the allocated amount of resources and mean VM count on a host machine are changed once the overbooking ratio engine computes new ratios. Based upon newly computed ratios in Equation (III.4), the performance assessor is employed to check whether these new overbooking ratios may trigger any SLA violations. As long as the predicted IPC is less than the historic maximum IPC value of that host machine (*i.e.*, a SLA violation will occur), the performance assessor increases the security percentage value in Equation (III.3) by 0.5% and requests new ratios from the overbooking ratio engine till the ratios converge to the values that do not violate the SLA. The security percentage is preferred as

[1] "IPC" metric is used interchangeably with the term "performance"

0.5% to gradually increase the security slack value, which in turn speeds up the convergence to optimum overbooking ratios. If the predicted IPC is too high (which means that the overbooking ratios are suboptimal), iOverbook raises the λ value by 0.5% to rapidly increase overbooking ratios to the best values that do not violate the SLA.

$$\begin{aligned}
\text{Input Layer} &: cr(t), mr(t), cor(t), mor(t), \\
&\quad vm(t), cc, mc \\
\text{Hidden Layer} &: 22 \text{ neurons} \\
\text{Activation Function} &\quad \text{(in hidden layer)} \\
&: \text{Tangent Sigmoid} \\
\text{Output Layer} &: P(t) \\
\text{Transfer Function} &\quad \text{(in output layer)} \\
&: \text{Pure Linear}
\end{aligned}$$

where

t = The predicted hour

$cr(t)$ = Mean CPU request at hour t

$mr(t)$ = Mean memory request at hour t

$cor(t)$ = CPU overbooking ratio at hour t

$mor(t)$ = Memory overbooking ratio at hour t

$vm(t)$ = Mean VM count at hour t

cc = CPU capacity of the host machine

mc = Memory capacity of the host machine

$P(t)$ = Mean performance at hour t

The best performance of the ANN was produced with 22 neurons in the hidden layer by

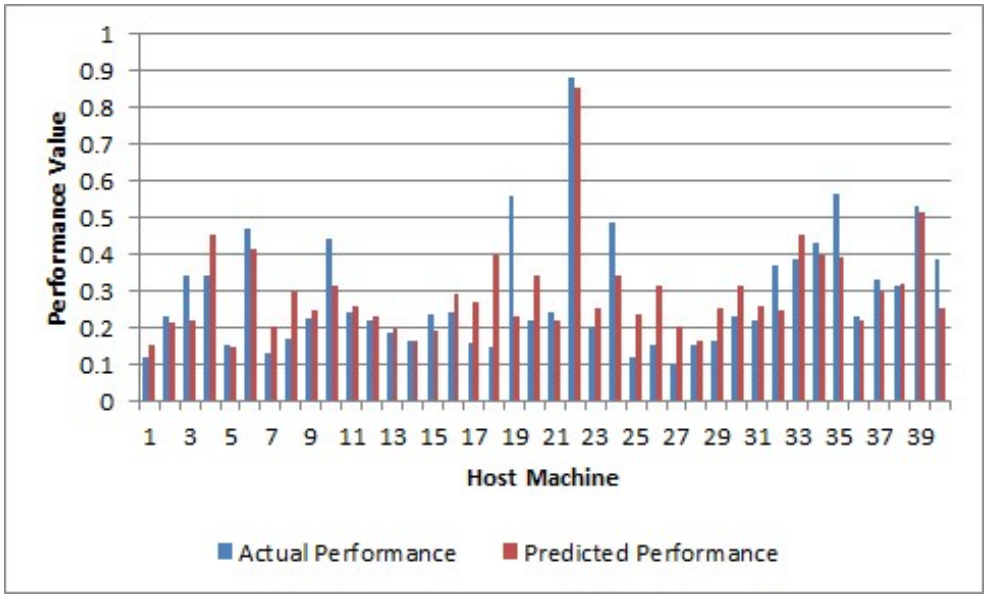


Figure 19: Actual and Predicted IPC Results Comparison

a trial-and-error approach with the *MSE* of 0.009 and the *R* of 0.67. The generated *MSE* and *R* values indicate that the performance assessor component predicts IPC output values with a somewhat higher yet negligible error compared to our resource usage predictor ANN. The outputs of this ANN are well correlated with its inputs.

The predicted IPC value along with the actual IPC values for each host machine are illustrated in Figure 19. The predicted hour is the same as the overbooking ratio prediction engine, which is chosen as the 696th hour in the time line. As can be seen from Figure 19, the predicted performance value follows the actual usage values well because of the lower *MSE* and good *R* values for each host machines in the training set. We believe that additional input parameters will help the ANN to lower the prediction errors and show better correlation.

III.4 Validating the iOverbook Approach

This section validates the iOverbook approach using the Google cluster usage trace.

Validation Approach Since it is not possible to recreate the Google's data center trace, we have used an alternate approach to validating iOverbook. We use part of the usage trace to train iOverbook. Subsequently, we use iOverbook to predict overbooking for a time interval that was not used in the training phase. The results of this prediction are then compared to the actual numbers appearing in the usage trace.

To that end we have used 696 hours of usage trace data to train iOverbook's ANNs except the 696th hour in that interval, and instead used iOverbook to predict the overbooking rates for the 696th hour (*i.e.*, ANN is trained with the first 695 hours of data and tested to predict a future hour, which is the 696th that has not been used in the training). The predicted overbooking rates (both CPU and memory) and performance are compared to the actual overbooking and performance seen from the usage trace.

In our experiments, two different overbooking ratios are computed under two different conditions: (1) if the predicted performance value ($P(t)$) is greater than or equal to the maximum performance value of that host machine in the trace (*i.e.*, $P(t) >= \max$), and (2) if the predicted performance value is greater than or equal to the maximum performance value of the same host machine and seven times the standard deviation of this value (*i.e.*, $P(t) >= \max + 7\sigma$). The motivation behind computing overbooking ratios under two different conditions is to provide results under tighter constraints.

We then analyze how these predicted overbooking ratios for each host machine help to improve the resource utilization and reduce power consumption in the data center. To determine the power consumption, we have utilized SPECpower_ssj2008 [2], an industrial benchmark measuring power and performance values of different computer architectures, to compute power consumption of host machines.

Comparing Actual versus Predicted Overbooking and Performance Figures 20 and 21 compare Google host machines' actual and iOverbook's overbooking ratios computed by the overbooking ratio prediction engine at t=696, for CPU and memory, respectively.

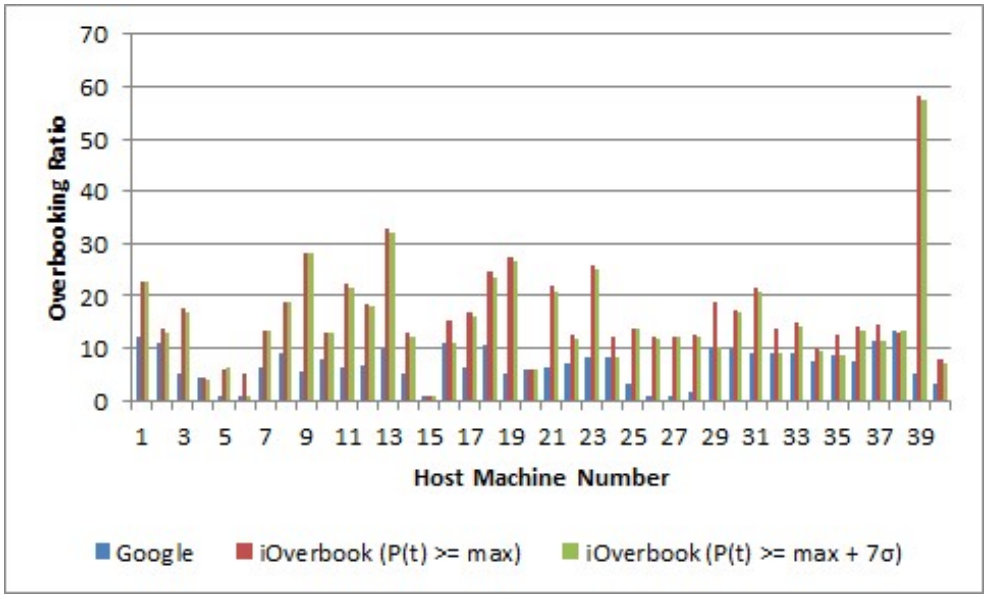


Figure 20: Comparison of Google's Host Machines' Actual and iOverbook's CPU Overbooking Ratios under Different Performance Considerations

In the context of the Google trace, the following inferences can be drawn from Figures 20 and 21. Overall, iOverbook was able to predict higher overbooking ratios for host machines compared to Google's overbooking without SLA violations.

(1) 38/40 host machines could have been overbooked without SLA violation under $P(t) >= \max$ condition (recall that performance is measured as IPC so any value less than the maximum performance is a SLA violation).

(2) 34/40 and 31/40 host machines could have had CPU and memory overbooking ratios greater than ten, respectively, without SLA violation under $P(t) >= \max$ condition.

(3) 30/40 host machines could have been overbooked without SLA violation under $P(t) >= \max + 7\sigma$ condition. These results are somewhat inferior compared to #1 due to a tighter performance constraint.

(4) 31/40 and 24/40 host machines could have had CPU and memory overbooking ratios greater than ten, respectively, without SLA violation under $P(t) >= \max + 7\sigma$ condition. These results are somewhat inferior compared to #2 due to a tighter performance constraint.

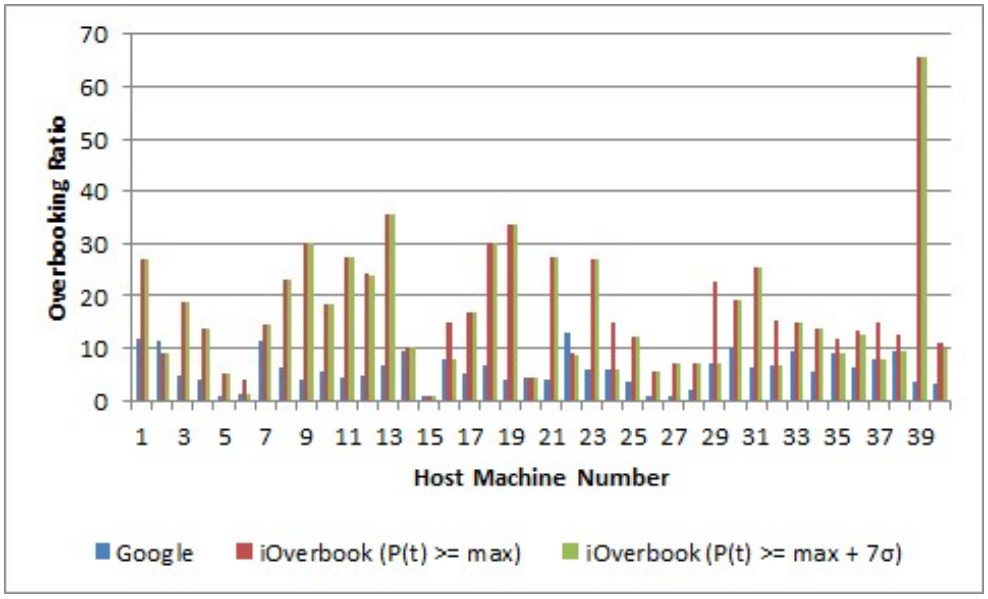


Figure 21: Comparison of Google's Host Machines' Actual and iOverbook's Memory Overbooking Ratios under Different Performance Considerations

In Figure 22, Google's host machines' actual (*i.e*, at t=696) and iOverbook's predicted performance values associated with the overbooking ratios in Figure 20 are depicted. An identical predicted and actual overbooking ratio in the figure means that the machine is not overbooked by iOverbook. As seen from the Figure 22, there are two host machines under $P(t) >= \max$ condition and ten host machines under $P(t) >= \max + 7\sigma$ condition that iOverbook predicted a SLA violation and did not compute overbooking ratios for those host machines and secured their actual ratios. Additionally, iOverbook flagged those cases as SLA violations and left the decision to the scheduler.

Improved Power Savings and Utilization due to iOverbook The cluster trace provides obfuscated configurations of machine attributes, which does not allow us to identify and compute the exact power consumption of those host machines. However, the platform id field in the trace, which provides a sense of hardware architecture type, represents the microarchitecture and chipset version of the host machine in the cluster [74]. Therefore, we surmise the potential configuration of the machine such that it is in tune with the normalized

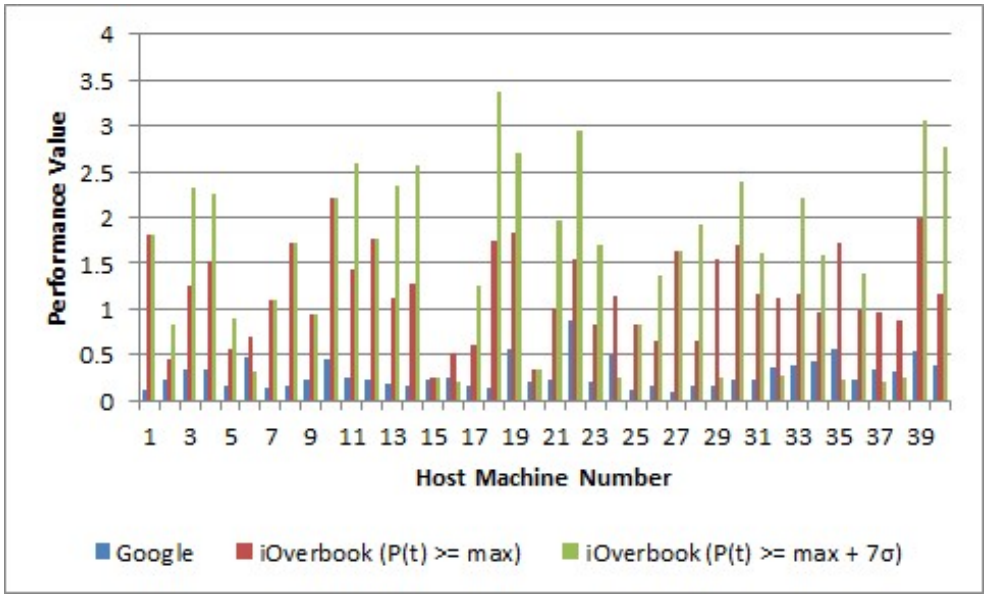


Figure 22: Google's Host Machines' Actual and iOverbook's Predicted Performance Value under Different Performance Considerations

resource capacities of the physical host machines and hardware architecture. The estimated configurations are shown in Table 5.

Table 5: Estimated Host Machine Configurations [2]

Actual Config.	Estimated Configuration			Power Values	
Platform ID/CPU/Memory	Processor Name	CPU (cores)	Memory (GB)	Mean Watts @100%	Mean Watts @Idle
A / 1 / 1	Xeon E5-2650L	32	64	386	142
B / 0.5 / 0.749	Xeon E5-2660	16	48	239	51.6
B / 0.5 / 0.4995	Xeon E5-2660	16	32	257	59.7
B / 0.5 / 0.2493	Xeon E5-2660	16	24	239	51.5
B / 0.5 / 0.1241	Xeon E5-2660	16	16	239	54

The comparison of power consumption results for different consolidation cases are shown in Table 6. Consolidation refers to packing as many tasks on as less number of machines as possible and leaving the rest of the machines at either powered off or idle mode. *Actual Case* is the current status of the host machines. *Case-1* and *Case-2* represent

the status of the host machines under $P(t) >= \max$ condition representing the expected power consumption if the host machines with no tasks on it after the consolidation were powered off or remain powered on but in idle mode conditions, respectively. *Case-3* and *Case-4* are the same cases as *Case-1* and *Case-2* except they show the results under the $P(t) >= \max + 7\sigma$ condition.

Table 6: Power Consumption Results in Test Set

	Total Watt	Number of Servers @On	Number of Servers @Off	Number of Servers @Idle	Savings
Actual	2909.87	40	0	0	0%
Case-1	1964.67	22	18	0	32%
Case-2	3007.97	40	0	22	-3%
Case-3	4873.33	23	17	0	30%
Case-4	7204.53	40	0	17	-3%

As seen from Table 6, iOverbook helped save roughly 32% and 30% of energy for *Case-1* and *Case-3*, respectively. However, if the host machines with no tasks on it are left in the idle state (*Case-2* and *Case-4*), then it has a negative impact on energy consumption, which we surmise can be attributed to the power consumption of host machines in idle mode. Migration costs were not considered in this calculation.

