Thermal Balancing by Autonomous Virtual Machine Migration

Habtetsega Moges Bekele

Thesis submitted for the degree of
Master in Network and System Administration
30 credits

Department of Informatics
Faculty of mathematics and natural sciences

UNIVERSITY OF OSLO

Spring 2018

Thermal Balancing by Autonomous Virtual Machine Migration

Habtetsega Moges Bekele

Abstract

With the ever growing server density and infrastructures encompassed in data centers, heat generation and power dissipation has increased drastically. This has brought a serious impact on system performance and reliability. Therefore, it has demanded an efficient thermal management, where thermal balancing is one of the various techniques.

This thesis presents a design and implementation of an autonomous virtual machine (VM) migration in order to attain thermal balance in server cluster hosting the VMs.

Two algorithms, based on server temperature readings, have been implemented to attain thermal balance in server cluster. The first algorithm employs temperature readings of all the servers in the cluster, to make autonomous VM migration by choosing the coolest server. Whereas, the second algorithm requires only the temperature reading of the server where the VMs are hosted on and the VMs learn their environment by trial and error to do the autonomous migration. The results showed that both algorithms were able to maintain thermal balance in the server cluster by applying the autonomous migration.

Contents

List of Figures

List of Tables

Acknowledgment

I would like to start by thanking God for all the blessings in my life.

My supervisors, **Hårek Haugerud** and **Anis Yazidi** has been remarkable mentors throughout the journey of this thesis. I am very grateful for the guidance, ideas, comments and feedbacks you have given me. It was a pleasure working with you!

I am also thankful to the **University of Oslo (UiO)** and **Oslo Metropolitan University (OsloMet)** for admitting me to this study program. The past two years has been demanding and also rewarding.

Thanks to my teachers from both schools, and my classmates who I have taken this study program with. You all were wonderful and kind.

My families and friend has been very supportive that I look up to whenever I needed motivation. Thank you for the love!

Lastly, and most importantly, I would like to thank my beloved husband for being the support and the inspiration I needed. I Love You!

Habtetsega Moges
May 22-2018
Oslo-Norway

Part I

Introduction

Chapter 1

Introduction

The rapid growth of internet and internet service providers brought the concept of *cloud services*, where internet users do not run and store data on their own, but access them from servers of the service providers. To reduce costly infrastructures and maintenance of running their own computing networks and servers, companies are moving their applications to cloud services. The cloud means someone else maintain applications and services in a remote locations called *data centers* [23].

Data centers are energy consuming facilities which are estimated to account for 1.4% of the global electricity consumption [43]. This data centers are scattered worldwide and are increasing in number and size. They range from few servers in a room to tens of thousands of servers and other accompanying hardwares.

Those huge data centers have introduced another level of power consumption, that came from the generated heat. Data centers are dependent on coal and other coal-intensive sources and it is considered to be the reason for the growing carbon footprint, which lead to a dramatic effect on the environment [2]. A large data center is an industrial-scale operation using as much electricity as a small town [21]. Mainly due to technological advances such as cloud computing and internet services, the growth in electricity consumption raised serious concern for data centers [6].

Virtualization is one of the fundamental technologies that made cloud computing work. Virtualization softwares allow one physical server to run several individual computing environments. Cloud providers have large data centers full of servers to run their cloud services, but they cannot allocate a single server to each customer. Thus, they virtualize the server, enabling each client to work with a separate "virtual" instance of the same software [7].

The growth of virtualization has added another important dimension to data center infrastructure management. Virtualization of hardware resources has been used as a method of power saving in data centers. It provides optimum hardware utilization, because it can run multiple operating systems on one physical server. The idea here is to combine many small machines called *virtual machines (VMs)* into one large physical server, so that the processor can be used more effectively.

A VM is a software simulation of a hardware platform that provides a virtual operating environment for guest operating systems [39]. The VMs interact with software emulation of the hardware they are running on, called *hypervisor* also known as a *virtual machine monitor (VMM).* Hypervisor is a software program that runs on a physical host hardware platform and supervises the execution of the guest operating systems on the VMs.

Techniques such as *Server consolidation* [45], *VM migration* [16] and *Load Balancing* has been proposed to increase the utilization of servers and create an opportunity to reduce the number of physical servers, saving huge amount of energy and reducing carbon footprints.