Table 7 compares the actual (from the trace) and iOverbook's resource utilization effects. The "Total CPU" request value in the table is based upon the overbooking ratios computed by iOverbook. The ratio of total CPU usage over total CPU requested that may be named as overestimation ratio is used to calculate the utilization value when the host machines are overbooked with iOverbook. As seen in Table 7, the actual utilization for 40 host machines in the trace was 8.7%. In contrast, the utilization level using iOverbook could have been 21.2% under $P(t) >= \max$ and 19.7% under $P(t) >= \max + 7\sigma$ conditions, which shows an improvement of 12.5% and 11%, respectively. These utilization

levels can easily be raised by manipulating the defined performance constraints, such as utilizing mean IPC rather than the maximum.

Table 7: Resource Utilization Results in Test Set

	Actual Value	iOverbook ($P(t) >= \max$)	iOverbook ($P(t) >= \max + 7\sigma$)
Total CPU Request	137.5874	333.2819	310.4185
Total CPU Capacity	20.5	20.5	20.5
Total CPU Usage / Total CPU Requested	0.0130	0.0130	0.0130
Total CPU Usage	1.7984	4.3564	4.0575
Mean CPU Utilization	8.8%	21.3%	19.8%

Lessons from the Validation Experiments These validation results demonstrate that adding higher standard deviations gives us less beneficial results but probably tighter and a preferred result to assure SLAs. CSPs can utilize our technique by first training our ANNs with their own historic data and then integrating iOverbook with their actual job and VM scheduler in the data center. Therefore, the scheduler could overbook each host machine by considering the overbooking ratios provided by iOverbook. Since our approach allows runtime updates to overbooking ratios, it can adapt autonomously to changing workloads. Therefore, credit-based CPU scheduler in the Xen hypervisor could be integrated with iOverbook to determine weight and cap parameters for each host machine to dynamically. This is the focus of our future work.

III.5 Concluding Remarks

This chapter presented iOverbook, which is an intelligent and online resource overbooking strategy for supporting cloud-based soft real-time applications and effective server utilization by using the insights from Google's production data center usage trace. iOverbook employs two artificial neural networks for predicting a host machine's future resource

usage and performance. It requires historical usage data that cloud providers can provide for use in their data centers. The forecasted values are used in computing significantly better CPU and memory overbooking ratios than those used by Google in their production data center without triggering SLA violations.

The prediction mechanism of iOverbook can be configured and tuned to another desired time interval rather than an hour interval. In the current work, we did not consider the potential for outliers in the available traces. Our future work will investigate effective filtering of outliers and using confidence intervals.

All scripts, source code, and experimental results of iOverbook are available to download at the `www.dre.vanderbilt.edu/~caglarf/download/iOverbook`.

CHAPTER IV

ISENSITIVE: AN INTELLIGENT PERFORMANCE INTERFERENCE-AWARE VIRTUAL MACHINE MIGRATION APPROACH

IV.1 Introduction

Cloud service providers (CSPs) always look to maintaining high resource utilization of cloud computing resources while keeping energy costs low and increasing revenues. To that end they often resort to resource overbooking techniques [15, 19, 60, 80]. The idea behind resource overbooking is to commit more resources, such as CPU and memory, than are actually available on the physical host machines. Statistical multiplexing is the key intuition behind the overbooking strategy where users of traditional cloud-hosted applications often request more resources than their applications actually need and almost never at the same time thereby providing an opportunity to the cloud provider to overbook. Contemporary hypervisors, such as Xen [8], KVM [48], and VMware ESX Server [63], provide the necessary support to make overbooking feasible to implement in practice.

Performance-sensitive applications found in a variety of high performance computing applications are now increasingly hosted in the cloud. Of particular interest are Big data analytics applications on real-time sensor streams related to Internet of Things (IoT). A key trait of this class of applications is its unpredictable arrival pattern of streaming data, which makes it hard to bound the number of requested resources ahead of time. Consequently, statistical multiplexing may not be effective and hence resource overbooking may incur negative impact on application performance because multiple VMs collocated due to resource overbooking can trigger significant performance interference [64, 65, 71, 72, 79, 89].

Performance isolation is an important consideration for hypervisors, however, there is no perfect solution to provide a virtualized environment where there is no performance

interference between VMs [43, 49]. Although there exists prior work on performance isolation [89] among VMs collocated on an overbooked host machine, it is still a challenging task to monitor and consider performance interference for VM placement and shield the VMs from its neighbors due to the nature of resource sharing, resource overbooking practices employed, configuration of underlying scheduling mechanism, other neighboring applications, and the fluctuating workload characteristics in the cloud. Therefore, an application running in one VM might still impact the performance of another application running in a separate VM on the same host machine. Specifically, network- and compute-intensive applications might be adversely impacted.

Addressing the performance interference challenges that stem from resource overbooking and satisfying the performance and response-time requirements of performance-sensitive applications will require effective trade-offs in the placement of VMs on host machines by carefully considering the actual workload characteristics of the VMs. Due to the changing dynamics of the workloads on the VMs and also because VMs often tend to migrate from one physical machine to another for a variety of reasons, traditional and offline heuristics such as bin packing will not be applicable for interference-aware VM placement in cloud computing. Consequently, we have focused on a system architecture that allows monitoring the performance interference variations in addition to the VM placement strategy considering the performance interference effects and the workload characteristics of the collocated VMs on the targeting host machine.

Our prior work to date involving VM-based resource management has considered power-performance trade-offs [21], physical server consolidation using VM overbooking [19], and auto tuning of hypervisor parameters [23] but none of these works account for performance interference between collocated VMs for dynamic resource management.

In this chapter we present an online performance interference monitoring and VM placement technique using a machine learning approach and made available in a middleware called *iSensitive*. In our approach, we first analyze the performance differences in

Base, *Non-Overbooked*, and *Overbooked* environments and compare the impact of resource utilization on performance interference. Using these insights we use machine learning to learn about the desired VM placement patterns and encode these patterns in the runtime middleware that uses them to make runtime VM placement decisions. Our recent work [22] described preliminary work in this regard. For this chapter, however, we made substantial enhancements to our preliminary work. For instance, we completely redesigned our machine learning-based model learning approach. Moreover, we provide extensive empirical validation of our approach.

The rest of this chapter is organized as follows: Section IV.2 describes relevant related work comparing it with our contributions; Section IV.3 presents the system architecture for our machine learning-based solution; Section IV.4 shows the validation results of iSensitive, and Section IV.5 presents concluding remarks alluding to future work.

IV.2 Related Work

This section presents related work on VM placement and solutions to address performance interference, and compares it with our solution called iSensitive.

Novakovic et al. [65] propose DeepDive which transparently identifies and manages performance interference. The experimental results show how throughput and latency of Cassandra No-SQL database is impacted in the Amazon EC2 environment. The measured performance of this application on identical EC2 VMs indicates the performance impacts of collocated VMs. DeepDive comprises three subsystems: (1) warning system, (2) interference analyzer, and (3) placement manager. The warning system conducts interference analysis at a lesser reliability whereas the interference analyzer is employed and starts measurements when the warning system suspects a VM is causing performance interference. The authors mention this as a costly process due to cloning the VM on a separate machine and running it on an isolated environment. The placement manager migrates the VM to another physical host by first mimicing the behavior of the VM being migrated using a

synthetic benchmark. This benchmark is executed on all target hosts for a short period of time before the migration takes place to see whether interference will occur again on the target host. iSensitive differs from DeepDive in that it first learns the best collocated VM patterns and conducts the placement decision based on it without trying to run a mimicked VM on each machines concurrently. Additionally, modeling the exact behavior of an application is a challenging task where enormous type of applications hosted in cloud environment.

In Barker and Shenoy [14], performance interference effects of background workloads (i.e. collocated VMs) on the same host machine are analyzed by evaluating the performance of latency-sensitive online games and gaming servers. It provides experimental results of network jitter and throughput fluctuations in the Amazon EC2 environment. Q-Clouds [64] is a QoS-aware framework to manage performance interference in the cloud. It works on the principle of provisioning additional resources to alleviate performance interference. It applies an online feedback mechanism to build a model for capturing interference interactions and use it for resource management. Moreover, the system employs a staging server to determine the resource requirements and leaves a head room, *i.e.*, slack resource for performance management. Q-Clouds allows specification of different levels of QoS, known as Q-states, to increase the resource utilization. However, the slack resources still lead to under utilization of the server resources. Frequent resource allocation due to feedback mechanism can also cause performance overhead for the hypervisor.

Zhu et al. [90] proposed an interference model which predicts application QoS. It considers time-variant inter-dependence among the different levels of resource contention. The authors develop a resource usage profile as a vector of matrices for different performance metrics and then apply a consolidation algorithm to accommodate applications to minimize interference and achieve QoS. We believe this work focuses on developing simplistic models for complex resource utilization relationships, whereas we use k-means clustering

to group the VMs in different classes to capture the complex relationships and then apply machine learning to determine performance interference.

TRACON [27] is a task and resource allocation framework for data-intensive applications. It develops three interference prediction models: weighted mean method model, linear model and non-linear model using statistical machine learning for reasoning. It then employs an interference-aware scheduler for reducing performance interference. The focus of this technique is network I/O-intensive applications whereas our approach focuses on more application types.

Kambadur et al. [46], studied the methodology and several complexities behind measuring performance interference in data centers due to resource contention and proposed a new technique based on finding the performance interference between base application and co-runners on the same machine. In this work, the authors have measured the performance interference in order to identify interference relationships and classes but have not demonstrated its application. We have leveraged some of the insights and parameters from this work in our work.

Moreno et al. [61] proposed a method for interference-aware virtual machine placement by analyzing its impact on energy efficiency in data centers. The combined interference score utilized in this work requires the knowledge of maximum throughput of each workload running on a host machine when mixed with other workload types. This might require employing some applications to reside on VMs to populate this information from the workload which may result in high overhead when a host runs numerous different types of workloads. In contrast, iSensitive discovers and extracts the best VM patterns by employing machine learning algorithms to learn performance interference levels. iSensitive also differs from this work based on its VM classification features that uses network utilization.

Maji et al. [55], propose an approach named IC2 which handles the performance interference problem from a different perspective. IC2 mitigates the interference by application (e.g. web server or database) reconfiguration through a machine learning-based technique.

IC2 handles the interference at application level rather than hardware level. IC2 targets Apache web server and PHP runtime configuration parameters and considers only three key parameters affecting application performance. IC2 also takes application level indicators to detect the interference where hardware level parameters are not accessible. IC2 improves the application response time to a certain degree. Even though application reconfiguration appears to be an attractive strategy to mitigate the interference at high-level, the complexity of reconfiguration increases as the number of application types hosted in the cloud increases.

IV.3 iSensitive Cloud Middleware Design and Architecture

We now present the design and architecture of the iSensitive middleware that addresses performance interference concerns in cloud platforms.

IV.3.1 Problem Statement

Resource contention and hence performance interference is unavoidable in virtualized environments due to the nature of resource sharing. We have validated this assumption empirically where we analyzed how performance interference stems from resource overbooking and how contention impacts the application performance running in the VMs managed by KVM hypervisor.

The experiments were conducted under three distinct setups named *Base*, *Non-Overbooked*, and *Overbooked*. The workload in the VMs in these setups are generated through applications randomly picked from the *phoronix test suite*. These benchmarking applications were good enough to represent different kinds of workloads and to dynamically generate substantial load in the VMs. *Virt-top* and *jMeter* tools were utilized to log various resource usage and performance metrics. The three setups were as follows:

- **Base**: In this setup, only one VM having 1 vCPU and 512MB of memory and comprising Apache Web Server resides on the host machine. Web requests from 50

concurrent users are posted to the web server from a separate host machine located in the same network cluster.

- **Non-Overbooked**: In non-overbooked setup, the requested resources that are available on the host machine are equal to the available resources. We created 12 VMs each one having 1 vCPU and 512MB of memory. Each VM in this setup comprises a benchmarking application randomly picked from the *phoronix test suite* except the one running Apache Web Server. The benchmarking application is run continuously in the VM. One VM out of 12 handled the web requests initiated from 50 concurrent users from a separate host machine in the same network cluster.

- **Overbooked**: The overbooking ratio used for the CPU resources is 2 which means that the requested CPU resource is two times greater than that are available on the host machine. We created 24 VMs each one having 1 vCPU and 512MB of memory. Again, as it was in the non-overbooked scenario, each VM comprises a benchmarking application which were randomly selected from the *phoronix test suite*. One VM out of 24 managed the web requests initiated from 50 concurrent users.

Figure 23 shows the comparison of web server throughput in *Base*, *Non-Overbooked*, and *Overbooked* scenarios. As can be seen from the figure, the throughput of application performance in these three test scenarios for *Base*, *Non-Overbooked*, and *Overbooked* are 179, 128, and 88 requests per second, respectively, that are handled successfully. As expected, *Base* provides the best application performance since there is no other VM contending for the resources. *Non-overbooked* environment is not as good as *Base*, but better than *Overbooked* environment because there are more VMs contending for available resources which triggers more performance interference between VMs in *Overbooked* scenario.

The CPU utilization of the VM hosting the web server is depicted in Figure 24. The root cause of the performance degradation between three different setups is clearly seen here that in *Non-Overbooked* and *Overbooked* environments CPU utilization is not as good

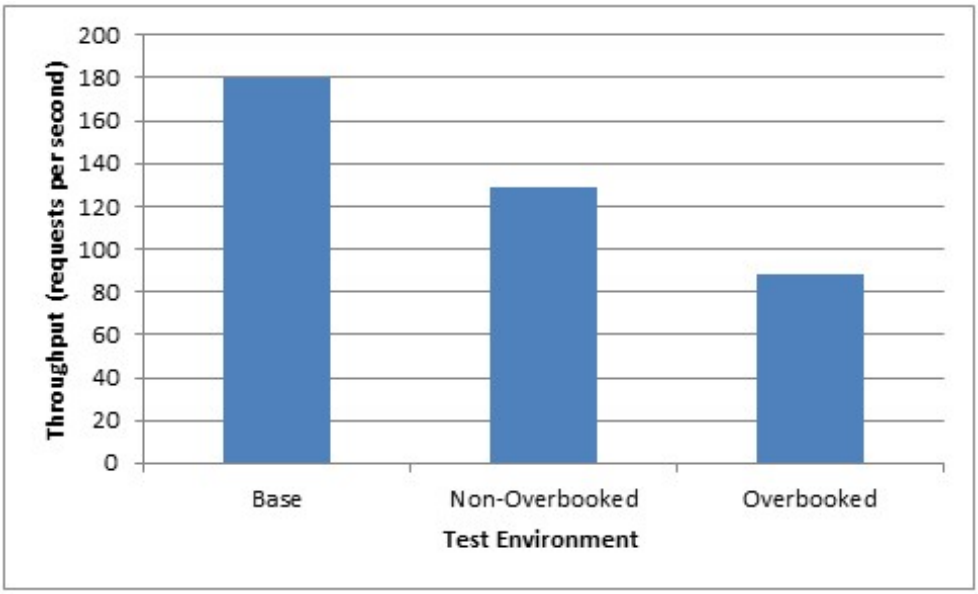


Figure 23: Comparison of Web Server Throughput in Base, Non-Overbooked, and Overbooked Environments

as *Base*. The jitter in the *Overbooked* scenario is considerably higher. As seen by these results, there is a significant performance impact between collocated VMs due to performance interference effects of collocated VMs hence the applications running in the VMs. Even though CPU resources on the host machine were not 100% utilized in both *Non-Overbooked* and *Overbooked*, the performance interference was unavoidable.

Consequently, it is imperative that performance interference must be considered especially for resource-overbooked environments which is the focus of this work. Moreover, virtual machines in the cloud are migrated from one host machine to another one in a data center and between data centers because of reasons such as consolidation, load balancing, thermal effects, and noisy neighbors. Therefore, to mitigate the performance interference between collocated VMs, the resource usage profiles of VMs must be examined and its neighboring VMs should be determined based upon this information for VM placement decisions in the cloud. Additionally, VM profiling should continue at run-time because of the fact that workloads might change dynamically.

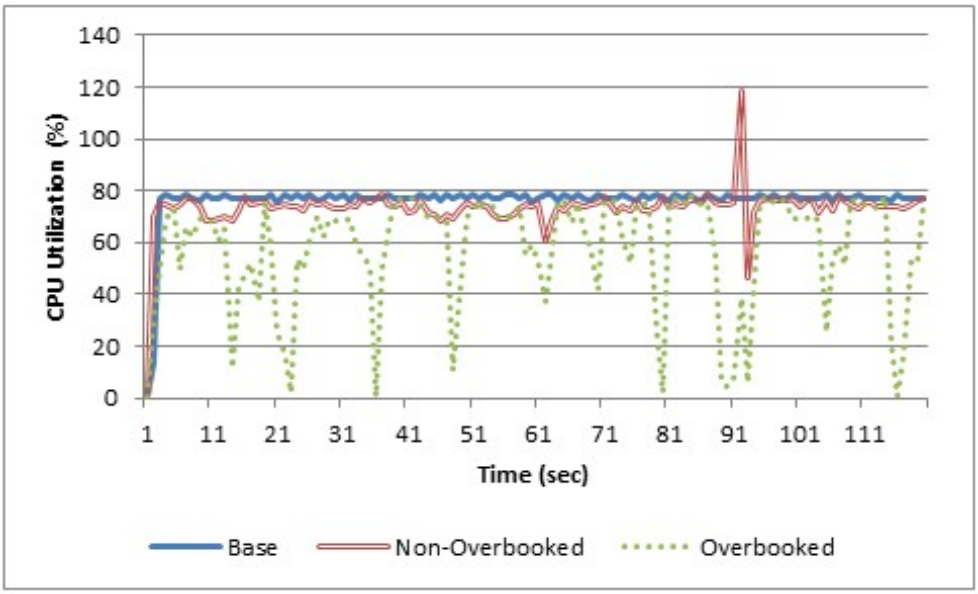


Figure 24: Comparison of Web Server CPU Utilization in Base, Non-Overbooked, and Overbooked Environments

IV.3.2 Overview of the iSensitive Approach

In this chapter we present a model predictive approach to address performance interference issues at runtime. To realize such an approach, the first step requires obtaining real-world historical resource usage data for VMs as well as host machines in the cloud so as to generate a system performance interference model. High quality and fine-grained historical data is crucial to gain a sense of the resource usage patterns and how they changed over time. Based on this analysis, we then create an accurate model of the system that can be used at runtime. In the second step, which is an online step, the learned model is then used to make decisions on VM placement that will minimize the performance interference in the resulting deployment. Our approach is generic enough and can be applied by cloud service providers in their data centers for their expected workloads and hardware platform.

Figure 25 shows the algorithmic design and building blocks of our framework called iSensitive that adopts the solution approach described above. As shown, iSensitive comprises two distinct modules: (1) Interference Model Learning Module (*offline*), and (2) Interference Model Execution and Monitoring Module (*online*). The Interference Model

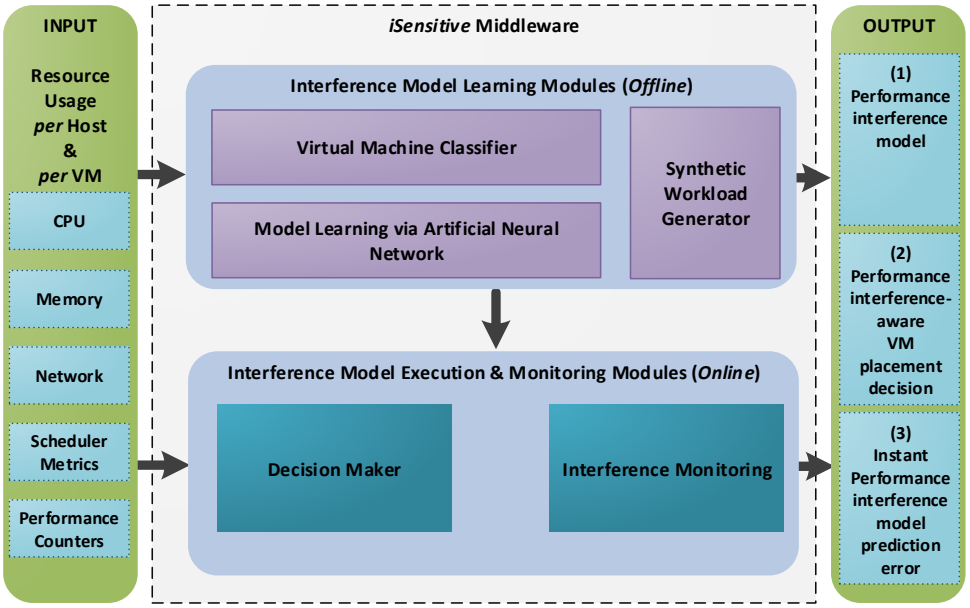


Figure 25: Conceptual Design of iSensitive Illustrating Input, Output, and System of Interest

Learning Module in turn comprises three main components: (1) Virtual Machine Classifier, (2) Model Learning via Artificial Neural Network, and (3) Synthetic Workload Generator. The Interference Model Execution and Monitoring Module consists of two primary components: (1) Decision Maker, and (2) Interference Monitoring.

Since resource utilization is a key indicator of performance interference as validated in Figure 24, iSensitive utilizes different resource usage metrics, such as CPU usage, memory usage, network I/O usage, internal scheduler metrics, hardware and kernel-level performance counters for VMs and physical host as input to the system. These metrics are retrieved with the help of (1) *perf,* performance analyzing tool in Linux, (2) *mpstat,* Linux command for processor related statistics, and (3) *libvirt,* toolkit to interact with the underlying virtualization system. The *virtual machine classifier* clusters VMs into similar sets of objects by employing the k-means algorithm and the silhouette method. These classes of

VMs are then used by the *artificial neural network* to extract the "best collocated VM patterns", which are those that lead to minimal performance interference on the host machines. In other words, a performance interference model of a host machine is generated.

Table 8: Benchmark Applications Utilized by iSensitive

Test Suite Name	Application Name	Application Description	Resource Intensiveness
Phoronix Test Suite	pts/build-apache	Timed Apache Compilation	Processor
Phoronix Test Suite	pts/compress-gzip	Gzip Compression	Processor
Phoronix Test Suite	pts/compress-pbzip2	Parallel BZIP2 Compression	Processor
Phoronix Test Suite	pts/espeak	eSpeak Speech Engine	Processor
Phoronix Test Suite	pts/n-queens	N-Queens	Processor
Phoronix Test Suite	pts/openssl	OpenSSL	Processor
Phoronix Test Suite	pts/tachyon	Tachyon	Processor
Phoronix Test Suite	pts/tscp	TSCP	Processor
Phoronix Test Suite	pts/stresscpu2	StressCPU2 Stress-Test	Processor
Phoronix Test Suite	pts/sample-program	Sample PI program	Processor
Phoronix Test Suite	pts/ramspeed	RAMspeed SMP	Memory
Phoronix Test Suite	pts/stream	Stream	Memory
Netperf	TCP_STREAM	TCP Stream Performance	Network
Netperf	TCP_RR	TCP Request Response	Network
Netperf	UDP_STREAM	UDP Stream Performance	Network
Httperf	TCP	Web Workload Generator	Network
Sysbench	OLTP	Database Server Performance	Disk

After the neural network is trained, the *decision maker* is employed to find the aptly suited host machine having the minimal performance interference by utilizing the trained model. *Interference monitoring* is responsible to compare the actual performance interference value and its predicted value. If the difference is greater than a threshold value, then that collocation pattern is saved for future model refinements.

Note that for our work, we have assumed that the physical host machines in the cloud data center are homogeneous and therefore a model generated for one host machine is applicable to all other physical hosts. If a data center comprises heterogeneous machine types, then performance interference models for each different host machine type must be created.

IV.3.3 Detailed System Design and Technical Approach

The details of each of the iSensitive components and their roles are explained below.

IV.3.3.1 Synthetic Workload Generator (offline phase)

Recall that to produce a predictive model of our system, our system needs to be trained using raw data and classification. Thus, our first objective is to obtain such a raw data. One of the key tools supplied by iSensitive is thus a synthetic workload generator. It is a Python-based tool that communicates with a cloud manager to instantiate, deploy, start and destroy virtual machines. It imitates the lifecycles of VMs.

To that end we have exploited the VM lifecycle events (e.g., create, destroy, migrate) and their resource configurations (e.g., number of virtual CPUs) from the Google Cluster Trace. To generate a training data set as realistic as possible to the real-world workload in the cloud data center, we mimicked the VM events data of five randomly chosen hosts from the Google Trace with ids 2790227930, 2113205802, 4550520892, 257335557, 317488481. We did not use more number of hosts for the VM lifecycle events due to the effort and scale needed in setting up experiments to generate the training sets.

Since the applications running in the VMs of the Google Trace are not described, we could only exploit the lifecycle events and their resource configurations. To produce a right mix of different types of application workload, we created a test suite using some of the popular benchmarking tools and real world applications stressing different aspects of a system as shown in Table 8. For every VM, the tool randomly picks a test from the suite and executes it. The iSensitive's monitoring tools collect the performance metrics and generate the training data set.

IV.3.3.2 Virtual Machine Classifier (offline phase)

Once the raw training data is created, the virtual machine classifier component clusters VMs based on their CPU, memory, and network usage by using *k-means clustering* [37].

k-means is an unsupervised learning algorithm that helps to classify the VMs in different classes based on their resource usage profiles. It provides good results with large datasets such as the one used in our approach. Note that we do not consider disk intensive applications in this work. Therefore, these three main resource usage metrics are utilized for profiling VMs.

To decide the best number of clusters, the *silhouette* method [75] is employed for the cluster data we utilize. The Silhouette [75] method fits well with the k-means clustered data and is hence employed in our approach to analyze the VM clusters. The higher the silhouette value is, the better the classification is. The best cluster number for our raw data set is found as 5 with a maximum mean silhouette value of 0.6560 over other cluster numbers. The resulting cluster center points found by virtual machine classifier component is shown in Table 9.

Table 9: Identified Cluster Center Points of Each Cluster

Cluster Number	CPU (%)	Memory (%)	Network IO (MBps)
Cluster 1 (C1)	12.83	44.60	1.04
Cluster 2 (C2)	199.03	27.54	9.19
Cluster 3 (C3)	76.29	42.75	1.46
Cluster 4 (C4)	128.05	17.07	234.06
Cluster 5 (C5)	104.08	36.68	121.48

IV.3.3.3 Model Learning via Artificial Neural Network (offline phase)

iSensitive relies on the historical data to model and capture the relationships between input and output parameters to discover the patterns of VM combinations and the resulting degree of performance interference. To capture the non-linear relationships between performance interference among the VMs and the large set of input factors for various classes

of VMs, we have applied back propagation-based artificial neural network (ANN) [38]. It is a supervised machine learning technique used to predict the performance interference which is otherwise difficult to estimate in our complex model.

Concretely, the ANN is trained to capture the relationships on how the different types and numbers of VMs of the same cluster found by the virtual machine classifier impact performance interference. The input parameters for the ANN are as follows:

N_1 = Total number of VMs of *Class 1*

N_2 = Total number of VMs of *Class 2*

N_3 = Total number of VMs of *Class 3*

N_4 = Total number of VMs of *Class 4*

N_5 = Total number of VMs of *Class 5*

C = CPU overbooking ratio

PIL = Performance Interference Level

The reason to choose number of VMs of each class is to capture the relationships between the different VM combinations along with host machine CPU overbooking ratio and discover the regularities in how these patterns affect the performance interference level (PIL) on a host machine.

We modeled the performance interference level as the sum of cache miss ratio, scheduler waiting time, scheduler io waiting time, and guest cpu usage percentages as follows.

PIL = *Cache Miss Ratio + Scheduler Wait Time % + Scheduler IO Wait Time % + Guest %*

The metrics in the performance interference model are some of the significant metrics capturing the contention at shared resources that might cause crucial performance degradation.

- **Cache Miss Ratio**: Represents the ratio of total system-wide last level cache (LLC)

misses to total number of retired instructions. It captures the contention occuring at the LLC cache and is a promising metric to model performance interference of memory and cache-intensive applications. The value is represented as per hundred instructions in order not to dominate the other values.

- **Scheduler Wait Time %**: Represents the waiting time incurred at scheduler's run queue which means that a VM is not able to access the physical CPU even though it is in the runnable state due to the CPU contention and causes increased latencies. Scheduler waiting time value may be very high for resource-overbooked environments. Therefore, this is also a promising metric capturing interference occurring at the scheduler.

- **Scheduler IO Wait Time %**: Represents the waiting time incurred due to the IO operations. The VM is in the idle mode while the system is waiting for an outstanding IO operation. This metrics helps to capture the contention for IO-bound applications and allows the to incorporate IO-level interference into model.

- **Guest %**: Expressed as the percentage of CPU time spent by all the VMs on the host machine. This is also an important metric to capture how busy are the CPU resources to serve the VMs and the less busy host machine with all the guests will ultimately have less contention at the CPU-level.

IV.3.3.4 Decision Maker (online)

When a VM placement request is made or if a VM must be migrated, the decision maker component is responsible to iterate over all the host machines in the cluster, run the trained ANN, and return the host machine info which will provide the lowest performance interference level. The VM can then be placed in the machine despite the cloud service provider utilizing overbooking strategies.

IV.3.3.5 Interference Monitoring (online)

When it is enabled, interference monitoring module keeps track of the error rate between actual and predicted performance interference level at run-time. Recall that iSensitive is trained offline with the historic data and utilizes the trained model for run-time predictions. However, there is always a possibility to encounter different workload patterns that were not known by the trained model. This will cause the system to incur high prediction errors. Therefore, the *interference monitoring* component is responsible for two tasks: (1) if the prediction error is greater than a configured threshold value, the actual workload pattern on the host is logged for re-training, (2) if a VM is way off from the actual cluster center points, it is also logged for re-clustering. These logged data are later used for re-training the performance interference model.

IV.3.4 iSensitive Distributed System Architecture

The iSensitive distributed system architecture that can be integrated with cloud infrastructure software, such as OpenStack, is depicted in Figure 26. As can be seen in the figure, iSensitive comprises a Virtual Machine Manager (V-Man) for each VM residing on a physical host, a Host Manager (H-Man) for each physical host, and a Cloud Manager (C-Man) to orchestrate the cluster of host machines in the cloud data center.

The V-Man is responsible for collecting resource information, such as memory utilization, for VMs. The reason to employ the V-Man inside a VM is because we were unable to retrieve some of the metrics at the host level. For example, the actual memory in use by a VM cannot be collected when the host machine is virtualized by the Xen hypervisor and accessed through the libvirt API. This is because of concerns such as reliability and various operating systems run by VMs. Therefore, the statistics which are only known by the VMâĂŹs kernel must be retrieved by an agent such as V-Man running inside the VM.