1.1 Motivation

Although various techniques have been used largely, power consumption of modern processors and server density in data centers is still growing and thermal management in data centers is being an important factor. The main objective of thermal management is to improve reliability and prevent system failure [31]. A historically useful generalization supported by Arrhenius' equation is that for every $10°C$ increase in temperature, the failure rate of a system doubles [22].

There are different mechanisms for thermal management, such as throttling, dynamic voltage scaling and thermal balancing. *Throttling* is an intentional lowering of the speed that is available over an internet connection, whereas *Dynamic voltage scaling* involves increasing or decreasing the voltage used in a component depending upon circumstances. Yet, the focus of this thesis, *thermal balancing*, is an approach to balance the temperatures of different servers through dynamic workload distribution in a server cluster. Thermal balancing has three main advantages to be used in data centers [18]. Firstly, it can effectively remove imbalanced heat in a server cluster which reduces the cooling cost for the overall data center. Secondly, it can prevent server overheating without causing any performance downsides. Finally, it can be applied to heterogeneous server clusters.

So far, different experiments and implementations have been carried out on virtualization, server consolidation, thermal balancing and more, to solve the issues from the generated heat and power dissipation in data centers. To mention a few; implementing deep learning algorithm at Google resulted in 40% cooling bill reduction within their data centers [17]. Another technique was from Microsoft where they submerged data centers to keep them cool and to harvest energy from the sea by building an underwater data centers [8]. A study done on a real data center in [32] shows that reducing the temperature difference from $10^{\circ}C$ to $2^{\circ}C$ resulted in a 25% reduction of the total energy cost associated with the cooling infrastructure.

In similar manner, this thesis aims to achieve thermal balance in a server cluster using virtualization technology and autonomous VM migration.

1.2 Problem Statement

How to achieve thermal balance on virtualized server cluster by autonomous migration of Virtual Machines hosted on them, based on the servers temperature readings.

In order to achieve the goal of the project the following questions are going to be addressed:

- How to remaster a custom VM based on light weight Linux distribution with a given workload?
- How to equip VMs with autonomous decision making capability?
- How to monitor and visualize VMs activities in server cluster in real time?

In this thesis, small light weight Linux distribution VMs will be used to handle a given workload. By constantly doing autonomous choices of whether to move and where to move, their aim will be to maintain a thermal balance in a server cluster.

Autonomous VM Migration - The provisioned VMs running on the servers are going to make their own decision based on implemented algorithms, to migrate, in a decentralized manner without co-ordination with one another.

Temperature Readings - The temperature data gathered from the server's temperature sensors is going to be used by the VMs to make a decision, whether to move and where to move.

Chapter 2

Background

This Chapter presents technologies, tools and concepts that are going to be exercised in this thesis. In addition, it will cover some of the early researches and related works done on the field.

2.1 Virtualization

Virtualization concepts were first introduced to reduce hardware cost and improve productivity [10]. The common way of virtualization gives an opportunity to run multiple operating systems on the same physical system by providing virtualized hardware to a guest operating system. This is done by a software layer called *Virtual Machine Monitor (VMM)* or *hypervisor*. In virtualization technology the VMM is used to manage both virtual machines (VMs) and the host machine by controlling the hardware resources and providing guest operating systems. The VMs share resources of the host system.

Hardware virtualization, which is one type of virtualization, is used in many computing areas [10, 28], such as server consolidation, VM migration and load balancing.

2.2 Libvirt

Libvirt is a library for managing virtualization solutions. It can be used to manage KVM, ZEN and many more virtualization technologies. Libvirt provides both GUI and command line tools for management. Among which, virtual machine manger is a well known GUI tool and virsh is a command line tool to manage (start, stop, pause, migrate, etc.) VMs as intended.

2.3 Kernel-based Virtual Machine

Kernel-based Virtual Machine(KVM) is a full virtualization solution and requires a processor with hardware virtualization support. KVM supports many operating systems including, Linux, BSD and Windows.

The KVM virtualization architecture can be seen on Figure 2.1.