The primary goal of the H-Man is to accumulate statistics associated with each VM as well as the physical host machine and post these information to the C-Man. The CPU,

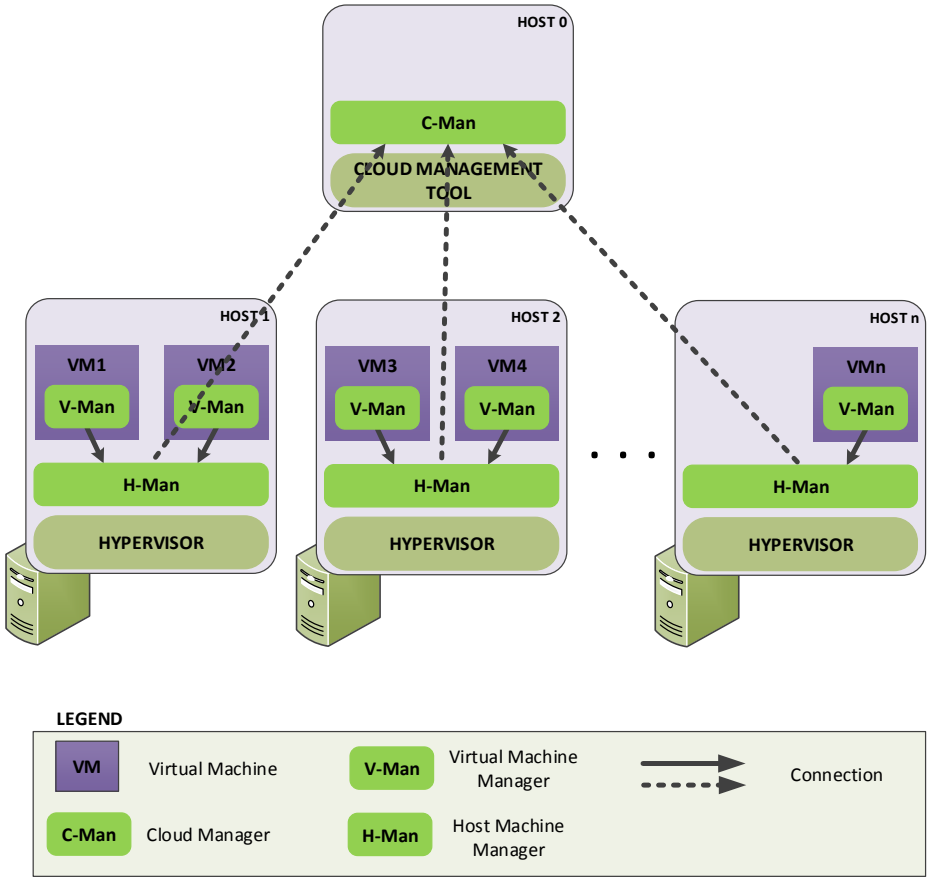


Figure 26: iSensitive System Architecture Diagram

memory, and network utilization of each VM and intrinsic scheduler parameters such as waiting time, I/O waiting time, and guest CPU percentage are some of the information being sent to the C-Man. Another critical information being sent to the C-Man is the cluster number of the VM. The cluster number of the VM is found by the minimum Euclidean distance from actual resource usage information of a VM (i.e. CPU, memory, and network utilization) to the center points of clusters found by the virtual machine classifier component.

Another challenge handled by the H-Man is handling instant spikes of resource usage. Based on the configuration, if the instant spikes cause the VM's cluster number to change for five consecutive cycles (i.e. 30 secs of interval between each cycles), then iSensitive

changes the actual cluster number with this new cluster number. The number of consecutive cycles triggering the change to the cluster number is a configuration parameter and can be tuned to another setting.

The C-Man is responsible for making decisions to place a VM on an aptly suited host machine in the cloud. All the host and VM-level statistics are collected through the collection of H-Mans and an overall view of the cloud environment is then defined by the C-Man. Eliminating hot spots, performance concern of a collocated high priority VM, server consolidation, and an antagonist, noisy neighbor VM are few of the primary reasons to migrate VMs. Whenever a VM needs to be migrated from one host machine to another one, the C-Man employs the decision maker and finds the target host machine where this VM should be hosted.

IV.4 Validating the iSensitive Approach

This section presents empirical validation of the iSensitive's performance interference-aware virtual machine placement algorithm in our private data center. We compare the performance improvements stemming from the use of iSensitive in migrating an Apache web server VM-based application iSensitive to one of the common approaches applied by data centers, e.g., first-fit bin-packing heuristics with preference for least occupied host [24], in our case, one with minimum CPU overbooking ratio. The experiments demonstrate that the iSensitive approach finds the aptly suited host machine with minimum performance interference level at the host-level and provides better performance to the applications running in the VM being migrated.

IV.4.1 Experimental Setup

The experiments were conducted in our private data center comprising a cluster of 7 homogeneous host machines. The host machines were managed by OpenNebula [56] cloud management software version 4.6.2. Table 10 provides the hardware and software

configuration while Table 11 provides the virtualization configuration of each host machine in our private data center.

Table 10: Hardware and Software Specification of the Experiment Host

Memory (GB)	32
Hard Disk (GB)	500
Processor Type	AMD Opteron 4170 HE
CPU Socket Count	2
Core Count per Socket	6
Base Speed (MHz)	2100
L1 Cache Size (KB)	128
L1 Cache Count	6
L2 Cache Size (KB)	512
L2 Cache Count	6
L2 Cache Speed (MHz)	2100
L3 Cache Size (KB)	6144
Integrated Memory Controller Speed (MHz)	2200
Operating System	Ubuntu 14.04 64-bit

Table 11: Virtualization Specification of the Experiment Host

Hypervisor	KVM
Kernel	Linux 3.13.0-24
Qemu Virtualizer	2.0.0
Guest Virtualization Mode	HVM
Guest Operating System	Ubuntu 14.04 64-bit

Figure 27 describes the setup created to validate the effectiveness of the iSensitive approach. We created 15 VMs per host each having 2 vCPUs and 512MB of memory on 5 different host machines. Each VM and host machine in this setup employs the V-Man and

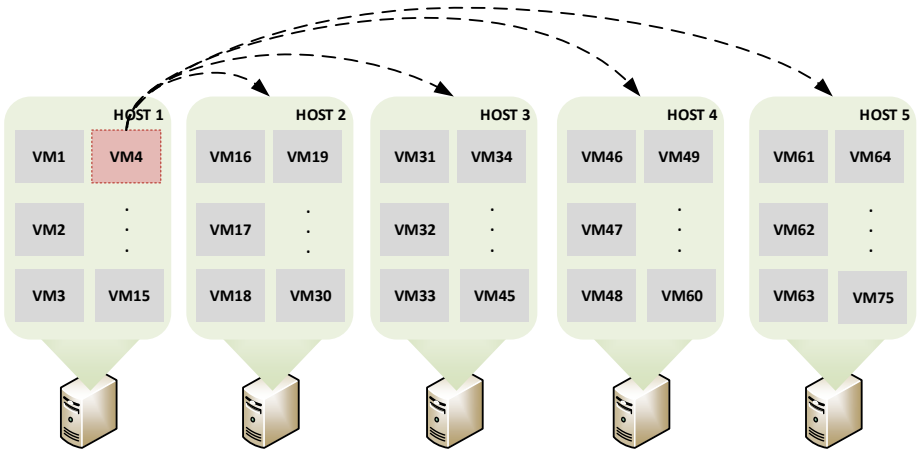


Figure 27: Experimental Setup

H-Man, respectively as explained in Section IV.3. Two more bare-metal host machines with the same configuration as in Table 10 were used. One machine was used to deploy C-Man and the other to send out client requests generated by the Apache jMeter load generator tool. jMeter sends out HTTP requests from 50 concurrent users to the Apache web server residing in the VM being tested. Virtualizing the server machine hosting client application may also have resource contention causing inconsistent test results. Hence we decided not to use a VM for hosting the client, thereby providin more robust and consistent results.

Recall from Section IV.3.3.2 that we had found 5 as the ideal number of clusters for the raw data we had generated. iSensitive tries to classify the VMs to one of these clusters. For the initialization of the experiments, randomly picked workloads from the benchmarking suite described in Section IV.3.3.1 were run on the 15 VMs on each of the five hosts. The host overbooking ratio for all 5 host machines was set to 2.5 for fairness between host machines. After the VMs were up and running, the workload of each VM was classified into one of the clusters. The resulting number of VMs per cluster number on each of the host machines was as shown in Table 12.

The experiments were conducted by selecting one of the VMs from Cluster 3 on Host

Table 12: Number of VMs in Each Cluster for Each Host

Host Name	C1	C2	C3	C4	C5
Host 1	7	1	7	0	0
Host 2	9	1	5	0	0
Host 3	6	1	7	1	0
Host 4	15	0	0	0	0
Host 5	7	1	5	0	1

1 and requesting a migration decision from iSensitive and comparing it to the migration decision using first-fit. The performance results were collected for a period of two minutes before and after migration and was found to be sufficient for analysis.

IV.4.2 Application Performance Improvement using iSensitive

We analyzed the performance of the target VM for the three cases, i.e. performance before the migration and performance after migration on the hosts decided by iSensitve and the first-fit heuristic. As mentioned earlier, the target VM was chosen from Host 1. iSensitive suggested its new location to be Host 4, whereas first-fit heuristic found it to be Host 2. For both the scenarios, we migrated the VM on these hosts and present the performance results before and after the migration.

Figure 28 presents the throughput for the three different scenarios. We observe that before the migration (see the bar for Host 1), the mean throughput of the Apache web server was 197 requests per seconds with a standard deviation value of 71, suggesting a sluggish performance. Hence, a decision to migrate is taken. After the migration to Host 2 (using first-fit), we see an improvement of 25% in throughput and the standard deviation value reduced to 47.7. Compared to first-fit's improvement, if the VM is migrated to Host 4 based as suggested by iSensitive, we see a performance improvement of 64% percent of mean throughput and a lower standard deviation 36.5 value (see the bar for Host 4). This shows that for our experimental use-case, the performance improvement due to iSensitive's

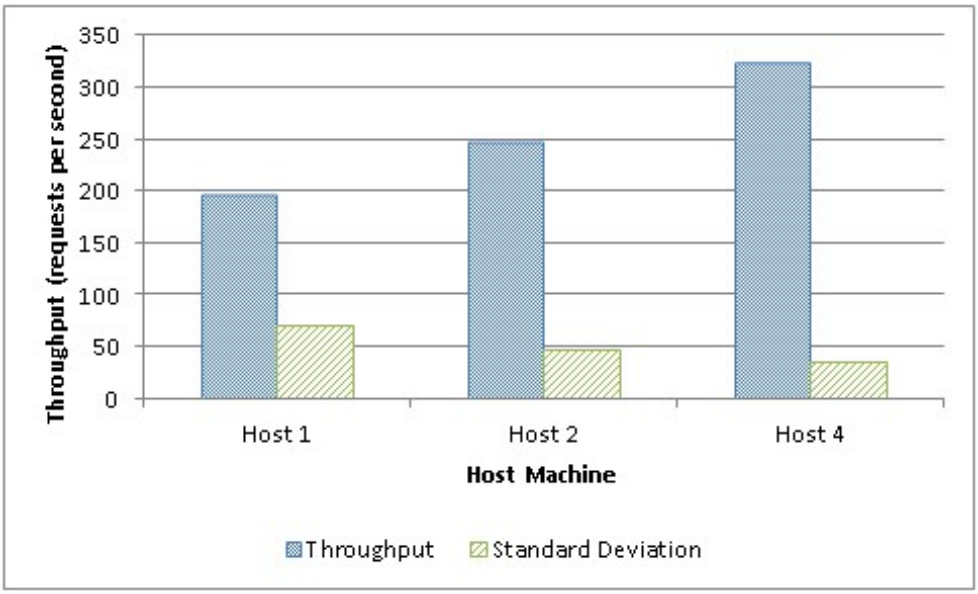


Figure 28: Comparison of Web Server Throughput on Hosts 1, 2 and 4

placement decision is 39% better for mean throughput than the commonly used strategy of first-fit heuristics.

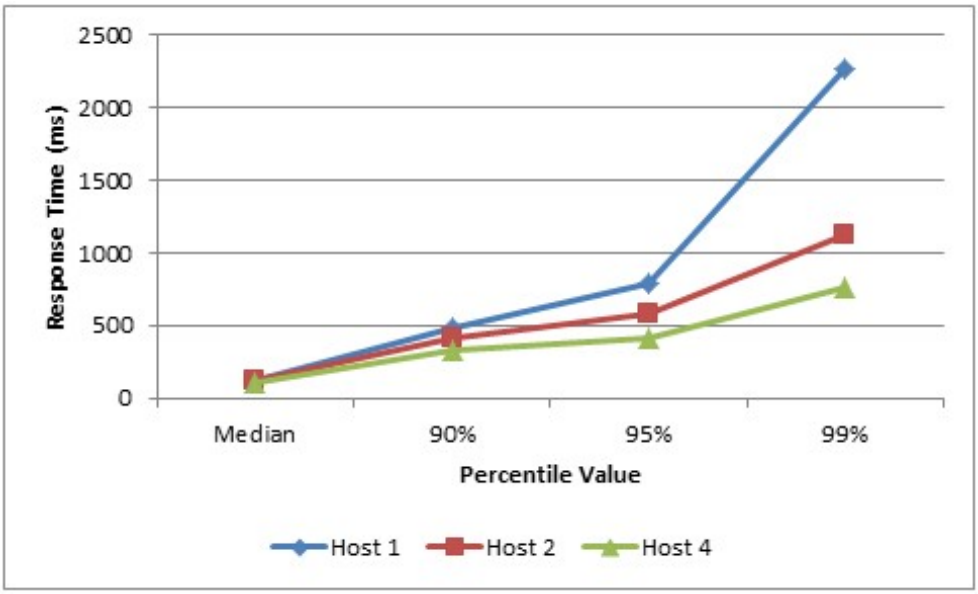


Figure 29: Comparison of Web Server Response Time Percentiles on Actual, Before, and After Migrating to a Host Machine

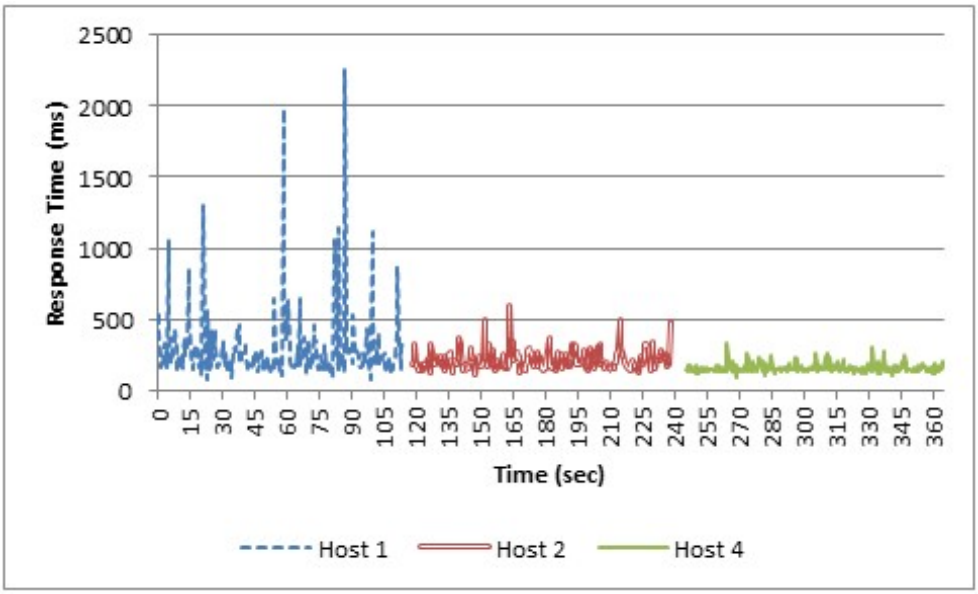


Figure 30: Comparison of Web Server Response Time Over Time on Actual, Before, and After Migrating to a Host Machine

Figures 29 and 30 depict the response time results for the same scenarios. In Figure 30, we can visually observe that the response time reduces significantly for iSensitive-suggested migration. Figure 29 confirms the same where we see performance improvements of 14.8, 21.6, 27.8, 31.9 % for mean, 90th percentile, 95th percentile and 99th percentile, respectively, for the iSensitive approach over the first-fit heuristics approach. These two different performance indicators of throughput and response time improvements confirm the efficacy of iSensitive solution.

We also measured the overhead of using iSensitive in the runtime phase. It was found to be less than 1% for V-Man and C-Man and ~5% for H-Man. These values are for 2.5 overbooking ratio where 15 V-Man connected to 1 H-Man with 1 sec heartbeat interval and 5 H-Man connected to 1 C-Man with 15 seconds heartbeat interval. This shows that iSensitive has low overhead, however, in future we would like to perform more scalability experiments.

IV.5 Conclusion

This chapter presented iSensitive, which is a performance interference-aware virtual machine placement middleware to support performance-sensitive cloud-hosted applications, such as Big Data processing of IoT sensory data. The approach comprises two steps. In the first step, which is an offline step, raw usage data of a data center is used to glean away VM workload patterns for clustering decisions and key insights into performance interference caused due to VM collocation. To that end, a clustering-based VM placement middleware was designed by utilizing back propagation artificial neural network. These insights are used in the second step, which is an online step, in finding an aptly suited host machine for VMs to minimize the performance interference effects and reduce the performance degradation in cloud-hosted IoT applications.

In this work, we have not yet considered disk-intensive applications but this will form a dimension of our future work. Disk utilization needs to be considered by the *virtual machine classifier* component in the future releases of iSensitive. Additionally, analyzing iSensitive's energy efficiency properties is left as future work.

The presented work and the algorithmic structure of iSensitive is generic enough to be used by cloud service providers for their platforms. They will need to learn the models based on historical data observed in their environment. Moreover, the building blocks of the iSensitive distributed system architecture can seamlessly integrate with cloud infrastructure software.

CHAPTER V

IPLACE: AN INTELLIGENT AND TUNABLE POWER- AND PERFORMANCE-AWARE VIRTUAL MACHINE PLACEMENT TECHNIQUE FOR CLOUD-BASED REAL-TIME APPLICATIONS

V.1 Motivation

Cloud data centers are massive-scale farms of networked servers and other resource types, such as storage, that are used to host different kinds of services simultaneously from multiple different customers. Due to the use of commodity hardware for the resources, failures are common within data centers that can cause some disruptions in the hosted services. Another major factor that can cause disruptions in data centers stems from the massive power requirements of the data centers both to operate the hardware as well as for cooling. Power outages due to excess demand can result in substantial disruptions to the data center. For instance, several prominent cloud service providers (CSPs) reported days-long partial or complete outages of their cloud services platform.[1] As an example of the adverse impact this can cause, in 2012 an intrinsic outage in Amadeus airline reservation system's data center triggered long lines and delays at many airports worldwide. Power outages are also shown to have an adverse impact on environment because they produce diesel exhaust.

Reducing energy consumption in data centers is thus an important criteria to reduce the chances of outages. This issue is particularly important considering an increasing trend towards hosting applications with soft real-time requirements in the cloud [30, 77], which cannot sustain significant service disruptions. For these applications, performance concerns, such as response time and service availability, are vital requirements and hence disruptions in data centers is often not desirable.

[1] A recent incident is reported at `www.datacenterknowledge.com/archives/2012/07/10/major-outage-salesforce-com/`.

One promising approach to maintaining availability and performance requirements of real-time applications after partial disruptions within the same data center is via live migration [28] of virtual machines (VMs).[2] VM migrations help in hardware maintenance, fault-tolerance and load-balancing. However, live migration may incur significant cost in terms of substantial network usage particularly when multiple simultaneous VM migrations are active at any given time thereby adding to the energy consumption. One key reason for the increased network usage is that existing approaches that use live VM migrations often tend to ignore the placement issues for the backup VMs, which in turn leads to unwanted usage of network and other resources thereby causing increased energy consumption.

For example, cloud-based high availability and performance solutions such as Remus [31], Paratus [34], and Kemari [78], require capturing the entire executions of the VM and transferring them to the backup machine as swiftly and seamlessly as possible. While these solutions are much desirable for maintaining application performance and availability, they tend to shift the responsibility of choosing the backup VMs to the cloud user. A solution that will relieve the cloud user of these responsibilities and automate the choice of backup VMs is desirable. In prior work [10, 11] we addressed this limitation in the Remus high availability solution by providing a backup VM placement mechanism that was based on simple bin packing heuristics. However, this work did not consider energy conservation as a criteria.

Another technique used by CSPs to improve resource utilization, reduce energy consumption, and thereby saving on energy bills is to employ "Resource Overbooking" [1, 15, 19]. Even in the case of resource overbooking, the placement of VMs on aptly suited host machines where SLA is not violated is crucial. We have observed that even idle VMs that are overbooked on a host machine might affect the performance of applications running

[2]Live migrations may be feasible across data centers but will incur additional and unpredictable networking delays, which may not be suitable for real-time applications.

in other collocated VMs on that host because of performance interference between collocated VMs [20, 64, 79] when resources are overbooked. Thus, the need for effective VM placement is a key requirement.

V.1.1 Challenges

In summary, energy conservation in data centers is increasingly becoming the focus of CSPs who are seeking ways to save on energy bills, reduce the chances of outages, and reduce adverse impact on the environment. Reducing energy consumption would imply shutting down large portions of the data center and employing resource overbooking. However, a naive approach to conserving energy may lead to applications not meeting their performance requirements, which is not acceptable to real time applications hosted in the cloud. Techniques that support both performance and availability in the cloud must continue to work. As we have seen above, a common theme that pervades these requirements is the need for effective VM placement in the data center. A common practice for VM placement decisions at the hypervisor level is bin packing heuristics such as first-fit, best-fit, and next-fit. However, these bin packing techniques do not consider power concerns of the CSPs nor performance requirements of applications.

V.1.2 Solution Approach

To address these objectives, this chapter presents *iPlace,* which is an intelligent and tunable power- and performance-aware virtual machine placement technique that is realized as cloud infrastructure middleware. The key contributions of iPlace include:

- An intelligent tunable power- and performance-aware virtual machine placement strategy in virtualized environments that satisfies soft real-time application QoS. The novelty of our VM placement approach stems from its use of a two-stage neural network which predicts (1) CPU usage at the first level and (2) uses the predicted CPU

usage at the first level to predict the power consumption and performance of a host machine at the second level. Section V.3 delves into the details of this contribution.

- It analyzes how energy consumption of data centers can be reduced while performance of soft real-time applications are ensured by employing iPlace. Section V.4 presents results of our empirical studies.

V.2 Related Work

This section explores prior work that employ schemes like live migration and server consolidation techniques that aim to address one or more of the performance, availability and energy consumption issues in cloud data centers.

Akiyama et. al propose MiyakoDori [9] which employs "memory reusing" technique to reduce the amount of memory transferred thereby reducing unnecessary energy consumption during the live migration. When a virtual machine monitor (VMM) initiates a live migration command, MiyakoDori retains the memory image of the VM on the source node. Identical memory pages are not transferred; only the manipulated memory content is transferred when that VM is migrated back to the original node. MiyakoDori saves substantial amount of memory between migrations thereby reducing energy consumption. This related work considers identical memory pages to reduce the energy consumption during live migrations whereas our work focuses on both reducing energy consumption and guaranteeing application performance. It is feasible that our work can leverage MiyakoDori in the live migration process.

Deshpande et al. address the problem of migrating several collocated VMs simultaneously [32]. In a data center, it is highly likely that collocated VMs might have the same operating system, similar software, and libraries installed on it. Therefore, the basic idea in the paper is transferring identical contents across the collocated VMs only once. Our work is once again complementary to this approach since we focus on finding an aptly suited

host machine for a VM. Thus, it is possible for our approach to leverage this related work for additional benefits.

The work closest to ours is by Hirofuchi et al. [40, 41] who propose an energy-efficient VM consolidation technique for optimizing VM locations to achieve energy savings while guaranteeing performance. In this work, post-copy live migration is utilized as opposed to pre-copy live migration since post-copy migration reacts to sudden load changes more quickly than pre-copy. Data center servers are categorized as shared and dedicated servers. Shared servers host the idle VMs while dedicated servers host CPU-intensive VMs. Shared servers take advantage of extra memory to host many idle VMs. The technique utilized in their paper is to migrate as many idle machines into shared servers as possible from dedicated servers and finally switch-off the dedicated servers in which no more VMs are left. Our work also comes under the purview of consolidation algorithms. In contrast to this work, our work does not differentiate between shared and dedicated servers, which reduces the complexity of our technique.

Berral et al. [16] propose a framework that provides intelligent dynamic consolidation of VMs in which deadline-sensitive applications are executing. A machine learning-based technique is employed to reduce the energy consumption while meeting SLA requirements for high performance computing (HPC) environments where applications have deadline constraints. This work differs from our work in that their work targets a HPC environment, which are more controlled and where exclusive access to resources is granted is targeted, whereas we primarily target public cloud environments.

Piao [70] proposes a VM placement and migration approach to optimize the heavy data transfer over the network. Due to the nature of network- and data-intensive workloads, applications hosted on various VMs often communicate with each other frequently over the network which in turn adversely affects the application performance and network overhead. Moreover, it might lead to network congestion and unexpected network latency. Therefore, migrating these kinds of applications within a close proximity of their counterparts reduces

the traffic on the network and ultimately optimizes the performance. The work in that paper differs from our work in that it attempts migrating highly coupled VMs to closeby locations. In contrast, we target compute-intensive applications and discover aptly suited host machine for their VMs with respect to power and performance. As future work, we will consider accounting for network usage as suggested in this prior work.

Khosravi et al. [47] have taken into account carbon footprint rate and power usage effectiveness (PUE) for designing VM placement strategy in data centers. The VM placement problem is considered as a bin packing problem with ($datacenter \times cluster \times host$) placement options. The authors propose Energy and Carbon-Efficient (ECE) VM placement algorithm based on best-fit heuristic to find a solution to the problem, and evaluate the algorithm using simulation. A difference with our work is that we have evaluated our solution inside a cloud data center and applied machine learning to account for a large set of factors affecting power and performance which are difficult to model in the system. In this related work, the authors considered power consumption as a function of CPU frequency, whereas we have taken several factors including memory, network, overbooking rate etc. into account for predicting performance and power.

Dong et al. [33] propose a VM placement scheme to reduce both the number of physical machines and network elements in a data center to reduce overall energy consumption. The optimization of physical servers is considered as a bin packing algorithm, while network optimization is formulated as a quadratic assignment problem. The proposed method is a combination of hierarchical clustering and best-fit to solve the optimization problem for VM placement and is evaluated based on simulations. In contrast, our work provides an intelligent placement algorithm which considers VM-based host overbooking, power consumption and performance, which is evaluated in a real data center.

V.3 Virtual Machine Placement using iPlace

Figure 31 depicts the strategy of iPlace, which is our intelligent power- and performance-aware virtual machine placement algorithm. The goal of iPlace is to find an aptly suited host machine by carefully considering the energy efficiency of the data center and performance requirements of soft-real time applications running on host machines. iPlace takes power changes and performance effects to the applications running on VMs for its placement decision. A tunable parameter named *performance preference level* is provided to iPlace in advance to set the performance requirement.

To find the aptly suited host machine, a two-level artificial neural network (ANN) is employed by our VM placement middleware, which are at the core of our system design and serve as the predictor mechanism. To train the ANNs, iPlace employs the Levenberg-Marquardt back-propagation algorithm [58]. At the first level, the mean CPU usage of a host machine after a VM were to be migrated to it is predicted by running the CPU usage predictor ANN. Subsequently, this predicted CPU usage value is utilized by the second level ANN. At the second level, power consumption and mean performance of the host machine is predicted by the power and performance predictor ANN. At runtime, the middleware will consult the prediction engine and if the predicted values are acceptable, the middleware will take the decision of placing the VM on a given host.

To understand how these ANNs are used to make runtime decisions, consider the case when one of the consolidation algorithms, high availability solutions, or scheduling mechanisms would like to migrate a VM from one host machine to another one. iPlace finds the aptly suited host machine by predicting the power consumption and performance values for each host machines in the cluster as though the VM was migrated on to it. As illustrated in Figure 31, iPlace employs both CPU usage predictor and power and performance predictor sequentially by feeding their required input values.

In our current design, iPlace targets only compute-intensive applications, therefore

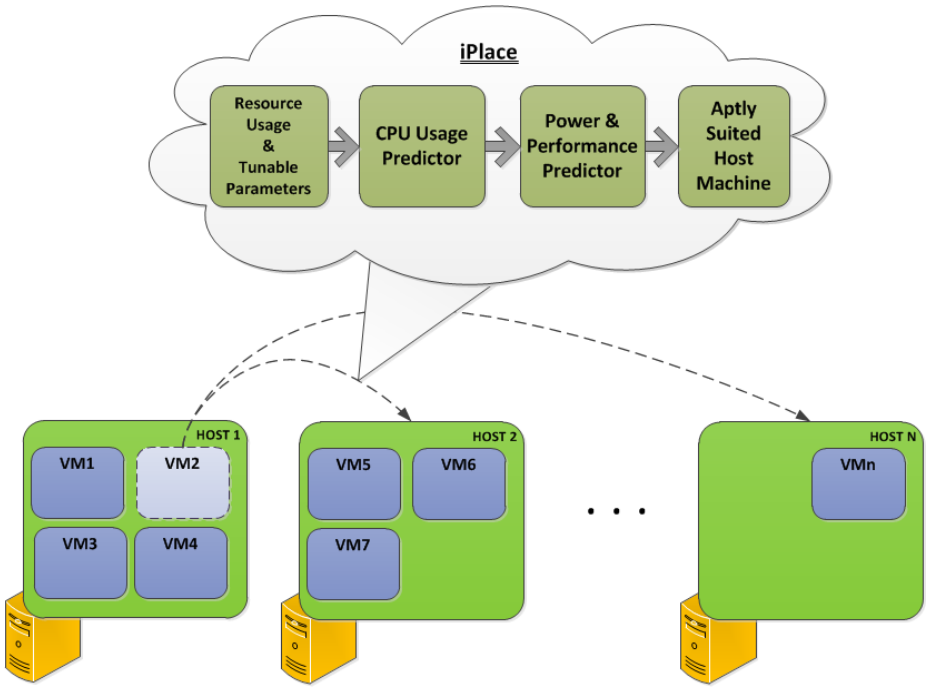


Figure 31: Illustration of iPlace's Virtual Machine Placement Strategy

$1/(CPUtime)$ metric was utilized in this work as the performance indicator of an application. The higher the performance value, the better the performance. Additionally, we assume that CSPs overbook their underlying cloud infrastructure to save energy costs. Details of the ANNs are described below.

V.3.1 CPU Usage Predictor

The structure of the CPU usage predictor ANN is depicted in Figure 32. The purpose of the CPU usage predictor is to estimate the amount of CPU usage of the host machine after a VM were to migrate onto it. Due to the CPU contention in over utilized virtualized environments, the mean CPU usage of a host machine might not increase by the same amount of CPU usage currently been illustrated by the VM being migrated. Thus, a simple subtraction on one machine and addition on another machine does not work. Therefore,

iPlace employs a CPU usage predictor ANN for this prediction so that the power consumption and performance of the host machine could be determined effectively by knowing the CPU usage of the host machine.

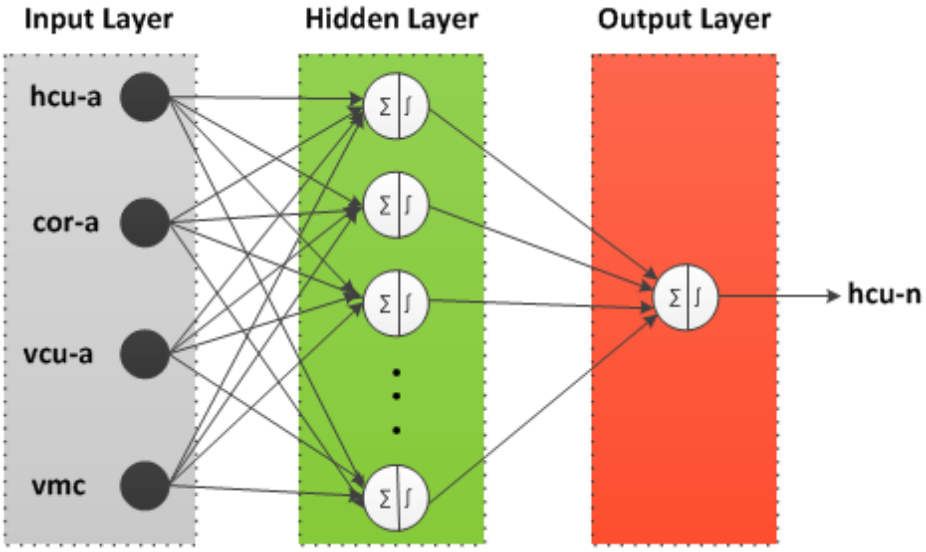


Figure 32: Structure of the CPU Usage Predictor ANN

The topology of the CPU usage predictor ANN is shown below.