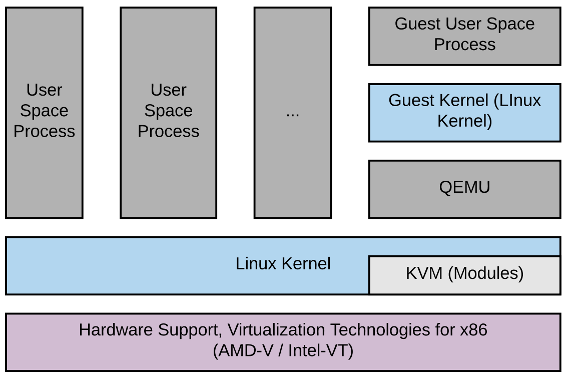
User Space Process
User Space Process
...
Guest User Space Process
Guest Kernel (LInux Kernel)
QEMU
Linux Kernel
KVM (Modules)
Hardware Support, Virtualization Technologies for x86 (AMD-V / Intel-VT)

Figure 2.1: KVM Virtualization Architecture

2.4 Live Migration

Live virtual machine migration is a type of VM migration where a running VM is migrated from one server to the other and it is important for dynamic resource management in data centers [29].

There are three VM migrations including *live migration*. Where live migration means that migration is done without any disturbance to the hosting servers. The other one,*cold migration* involves shutting VM off from one physical server and starting it on the other machine. and lastly, *warm migration*, allows the VM running on the first host server is suspended while copying the RAM and CPU registers to the second host server servers and then able to continue from the second host server.

Live migration can be done in two ways, *sequential* and *parallel*. In sequential live migration the migration of VMs happen one after the other whereas in parallel live migration VMs are migrated simultaneously [46, 48].

2.4.1 Live Migration Techniques

While doing a live migration; memory, storage and network connectivity of the virtual machine are transferred from the host server to the destination server. Memory migration of a VM in general have three phases[13, 26, 37].

Push Phase, *Stop and Copy Phase* and *Pull Phase*.

There are two techniques in order to migrate the VMs memory state, *Pre-copy memory migration* and *Post-copy memory migration* [5].

2.4.1.1 Pre-copy memory migration

In pre-copy memory migration, the hypervisor copies all the memory pages from source to destination while the VM is still running on the source. For the case of "dirty" pages i.e, memory pages that change during the process, they will be re-copied by subsequent iterations. Once the number of dirty pages is relatively small, or reaches a maximum, the virtual machine is stopped on the original host, CPU and remaining dirty pages are transferred and the virtual machine is resumed at the destination host. The approach behind Pre-copy memory migration is to transfer the memory from source machine to destination over a series of iteration [5, 13].

2.4.1.2 Post-copy memory migration

Post-copy VM migration is initiated by suspending the VM at the original host. While the VM is suspended, a minimal requirement for the execution state of the VM is transferred to the destination host [5]. The VM is then resumed at the destination host and the source host pushes the remaining memory pages of the VM to the destination host. Post-copy sends each page exactly once whereas, pre-copy can transfer the same page multiple times if the page is changed repeatedly at the source host while migration. On the other hand, pre-copy retains an up-to-date state of the VM at the source during migration, whereas with post-copy, the VM's state is distributed over both source and destination. Another thing to note here is, if the destination fails during migration, pre-copy can recover the VM, whereas post-copy cannot [5].

2.5 Non-Uniform Memory Access Architecture

Non-Uniform Memory Access (NUMA) is a method of configuring a cluster of microprocessor in a multiprocessing system so that they can share memory locally, improving performance and the ability of the system to be expanded. This architectures appear as the solution to ease the scalability of modern memory architectures, by interconnecting distributed memory banks [19, 27] and appears as one way of reducing the number of CPUs competing for access to a shared memory bus.