Input Layer : $hcu-a, cor-a, vcu-a, vmc$

Hidden Layer : 9 neurons

Activation Function (in hidden layer)

: Tangent Sigmoid

Output Layer : $hcu-n$

Transfer Function (in output layer)

: Pure Linear

where

$hcu-a$ = Actual mean CPU usage of the host machine before VM is migrated on it

$cor-a$ = Actual CPU overbooking ratio of the host machine before VM is migrated on it

$vcu-a$ = Actual CPU usage of the VM being migrated onto the host machine

vmc = Actual VM count on the host machine before VM is migrated on it

$hcu-n$ = CPU usage of the host machine after VM is migrated onto it

The CPU overbooking ratio and mean CPU usage of the host machine provided to this ANN are computed by Equations (V.2) and (V.4).

$$Total\ vCPU\ Requested = \sum_{i=0}^{m} vCPU_i \tag{V.1}$$

$$CPU\ Overbooking\ Ratio = \frac{Total\ vCPU\ Requested}{Total\ pCPU\ Cores} \tag{V.2}$$

$$Total\ CPU\ Usage = \sum_{i=0}^{m} vmCPUUsage_i \tag{V.3}$$

$$Host\ Mean\ CPU\ Usage = \frac{Total\ CPU\ Usage}{m} \tag{V.4}$$

where

$Total\ vCPU\ Requested$: Total number of virtual CPU cores requested on a host machine

m : Total number of the guest VMs on a host machine

$vCPU$: Number of virtual CPU cores of a VM

$Total\ pCPU\ Cores$: Total number of physical CPU cores of a host machine

The number of neurons in the hidden layer is determined based on experimentation by trying different numbers and examining the system results. The reliability and accuracy of ANNs employed by iPlace is examined by carefully looking into the mean squared error (MSE) and regression (R) values. The MSE value provides average squared difference between input and output whereas the R value describes how the input of the system is correlated with its output. The best performance of the CPU usage predictor ANN was produced with 9 neurons in the hidden layer, and MSE of 0.00044 and R of 0.99. As shown in Figure 33, these MSE and R values clearly indicate that CPU usage predictor precisely estimates the host machine's CPU usage.

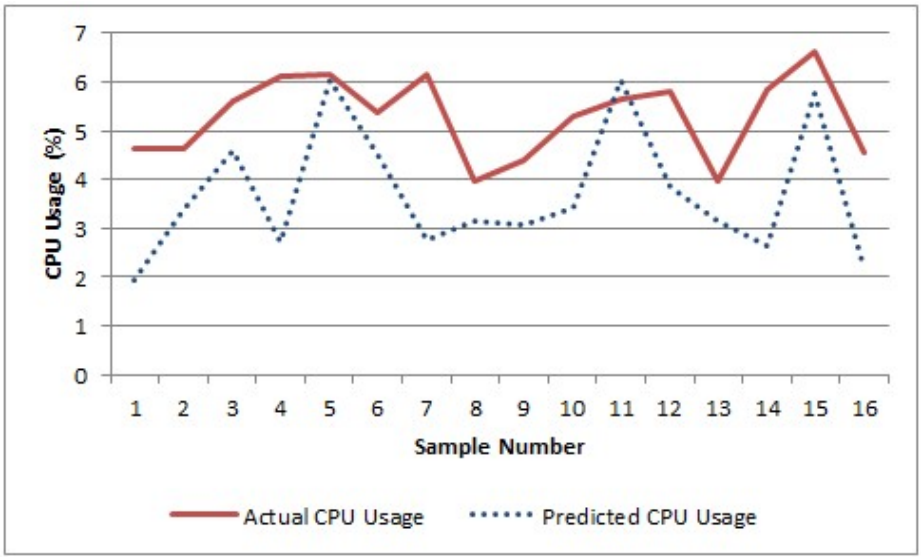


Figure 33: Comparison of Actual and Predicted CPU Usage of Host Machine

V.3.2 Power and Performance Predictor

The structure of the power and performance predictor ANN is depicted in Figure 34. The output of the CPU usage predictor ANN is provided as input to the power and performance predictor ANN. The purpose of power and performance predictor is to predict power consumption and performance of the host machine if the VM were to migrate to it.

The topology of power and performance predictor ANN is detailed below.

Input Layer : cu, cor, mor, vmc

Hidden Layer : 12 neurons

Activation Function (in hidden layer)

: Tangent Sigmoid

Output Layer : $Pow, Perf$

Transfer Function (in output layer)

: Pure Linear

where

$cu =$ Mean CPU usage of the host machine

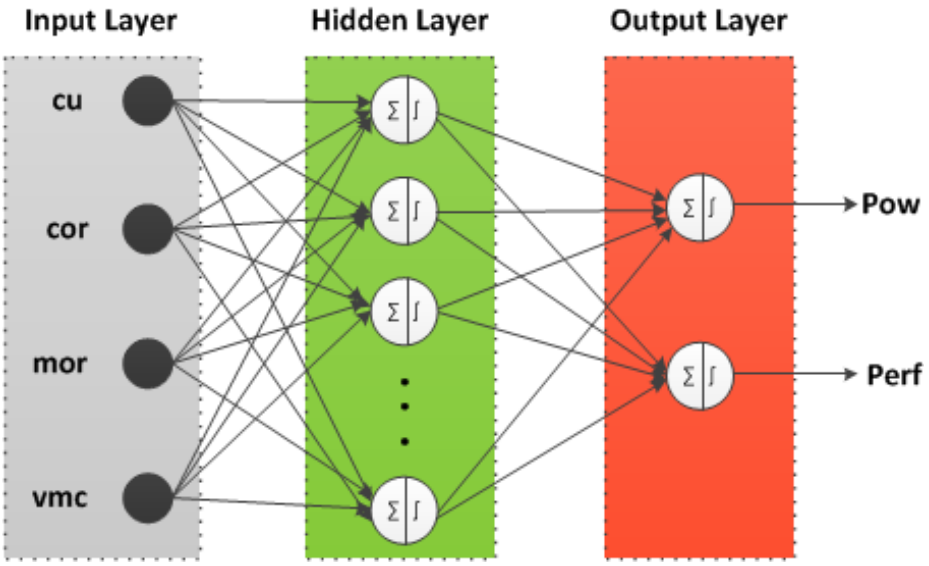


Figure 34: Structure of the Power and Performance Predictor Artificial Neural Network

cor = CPU overbooking ratio of the host machine

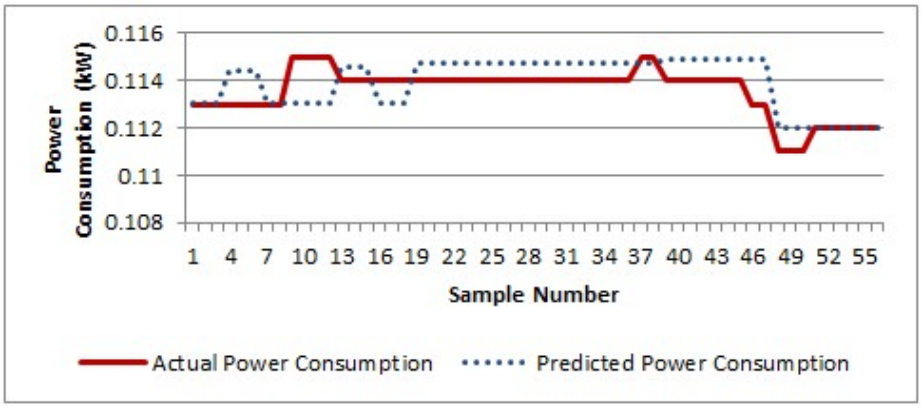
mor = Memory overbooking ratio of the host machine

vmc = VM count on the host machine

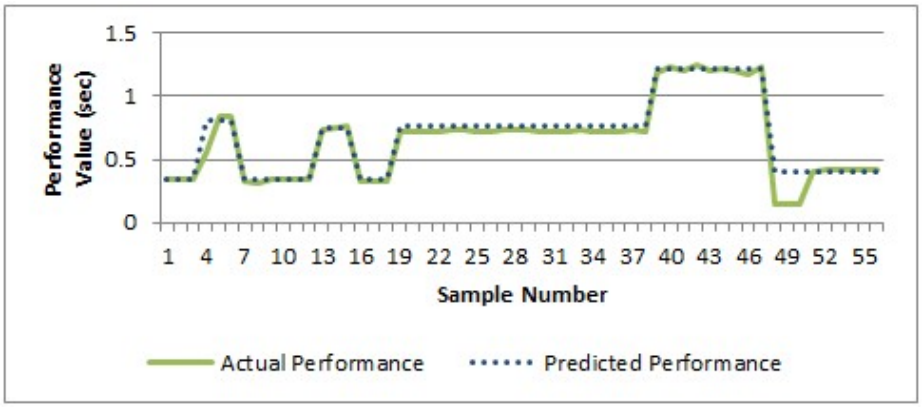
Pow = Power consumption of the host machine

$Perf$ = Performance value of the host machine (*i.e.* mean performance of all the guest VMs running on the host machine)

The best performance of the power and performance predictor ANN was produced with 12 neurons in the hidden layer, MSE of 0.008, and R of 0.97. These MSE and R values



(a) Power Consumption



(b) Performance

Figure 35: Comparison of Actual and Predicted Power Consumption and Performance Value Results of Host Machine

clearly show that the power and performance predictor ANN precisely estimates the host machine's power consumption and performance. Figure 35 depicts the comparison of actual power consumption and performance values of host machine along with the predicted values of power and performance predictor.

By carefully observing the data generated by our simulation software, we determined the mean performance value (μ) as 1.75 and the standard deviation (σ) as 1.17. These values are the assumed indicators for performance requirement of the soft real-time application and utilized to check whether the performance requirement of the soft real-time application validated by the Equation (V.5). This performance indicator is assured on the host machine where it will be migrated with best effort. α in Equation (V.5) is basically the performance

preference level parameter passed by the system user. The tighter performance requirement might cause iPlace not to be able to find any host machine.

$$Pr = \mu + \alpha * \sigma \tag{V.5}$$

where

Pr : Performance requirement of the VM

μ : Mean performance value
computed by looking to the
values in the cluster

α : Performance preference level

σ : Standard deviation
value in the cluster
computed by looking to the
values in the cluster

This performance parameter along with the resource usage information is provided to iPlace. iPlace employs the CPU usage predictor ANN first and feeds the predicted CPU usage of the host machine to the power and performance predictor ANN then.

After receiving the CPU usage, iPlace finds all the host machines satisfying the performance requirement in Equation (V.5) by comparing these predicted performance values with the value returned from Equation (V.5) by starting from the *performance preference level*. If none of the host machines satisfies the requested *performance preference level*,

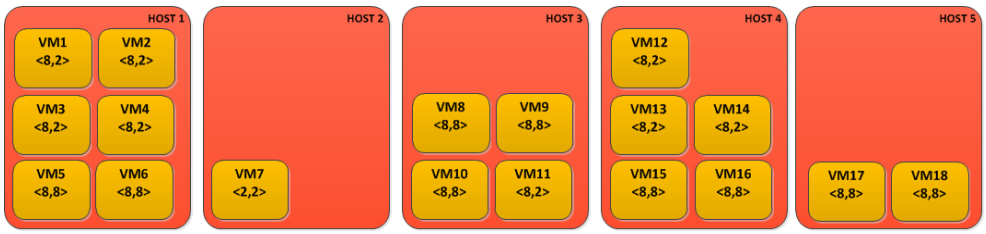


Figure 36: Initial Configuration of the Cluster Utilized in Test Cases

iPlace gradually lower the *performance preference level* by one and checks each host machine again to find another host machine that will satisfy this new performance requirement. Then, iPlace computes the power change on the each host machine satisfying the performance requirement to see how much power will increase. Finally, the placement decision is made onto the host machine which satisfies the performance requirement and has the minimum power change.

V.4 Validating the iPlace Approach

The iPlace framework consists of two stage neural network. For accurate predictions, the training data set for both the stages should be as close as possible to real-world data. To achieve the same, we have used a private data center consisting of five hosts on which we have emulated a workload that is similar to that of a production data released by Google Inc. from one of their cluster's trace log [74]. Our private data center comprises a homogeneous set of machines and is managed by the OpenNebula cloud management solution version 3.2.1. Table 13 provides the configuration of each host used as a cluster node. Each host is connected to a *Watts Up? Pro* power meter, which can report consumed power with frequency of once a second and an accuracy of a tenth of a watt.

The Google cluster trace contains a dataset for about 12,000 distinct machines collected over a 29 day period in the month of May, 2011. We chose one of the host with ID 257408495 from the cluster and reproduced the workload on one of the host in our data center for one day. The VM configuration and resource usage in the dataset was normalized

Table 13: Hardware and Software specification of Cluster Nodes

Processor	2.1 GHz Opteron
Number of CPU cores	12
Memory	32 GB
Hard disk	8 TB
Operating System	Ubuntu 10.04 64-bit
Hypervisor	Xen 4.1.2
Guest virtualization mode	Para

which we scaled according to host configuration and pruned the data which did not fit the characteristics of our host.

The workload on the host was generated using our simulation software coded in Python which used OpenNebula to create and delete virtual machines. We executed *Lookbusy*, a synthetic load generator, processes to mimic the CPU and memory workloads. Our resource monitoring application was coded in C++, which runs to collect the resource usage information of the host using *libvirt* library at certain specified interval.

Matlab software is used to train and run the ANNs as well as deciding the placement decision. Additionally, a SQL server database management system is utilized to import the resource information data of each host machine and prepare the training set for ANNs.

V.4.1 Experimental Results

In this section we show the experimental results of iPlace that we have tested and evaluated by following two test cases we have defined. The initial configuration of our cluster is depicted in Figure 36. In Figure 36, the tuple under each VM name represents the resource capacity of the VM in the format of <*cpu, memory*>. Additionally, initial resource usage and overbooking ratios of each host machine in the cluster is illustrated in Table 14.

Use Case 1: In this use case, we assumed that there was an abnormal activity causing performance degradation of VM2, which is a high priority VM on Host 1. Therefore,

Table 14: Initial Resource Usage of Host Machines in the Cluster

	CPU Usage	CPU Over-booking Ratio	Memory Overbooking Ratio
HOST 1	16%	4	0.75
HOST 2	18%	0.17	0.06
HOST 3	30%	2.6	0.81
HOST 4	23%	3.3	0.69
HOST 5	1.5%	1.3	0.5

a decision was made to migrate it to another host machine in the cluster. We analyzed and compared the results of placement decision of iPlace with a first-fit heuristic of bin packing algorithm in the context of power and performance. We have tested iPlace with three different performance preference levels (i.e. α values of -1, 0, 1).

Recall that performance preference level is the tunable performance parameter passed by the system user. It is also used in the performance requirement equation V.5 with α as the parameter. Based on that performance preference level, the standard deviation is adjusted and performance requirement is either tightened or softened. As the performance preference level goes below 0, -1, -2 ..., the performance requirement of the application is softened and vice versa.

As expected, the placement decision of first-fit heuristic is to place the VM on the first host machine in which it fits. Therefore, a first-fit heuristic placed VM2 on to Host 2. iPlace decides the target host machine by observing power changes on host machines and performance effects on the applications running on the VM. Thus, the placement decision for each performance preference level presented in Table 15 for this use case might be dissimilar for any other use cases.

As shown in Table 15, iPlace decided to migrate VM2 to Host 5 at both performance preference levels of α = -1 and α= 0. iPlace assured the performance requirement of VM2

Table 15: Test Results of Use Case 1

Performance Preference Level (α)	Placement Decision
-1	HOST 5
0	HOST 5
1	NONE

on none of the host machines in the cluster for the tighter performance preference level of $\alpha = 1$.

To detail the case where performance preference level of α =0, iPlace predicted that only Host 3 and Host 5 satisfied the performance requirement in Equation (V.5). However, iPlace decided to migrate VM2 onto the Host 5 due to the prediction of lower amount of power increase by 0.0274kW on Host 5 versus 0.0281kW on Host 3.

Compared to the first-fit heuristic, iPlace could not assure the performance requirement of VM2 on Host 2 even though VM2 fits on it. Therefore, it discarded Host 2 for its placement decision for VM2.

Use Case 2: In this use case, we assumed that a decision was made to migrate all VMs residing on one of the less utilized host machines onto the rest of the host machines decided by iPlace. Host 2 was selected as the target due to having only one VM. Therefore, VM7 will be migrated and Host 2 will be shut down.

At initial run with performance preference level of $\alpha = 0$, iPlace could not find a host machine for that criteria. It determined Host 3, Host 4, and Host 5 after iterating till the performance preference level of α = -2. However, iPlace decided to migrate VM7 onto the Host 3 due to the power change concerns.

After migrating VM7 onto the Host 3, Host 2 becomes idle and started to consume 0.091kW power with no VMs running on it. Therefore, we assumed it was turned off and computed the overall power consumption of the cluster. The total power consumption of

the cluster dropped from 0.708kW to 0.614kW which saved about 13% of power consumed by the cluster.

V.5 Concluding Remarks

In this chapter we proposed iPlace, which is an intelligent tunable power- and performance-aware virtual machine placement strategy. The work was motivated by the need to conserve energy in data centers yet manage the performance and availability requirements of soft real-time applications that are increasingly being hosted in cloud data centers. To that end we have developed a two-level artificial neural network (ANN) with the stage one responsible for CPU usage prediction, and stage two responsible for power and performance prediction. The two stage ANN was designed, trained and employed to forecast the host machine's CPU usage, power consumption, and performance. For training purposes and evaluation, we generated workloads in our private cloud that emulated data from a Google's production server. We have tested and evaluated iPlace in our private cloud and compared the results with first-fit bin-packing heuristic. The results shows that iPlace can help to save certain degrees of power consumption by satisfying variety of performance requirements. Compared to the first-fit heuristic, iPlace places VMs on host machines where application performance is assured and energy efficiency is maximized.

Since the private cloud environment where we tested and evaluated iPlace is a homogeneous environment, our test results were validated only in a homogeneous environment. However, iPlace can easily be employed in a heterogeneous environment by providing additional host machine capacity information parameters to the ANNs, such as number of cores and memory size. In this work, we targeted only the compute-intensive applications due to the performance metric we utilized. By integrating more generic application performance metrics, such as response time or throughput, iPlace can support a variety of application types in the cloud environment. These dimensions will form the basis of our future work.

The source code for iPlace is available for download at `www.dre.vanderbilt.edu/~caglarf/download/iPlace`.

CHAPTER VI

CONCLUDING REMARKS

The research conducted for this doctoral dissertation was motivated by the need for innovative solutions to address dynamic resource management and energy conservation challenges in cloud data centers, specifically focusing on the virtualization, cloud management, application and service delivery layers. To that end this dissertation presents a set of novel solutions each of which addresses a specific set of challenges. Each of these solutions provides a systematic and scientific approach that a cloud service provider can implement in their data centers to address energy consumption and resource utilization challenges.

The research contributions in this dissertation and the specific set of challenges they address can be classified along four dimensions as follows:

1. **Autonomous and Dynamic Reconfiguration of Hypervisor Scheduler.** The challenges in the area of autonomous and dynamic scheduler reconfiguration are addressed by *Engineering the Performance of Xen Hypervisor via Autonomous and Dynamic Scheduler Reconfiguration* middleware, called iTune. iTune automatically reloads the optimum configuration based on the changing workload on the host machine. iTune comprises three phases named Discoverer, Optimizer, and Observer and employs machine learning algorithms. iTune provides options to mark the VMs into one of the four latency sensitivity categories (i.e. LS-1, LS-2, LS-3, and NLS). This allows iTune to assure performance requirements associated with these latency sensitivity levels.

   Although iTune has currently been demonstrated in the context of the Xen credit scheduler, testing the approach and comparing the results for other systems software are left as a future work. Additionally, the number of regions of operation (i.e.,

clusters) for training set is based on a specific workload we generated. Hence, we suggest that CSPs first apply iTune to their historic workloads.

2. **Resource-Overbooking to Support Soft Real-time Applications.** The challenges in the area of dynamic resource-overbooking are addressed by *Intelligent Resource-Overbooking to Support Soft Real-time Applications in the Cloud*, called iOverbook. iOverbook determines the CPU and memory overbooking ratios for each host machine in the cloud by predicting their future resource usage demands, and considering the QoS requirements of soft real-time applications. The benefits and efficacy of iOverbook were evaluated in the context of resource utilization and energy efficiency in the data centers by utilizing Google's cluster trace log data. Our future work for iOverbook will investigate effective filtering of outliers and using confidence intervals.

3. **Performance Interference Effects on Application Performance.** The challenges in the area of performance interference effects on application performance are addressed by *An Intelligent Performance Interference-aware Virtual Machine Migration Approach*, called iSensitive. The proposed research investigated the performance interference effects on application performance and creating a model to make intelligent virtual machine placement.

   Presently, iSensitive does not consider disk-intensive applications. Disk-intensive applications need to be considered for the systems utilizing local disk. Additionally, analyzing energy efficiency and performance interference properties should also be considered.

4. **Power- and Performance-Aware Virtual Machine Placement.** The challenges in the area of power- and performance-aware virtual machine placement are addressed by *An Intelligent and Tunable Power- and Performance-Aware Virtual Machine Placement Technique for Cloud-based Real-time Applications*, called iPlace.

iPlace employs two-level artificial neural networks to predict a host machine's CPU usage at the first level and power consumption and performance of the host machine at the second level.

In its current form, iPlace targets only the compute-intensive applications due to the metrics utilized. Supporting variety of application types in the cloud environment should be considered.

In our research we have not investigated the combined benefits of the techniques we have developed. Thus, another dimension of future work will require a framework that holistically integrates all these techniques were different trade-offs may be necessary.

VI.1 Summary of Research Contributions and Technical Insights

This dissertation provides a systematic and reproducible approach to address each of the challenges that arises in cloud data centers. Cloud service providers can readily implement our solutions using the described scientific process behind each solution. The research contributions and the long lasting insights they provide are summarized below:

- **Autonomous and dynamic scheduler configuration**

  1. Deeper insights into how the Xen hypervisor's internal scheduler parameters and performance are correlated with each other.
  2. Architecture for an intelligent, autonomous and self-tuning middleware called iTune to optimize Xen scheduler configuration.
  3. Options to mark VMs into one of the categories: (1) latency-sensitive level-1 (LS-1), (2) latency-sensitive level-2 (LS-2), (3) latency-sensitive level-3 (LS-3), (4) non-latency sensitive (NLS). Additionally, assure to deliver the associated performance requirement of applications in these categories.
  4. Empirical results on how iTune can find the optimum configuration parameters for Xen credit scheduler and self tune it based on varying workload at run-time.

- **Intelligent overbooking strategy for cloud management layer**

    1. Architecture for autonomous and performance-aware overbooking strategy for each host machine in heterogeneous virtualized datacenters that satisfies soft real-time application QoS, called iOverbook.
    2. Thorough experimental validations including an analysis of how resource utilization levels can be improved and power consumption reduced in the cloud data centers by utilizing iOverbook.

- **Performance interference-aware virtual machine migration approach**

    1. Online performance interference monitoring and VM placement technique made available in a middleware called iSensitive.
    2. An analysis of performance differences in Base, Non-Overbooked, and Overbooked environments, and comparison of the impact of resource utilization on performance interference.
    3. Empirical validation of how performance interference-aware virtual machine placement techniques assure better application performance.

- **Tunable power- and performance-aware virtual machine placement algorithm**

    1. An intelligent tunable power- and performance-aware virtual machine placement strategy in virtualized environments that satisfies soft real-time application QoS.
    2. Design of a two-stage neural network which predicts the power consumption and performance of a host machine at the second level.
    3. Empirical results on how energy consumption of data centers can be reduced while performance of soft real-time applications are ensured by employing iPlace.

VI.2 Summary of Publications

1. **Faruk Caglar**, Shekhar, S., Gokhale, A., Satabdi Basu, Tazrian Rafi, Kinnebrew, J.S., & Biswas, G. Cloud-hosted Simulation-as-a-Service for High School STEM Education. The Elsevier Journal of Simulation Modelling Practice and Theory, 2015.

2. **Faruk Caglar**, Shashank Shekhar, and Aniruddha Gokhale, iTune: Engineering the Performance of Xen Hypervisor via Autonomous and Dynamic Scheduler Reconfiguration, IEEE Transactions on Services Computing (TSC), 2015 (*in submission*).

3. Kyoungho An, Shashank Shekhar, **Faruk Caglar**, Aniruddha Gokhale, and Shivakumar Sastry, "A Cloud Middleware for Assuring Performance and High Availability of Soft Real-time Applications", The Elsevier Journal of Systems Architecture (JSA): Embedded Systems Design, 2014.

4. Shashank Shekhar, Hamza Abdelaziz, Michael Walker, **Faruk Caglar**, Aniruddha Gokhale, and Xenofon Koutsoukos, "A Simulation as a Service Cloud Middleware", The Springer Journal of Annals of Telecommunications, 2014 (*in submission*).

5. Shashank Shekhar, Shweta Khare, **Faruk Caglar**, Aniruddha Gokhale, Douglas Schmidt, and Xenofon Koutsoukos, "Middleware-enabled DDDAS", Book Chapter in Springer, 2014 (*in submission*)

6. Basu, S., Kinnebrew, J.S., Shekhar, S., **Faruk Caglar**, Rafi, T.H., Biswas, G., and Gokhale, A. Collaborative Problem Solving using a Cloud-based Infrastructure to Support High school STEM Education. In Proceedings of the 122nd ASEE Annual Conference and Exposition. Seattle, WA, USA, 2015

7. **Faruk Caglar**, Shashank Shekhar, and Aniruddha Gokhale. iPlace: An Intelligent and Tunable Power- and Performance-Aware Virtual Machine Placement Technique

for Cloud-based Real-time Applications, 17th IEEE Symposium on Object/Component/Service-oriented Real-time Distributed Computing (ISORC), Reno, Nevada, USA, June 10, 2014

8. **Faruk Caglar** and Aniruddha Gokhale, iOverbook: Intelligent Resource-Overbooking to Support Soft Real-time Applications in the Cloud, 7th International Conference on Cloud Computing (IEEECloud), Alaska, USA, June 27, 2014

9. **Faruk Caglar**, Shashank Shekhar, Aniruddha Gokhale and Xenefon Koutsoukos, An Intelligent Performance Interference-aware Virtual Machine Migration Approach, 22nd Annual IEEE International Conference on High Performance Computing (HiPC 2015), Bangalore, India, Dec 16-19, 2015 (*in submission*)

10. **Faruk Caglar**, Shashank Shekhar and Aniruddha Gokhale, Performance Interference-aware Virtual Machine Placement Strategy for Supporting Soft Real-time Applications in the Cloud, 3rd International Workshop on Real-time and Distributed Computing in Emerging Applications (REACTION 2014), Rome, Italy, Dec 2, 2014.

11. **Faruk Caglar**, Kyoungho An, Shashank Shekhar, and Aniruddha Gokhale. Model-driven Performance Estimation, Deployment, and Resource Management for Cloud-hosted Services, The 13th Workshop on Domain-Specific Modeling at ACM Sigplan SPLASH Conference, Indianapolis, Indiana, USA, October 27, 2013

12. Shashank Shekhar, **Faruk Caglar**, Kyoungho An, Takayuki Kuroda, Aniruddha Gokhale and Swapna Gokhale A Model-driven Approach for Price/Performance Tradeoffs in Cloud-based MapReduce Application Deployment, MDHPCL Workshop at ACM/IEEE 16th International Conference on Model Driven Engineering Languages and Systems, Miami, Florida USA, September 29, 2013

13. Dukeman, Anton, **Faruk Caglar**, Shashank Shekhar, John Kinnebrew, Gautam Biswas, Doug Fisher, and Aniruddha Gokhale. Teaching Computational Thinking Skills in

C3STEM with Traffic Simulation, In Proceedings of the 1st IEEE International Conference on Human Factors in Computing & Informatics. Maribor, Slovenia. July 1 -3, 2013

14. **Faruk Caglar**, Kyoungho An, Shashank Shekhar, and Aniruddha Gokhale. Model-driven Performance Estimation, Deployment, and Resource Management for Cloud-hosted Services, Poster Presentation at ACM Sigplan SPLASH Conference, Indianapolis, Indiana, USA, October 27, 2013

15. Shekhar, S., **Faruk Caglar**, Dukeman, A., Hou, L., Gokhale, A., Kinnebrew, J.S., & Biswas, G. (2014). A Collaborative K-12 STEM Education Framework Using Traffic Flow as a Real-world Challenge Problem. In the 121st American Society for Engineering Education Annual Conference & Exposition. Indianapolis, IN.

16. Dukeman, A., Shekhar, S., **Faruk Caglar**, Gokhale, A., Biswas, G., & Kinnebrew, J.S. (2014). Analyzing Students' Computational Models as they Learn in STEM Disciplines. In the 121st American Society for Engineering Education Annual Conference & Exposition. Indianapolis, IN

17. Shashank Shekhar, **Faruk Caglar**, Kyoungho An, Takayuki Kuroda, Aniruddha Gokhale and Swapna Gokhale, "A Model-driven Approach for Price/Performance Tradeoffs in Cloud-based MapReduce Application Deployment", MODELS 2013 workshop on Model-Driven Engineering for High Performance and CLoud computing (MDHPCL 2013), Miami FL, Sep 29, 2013.

18. **Faruk Caglar**, Shashank Shekhar, Kyoungho An and Aniruddha Gokhale, "WiP Abstract: Intelligent Power- and Performance-aware Tradeoffs for Multicore Servers in Cloud Data Centers", Work-in-Progress (WiP) session at 4th ACM/IEEE International Conference on Cyber Physical Systems (ICCPS 2013), Philadelphia PA, Apr 9-11, 2013.

19. Kyoungho An, **Faruk Caglar**, Shashank Shekhar and Aniruddha Gokhale, "A Framework for Effective Placement of Virtual Machine Replicas for Highly Available Performance-sensitive Cloud-based Applications", RTSS 2012 workshop on Real-time and Distributed Computing in Emerging Applications (REACTION 2012), San Juan, Puerto Rico, Dec 4-7, 2012.

20. Kyoungho An, Subhav Pradhan, **Faruk Caglar** and Aniruddha Gokhale, "A Publish/Subscribe Middleware for Dependable and Real-time Resource Monitoring in the Cloud", Middleware 2012 workshop on Secure and Dependable Middleware for Cloud Monitoring and Management (SDMCMM 2012), Montreal, Quebec, Canada, Dec 3-7, 2012.

21. **Faruk Caglar**, Kyoungho An, Aniruddha Gokhale and Tihamer Levendovszky, "Transitioning to the Cloud? A Model-driven Analysis and Automated Deployment Capability for Cloud Services", MODELS 2012 workshop on Model-Driven Engineering for High Performance and CLoud computing (MDHPCL 2012), Innsbruck, Austria, Sep 30 - Oct 5, 2012.

REFERENCES

[1] Best practices for oversubscription of cpu, memory and storage in vsphere virtual environments. https://software.dell.com, September 2013.

[2] Standard performance evaluation corporation. http://www.spec.org/power_ssj2008/, October 2013.

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

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

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

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

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

[8] Tim Abels, Puneet Dhawan, and Balasubramanian Chandrasekaran. An overview of xen virtualization. *Dell Power Solutions*, 8:109–111, 2005.

[9] Soramichi Akiyama, Takahiro Hirofuchi, Ryousei Takano, and Shinichi Honiden. Miyakodori: A memory reusing mechanism for dynamic vm consolidation. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 606–613. IEEE, 2012.

[10] Kyoungho An, Faruk Caglar, Shashank Shekhar, and Aniruddha Gokhale. Automated Placement of Virtual Machine Replicas to Support Reliable Distributed Real-time and Embedded Systems in the Cloud. In *International Workshop on Real-time and Distributed Computing in Emerging Applications (REACTION), 33rd IEEE Real-time Systems Symposium (RTSS '12)*, San Juan, Puerto Rico, USA, December 2012. IEEE.

[11] Kyoungho An, Shashank Shekhar, Faruk Caglar, Aniruddha Gokhale, and Shivakumar Sastry. A Cloud Middleware for Assuring Performance and High Availability of Soft Real-time Applications. *Elsevier Journal of Systems Architecture (JSA)*, 60(9):757–769, December 2014.