NUMA architecture, have identical processors connected to a scalable network, and each processor has a portion of memory attached directly to it. The primary difference between a NUMA and distributed memory architecture is that no processor can have mappings to memory connected to other processors in case of distributed memory architecture, however in

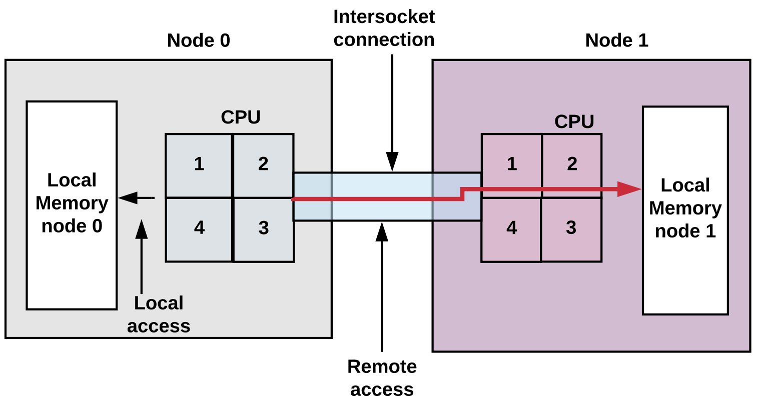

Figure 2.2: NUMA Architecture

NUMA, it is possible.

Figure 2.2 shows a NUMA architecture with two nodes and four CPUs and a local memory for each node.

2.6 Reinforcement Learning

Reinforcement learning is one paradigm of *machine learning* inspired by behaviorist psychology. Instead of giving instructions, it lets the machine go and figure out how to achieve a given task. It train algorithms using a system of reward and punishment. In addition, reinforcement learning involves learning what the next action should be and how to map situations to actions so as to maintain higher rewards [40]. Agents learn by interacting with their environment and receive rewards for performing correctly and penalties for performing incorrectly. Therefore, an agent aims to maximize its reward and minimize its penalty whenever decision is made.

The other two paradigms of machine learning are *supervised learning* and *unsupervised learning*.

Supervised learning is machine learning in which, as the name indicates, there is a given input together with the intended output. This set of labeled examples of input and output pairs are provided by an external supervisor. Each example is a description of a situation together with the label of the correct action the system should take accordingly to a given situation.

Unsupervised learning on the other hand is task of learning to find the hidden. It works in collections of uncategorized data. As the name indicates it is without an external supervisor. Unsupervised learning aims for an Artificial Intelligent (AI) system that is presented with uncategorized data and act on it without prior training Whereas, objective of supervised learn-

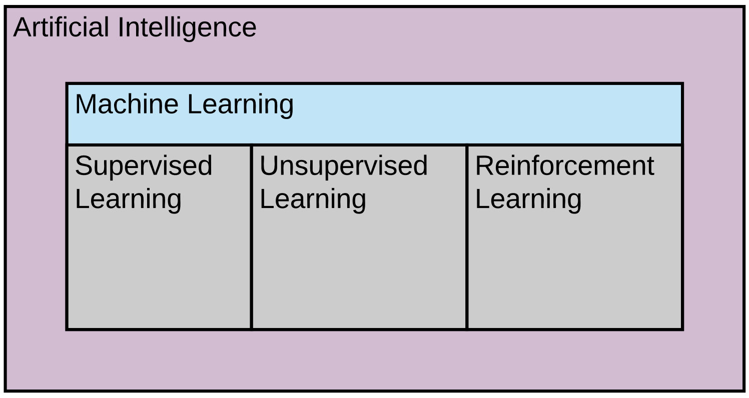

Figure 2.3: Machine Learning Classifications

ing is for the system to generalize, its responses so that it acts correctly in situations not present in the training set.

Reinforcement learning differs from both supervised and unsupervised learning in such a way that reinforcement learning does not rely on examples of correct behaviour in contrast to supervised learning and it is trying to maximize a reward signal instead of trying to find hidden structure as opposed to unsupervised learning.

In reinforcement learning, the agent has to *exploit* what it has already experienced in order to obtain reward, but it also has to *explore* in order to make better action selections in the future [41]. Another key feature of reinforcement learning is that it explicitly considers the whole problem of a goal-directed agent interacting with an uncertain environment. All reinforcement learning agents have known goals that lead them to learn parts of their environments based on subsequent chosen actions to impact their environment. [40, 41].