[12] Gail Axelrod. 47 stats you need to know about the google apps ecosystem. http://blog.bettercloud.com/google-apps-stats/, 2013.

[13] 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*, volume 37, pages 164–177. ACM, 2003.

[14] Sean Kenneth Barker and Prashant Shenoy. Empirical evaluation of latency-sensitive application performance in the cloud. In *Proceedings of the first annual ACM SIGMM conference on Multimedia systems*, pages 35–46. ACM, 2010.

[15] Salman A Baset, Long Wang, and Chunqiang Tang. Towards an understanding of oversubscription in cloud. In *Proceedings of the 2nd USENIX conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*, pages 7–7. USENIX Association, 2012.

[16] Josep Ll Berral, Íñigo Goiri, Ramón Nou, Ferran Julià, Jordi Guitart, Ricard Gavaldà, and Jordi Torres. Towards energy-aware scheduling in data centers using machine learning. In *Proceedings of the 1st International Conference on energy-Efficient Computing and Networking*, pages 215–224. ACM, 2010.

[17] Sergey Blagodurov, Daniel Gmach, Martin Arlitt, Yuan Chen, Chris Hyser, and Alexandra 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*, pages 277–285. IEEE, 2013.

[18] Kate Brandt. Better buildings challenge expands to take on data centers. `http://www.whitehouse.gov/blog/2014/09/30/better-buildings-challenge-expands-take-data-centers`, 2014.

[19] Faruk Caglar and Aniruddha 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)*, page 10, Anchorage, AL, USA, June 2014. IEEE.

[20] Faruk Caglar, Shashank Shekhar, and Aniruddha Gokhale. A Performance Interference-aware Virtual Machine Placement Strategy for Supporting Soft Real-time Applications in the Cloud. Technical Report ISIS-13-105, Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN, USA, 2013.

[21] Faruk Caglar, Shashank Shekhar, and Aniruddha 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, June 2014. IEEE.

[22] Faruk Caglar, Shashank Shekhar, and Aniruddha Gokhale. Performance Interference-aware Virtual Machine Placement Strategy for Supporting Soft Real-time Applications in the Cloud. In *3rd International Workshop on Real-time and Distributed Computing in Emerging Applications (REACTION), IEEE RTSS 2014*, page 6, Rome, Italy, December 2014. IEEE.

[23] Faruk Caglar, Shashank Shekhar, and Aniruddha Gokhale. iTune: Engineering the Performance of Xen Hypervisor via Autonomous and Dynamic Scheduler Reconfiguration. *Submitted to the IEEE Transactions on Services Computing (TSC)*, 2015.

[24] Ricardo Stegh Camati, Alcides Calsavara, and Luiz Lima Jr. Solving the virtual machine placement problem as a multiple multidimensional knapsack problem. *ICN 2014*, page 264, 2014.

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

[26] Ludmila Cherkasova and Rob Gardner. Measuring cpu overhead for i/o processing in the xen virtual machine monitor. In *USENIX Annual Technical Conference, General Track*, volume 50, 2005.

[27] Ron C Chiang and H Howie Huang. Tracon: Interference-aware scheduling for data-intensive applications in virtualized environments. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, page 47. ACM, 2011.

[28] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live migration of virtual machines. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*, pages 273–286. USENIX Association, 2005.

[29] Antonio Corradi, Mario Fanelli, and Luca Foschini. Vm consolidation: a real case based on openstack cloud. *Future Generation Computer Systems*, 32:118–127, 2014.

[30] Antonio Corradi, Luca Foschini, Javier Povedano-Molina, and Juan M Lopez-Soler. Dds-enabled cloud management support for fast task offloading. In *Computers and Communications (ISCC), 2012 IEEE Symposium on*, pages 000067–000074. IEEE, 2012.

[31] B. Cully, G. Lefebvre, D. Meyer, M. Feeley, N. Hutchinson, and A. Warfield. Remus: High availability via asynchronous virtual machine replication. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, pages 161–174. USENIX Association, 2008.

[32] Umesh Deshpande, Xiaoshuang Wang, and Kartik Gopalan. Live gang migration of virtual machines. In *Proceedings of the 20th international symposium on High*

performance distributed computing, pages 135–146. ACM, 2011.

[33] Jiankang Dong, Xing Jin, Hongbo Wang, Yangyang Li, Peng Zhang, and Shiduan Cheng. Energy-saving virtual machine placement in cloud data centers. In *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*, pages 618–624. IEEE, 2013.

[34] Yuyang Du and Hongliang Yu. Paratus: Instantaneous failover via virtual machine replication. In *Grid and Cooperative Computing, 2009. GCC'09. Eighth International Conference on*, pages 307–312. IEEE, 2009.

[35] Richard E Edwards, Joshua New, and Lynne E Parker. Predicting future hourly residential electrical consumption: A machine learning case study. *Energy and Buildings*, 49:591–603, 2012.

[36] Diwaker Gupta, Rob Gardner, and Ludmila Cherkasova. Xenmon: Qos monitoring and performance profiling tool. *Hewlett-Packard Labs, Tech. Rep. HPL-2005-187*, 2005.

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

[38] Robert Hecht-Nielsen. Theory of the backpropagation neural network. In *Neural Networks, 1989. IJCNN., International Joint Conference on*, pages 593–605. IEEE, 1989.

[39] Jin Heo and Lenin Singaravelu. Deploying Extremely Latency-sensitive Applications in vSphere 5.5: Performance Study. Technical report, VMware, Inc., 2013. `www.vmware.com/files/pdf/techpaper/latency-sensitive-perf-vsphere55.pdf`.

[40] Takahiro Hirofuchi, Hidemoto Nakada, Satoshi Itoh, and Satoshi Sekiguchi. Making vm consolidation more energy-efficient by postcopy live migration. In *CLOUD COMPUTING 2011, The Second International Conference on Cloud Computing, GRIDs, and Virtualization*, pages 195–204, 2011.

[41] Takahiro Hirofuchi, Hidemoto Nakada, Satoshi Itoh, and Satoshi Sekiguchi. Reactive consolidation of virtual machines enabled by postcopy live migration. In *Proceedings of the 5th international workshop on Virtualization technologies in distributed computing*, pages 11–18. ACM, 2011.

[42] Rachel Householder, Scott Arnold, and Robert Green. On cloud-based oversubscription. *arXiv preprint arXiv:1402.4758*, 2014.

[43] Jinho Hwang, Sai Zeng, FY Wu, and Timothy Wood. A component-based performance comparison of four hypervisors. In *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*, pages 269–276. IEEE, 2013.

[44] MT Imam, SF Miskhat, RM Rahman, and M Ashraful Amin. Neural network and regression based processor load prediction for efficient scaling of grid and cloud resources. In *Computer and Information Technology (ICCIT), 2011 14th International Conference on*, pages 333–338. IEEE, 2011.

[45] Intel Corporation Inc. What happens in an internet minute? `http://www.intel.com/content/www/us/en/communications/internet-minute-infographic.html`, 2014.

[46] Melanie Kambadur, Tipp Moseley, Rick Hank, and Martha A Kim. Measuring interference between live datacenter applications. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 51. IEEE Computer Society Press, 2012.

[47] Atefeh Khosravi, Saurabh Kumar Garg, and Rajkumar Buyya. Energy and carbon-efficient placement of virtual machines in distributed cloud data centers. In *Euro-Par 2013 Parallel Processing*, pages 317–328. Springer, 2013.

[48] Avi Kivity, Yaniv Kamay, Dor Laor, Uri Lublin, and Anthony Liguori. kvm: the Linux virtual machine monitor. In *Proceedings of the Linux Symposium*, volume 1, pages 225–230, 2007.

[49] Younggyun Koh, Rob Knauerhase, Paul Brett, Mic Bowman, Zhihua Wen, and Calton Pu. An analysis of performance interference effects in virtual environments. In *Performance Analysis of Systems & Software, 2007. ISPASS 2007. IEEE International Symposium on*, pages 200–209. IEEE, 2007.

[50] Jonathan Koomey. Growth in data center electricity use 2005 to 2010. *A report by Analytical Press, completed at the request of The New York Times*, 2011.

[51] Ben Kröse, Ben Krose, Patrick van der Smagt, and Patrick Smagt. An introduction to neural networks. 1993.

[52] Min Lee, AS Krishnakumar, Parameshwaran Krishnan, Navjot Singh, and Shalini Yajnik. Xentune: Detecting xen scheduling bottlenecks for media applications. In *Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE*, pages 1–6. IEEE, 2010.

[53] 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.

[54] Zitao Liu and Sangyeun Cho. Characterizing machines and workloads on a google cluster. In *Parallel Processing Workshops (ICPPW), 2012 41st International Conference on*, pages 397–403. IEEE, 2012.

[55] Amiya K Maji, Subrata Mitra, Bowen Zhou, Saurabh Bagchi, and Akshat Verma. Mitigating interference in cloud services by middleware reconfiguration. In *Proceedings of the 15th International Middleware Conference*, pages 277–288. ACM, 2014.

[56] Dejan Milojičić, Ignacio M Llorente, and Ruben S Montero. Opennebula: A cloud management tool. *IEEE Internet Computing*, 15(2):0011–14, 2011.

[57] Asit K Mishra, Joseph L Hellerstein, Walfredo Cirne, and Chita R Das. Towards characterizing cloud backend workloads: insights from google compute clusters. *ACM SIGMETRICS Performance Evaluation Review*, 37(4):34–41, 2010.

[58] Jorge J. More. The Levenberg-Marquardt Algorithm: Implementation and Theory. In G.A. Watson, editor, *Numerical Analysis*, volume 630 of *Lecture Notes in Mathematics*, pages 105–116. Springer Berlin Heidelberg, 1978.

[59] Ismael Solis Moreno and Jie Xu. Customer-aware resource overallocation to improve energy efficiency in realtime cloud computing data centers. In *Service-Oriented Computing and Applications (SOCA), 2011 IEEE International Conference on*, pages 1–8. IEEE, 2011.

[60] Ismael Solis Moreno and Jie Xu. Neural network-based overallocation for improved energy-efficiency in real-time cloud environments. In *Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC), 2012 IEEE 15th International Symposium on*, pages 119–126. IEEE, 2012.

[61] Ismael Solis Moreno, Renyu Yang, Jie Xu, and Tianyu Wo. Improved energy-efficiency in cloud datacenters with interference-aware virtual machine placement. In *Autonomous Decentralized Systems (ISADS), 2013 IEEE Eleventh International Symposium on*, pages 1–8. IEEE, 2013.

[62] David Mosberger and Tai Jin. httperfâĂŤa tool for measuring web server performance. *ACM SIGMETRICS Performance Evaluation Review*, 26(3):31–37, 1998.

[63] Al Muller and Seburn Wilson. Virtualization with vmware esx server. 2005.

[64] Ripal Nathuji, Aman Kansal, and Alireza Ghaffarkhah. Q-clouds: managing performance interference effects for qos-aware clouds. In *Proceedings of the 5th European conference on Computer systems*, pages 237–250. ACM, 2010.

[65] Dejan Novaković, Nedeljko Vasić, Stanko Novaković, Dejan Kostić, and Ricardo Bianchini. Deepdive: Transparently identifying and managing performance interference in virtualized environments. In *Proceedings of the 2013 USENIX Conference*

on Annual Technical Conference, USENIX ATC'13, pages 219–230, Berkeley, CA, USA, 2013. USENIX Association.

[66] Tomas Olsson. Evaluating machine learning for predicting next-day hot water production of a heat pump. In *4th International Conference on Power Engineering, Energy and Electrical Drives - IEEE POWERENG 2013*, May 2013.

[67] Cian O'Luanaigh. Cern data center passes 100 petabytes. `http://home.web.cern.ch/about/updates/2013/02/cern-data-centre-passes-100-petabytes`, 2013.

[68] Pradeep Padala, Kai-Yuan Hou, Kang G Shin, Xiaoyun Zhu, Mustafa Uysal, Zhikui Wang, Sharad Singhal, and Arif Merchant. Automated control of multiple virtualized resources. In *Proceedings of the 4th ACM European conference on Computer systems*, pages 13–26. ACM, 2009.

[69] Compuware/Research In Action White Paper. The hidden costs of managing applications in the cloud. Dec 2012.

[70] Jing Tai Piao and Jun Yan. A network-aware virtual machine placement and migration approach in cloud computing. In *Grid and Cooperative Computing (GCC), 2010 9th International Conference on*, pages 87–92. IEEE, 2010.

[71] Xing Pu, Ling Liu, Yiduo Mei, Sankaran Sivathanu, Younggyun Koh, and Calton Pu. Understanding performance interference of i/o workload in virtualized cloud environments. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pages 51–58. IEEE, 2010.

[72] Navaneeth Rameshan, Leandro Navarro, Enric Monte, and Vladimir Vlassov. Stay-away, protecting sensitive applications from performance interference. In *Proceedings of the 15th International Middleware Conference*, pages 301–312. ACM, 2014.

[73] Charles Reiss, Alexey Tumanov, Gregory R Ganger, Randy H Katz, and Michael A Kozuch. Towards understanding heterogeneous clouds at scale: Google trace analysis. *Intel Science and Technology Center for Cloud Computing, Tech. Rep*, 2012.

[74] Charles Reiss, John Wilkes, and Joseph Hellerstein. Google cluster-usage traces: format+ schema. *Google Inc., White Paper*, 2011.

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

[76] Nilabja Roy, Abhishek Dubey, and Aniruddha Gokhale. Efficient Autoscaling in the Cloud using Predictive Models for Workload Forecasting. In *4th IEEE International Conference on Cloud Computing (Cloud 2011)*, pages 500–507, Washington, DC, July 2011. IEEE.

[77] Teresa M Takai. Cloud computing strategy. Technical report, DTIC Document, 2012.

[78] Y. Tamura, K. Sato, S. Kihara, and S. Moriai. Kemari: Virtual machine synchronization for fault tolerance. *In USENIX 2008 Poster Session*, 2008.

[79] Omesh Tickoo, Ravi Iyer, Ramesh Illikkal, and Don Newell. Modeling virtual machine performance: challenges and approaches. *ACM SIGMETRICS Performance Evaluation Review*, 37(3):55–60, 2010.

[80] L. Tomas and J. Tordsson. An autonomic approach to risk-aware data center overbooking. *Cloud Computing, IEEE Transactions on*, 2(3):292–305, July 2014.

[81] Luis Tomás and Johan Tordsson. Improving cloud infrastructure utilization through overbooking. In *Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference*, CAC '13, pages 5:1–5:10, New York, NY, USA, 2013. ACM.

[82] US Environmental Protection Agency. Report to Congress on Server and Data Center Energy Efficiency: Public Law 109-431, 2007.

[83] Carl A Waldspurger. Memory resource management in vmware esx server. *ACM SIGOPS Operating Systems Review*, 36(SI):181–194, 2002.

[84] Dili Wu and Aniruddha 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)*, pages 89–98, Bengaluru, India, December 2013. IEEE.

[85] Sisu Xi, Justin Wilson, Chenyang Lu, and Christopher Gill. RT-Xen: Towards Real-time Hypervisor Scheduling in Xen. In *Proceedings of the International Conference on Embedded Software (EMSOFT)*, pages 39–48. ACM, 2011.

[86] Cong Xu, Sahan Gamage, Pawan N Rao, Ardalan Kangarlou, Ramana Rao Kompella, and Dongyan 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*, pages 3–14. ACM, 2012.

[87] Yunjing Xu, Michael Bailey, Brian Noble, and Farnam Jahanian. Small is better: avoiding latency traps in virtualized data centers. In *Proceedings of the 4th annual Symposium on Cloud Computing*, page 7. ACM, 2013.

[88] Lingfang Zeng, Yang Wang, Wei Shi, and Dan 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*, pages 267–274, Dec 2013.

[89] Xiao Zhang, Eric Tune, Robert Hagmann, Rohit Jnagal, Vrigo Gokhale, and John

Wilkes. Cpi2: Cpu performance isolation for shared compute clusters. In *Proceedings of the 8th ACM European Conference on Computer Systems*, EuroSys '13, pages 379–391, New York, NY, USA, 2013. ACM.

[90] Qian Zhu and Teresa Tung. A performance interference model for managing consolidated workloads in qos-aware clouds. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 170–179. IEEE, 2012.