2.6.1 Elements of Reinforcement Learning

In reinforcement learning system, six main elements are identified: *agent*: which is the intelligent program, *environment*: the external condition the agent interacts with, *policy*: a mapping from states to action that defines the agent's behaviour at a given time, *reward function*: which defines the goal in the reinforcement learning problem and indicates what is good in an immediate sense, *value function*: which is the total amount of reward an agent can expect to accumulate, starting form that state and specifies what is good in the long run. lastly, a *model*:something that mimic the behaviour

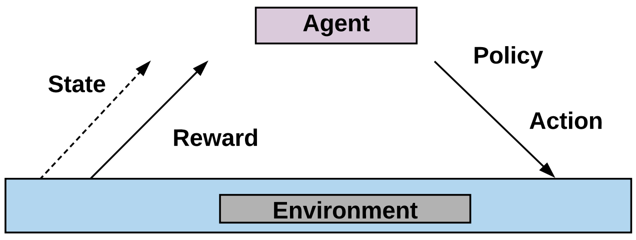

Figure 2.4: Elements of Reinforcement Learning

of the environment so as to predict the next state and reward. Figure 2.4 shows elements of reinforcement learning.

2.6.2 Learning Automata

Automaton plural(Automata) is a term used in computer science and mathematics for a theoretical machine that change its internal state based on inputs and its previous state.

Learning Automata(LA) is type of machine learning algorithm that fall into the range of reinforcement learning. A learning automaton as Wikipedia defines it *"is an adaptive decision-making unit situated in a random environment that learns the optimal action through repeated interactions with its environment. The actions are chosen according to a specific probability distribution which is updated based on the environment response the automaton obtains by performing a particular action."*

Automaton interacts with the environment by choosing an action. This action taken by the automaton initiates a response from the environment, rewards or penalties, Based on the subsequent responses, the automaton grows into developing a knowledge of its environment by choosing the optimal action [1, 47].

2.7 Tools

In order to fully understand the needed features of the project, a short introduction to the tools used is provided in this section.

2.7.1 The Core Project

The Core Project [36] is a project dedicated to providing a minimal Linux distribution that can be configured for any number of purposes. There are three main flavors in the Core suite.

Tiny core Linux(TLC) is a minimal Linux operating system developed by Robert Shingledecker in 2008. Tiny core is designed to run from a RAM copy created at boot time. This feature adds a number of functionality, such as file protection, originality and fast creation. Its small size i.e, 16MB makes it flexible enough to be stored and run USB, CD or even embedded devices. It can also be configured to create customized ISO image, ready to be booted in a VM.

Micro Core Linux(Core) is a smaller variant of Tiny Core without a graphical desktop by default, even though it can be created if needed. Its size is 11MB and is good choice for servers.

CorePlus is 106MB in size with different windows managers, various keyboard layouts and wireless support. It is not a distribution, rather an instalation image.

2.7.2 ezremaster

ezremaster [35] is an open source Graphical User Interface(GUI) application that simplifies remastering Tiny Core or Micro Core Linux. It supports setting all of the boot codes, Adding extension to the remaster and recreating custom ISO image.

2.7.3 Bokeh

Bokeh [9] is a python interactive visualization library that is ideal for quick and easy interactive plots, dashboards, and data applications. It differs from other Python visualization libraries such as Matplotlib in the fact that it is interactive and uses web browsers for presentation. Bokeh provides elegant, concise construction of novel graphics with high-performance interactivity over very large or streaming data sets.

2.7.4 systemd

In Unix-based computer operating systems, initialization(init) is the first process started during booting of the computer system and continues running until the system is off.

systemd was started in 2010 by Lennart Poettering and Kay Sievers. In May 2011 Fedora was the first major Linux distribution to enable systemd as the default init system. As of 2015, most Linux distributions have adopted systemd as their default init system. It handles all the system service calls i.e. start, stop, enable. Enabling a system service ‘systemctl enable <service-name>’, tells systemd to start the service on reboots.

2.7.5 stress-ng

stress-ng, is a re implementation of the original stress tool written by Amos Waterland. It is designed to stress various components of a Linux system. *"stressng will stress test a computer system in various select-able ways. It was designed to exercise various physical subsystems of a computer as well as the various operating system kernel interfaces. stressng also has a wide range of CPU specific stress tests that exercise floating point, integer, bit manipulation and control flow"* [15].

The stress-ng tool includes over 60 different stress tests, over 50 CPU specific stress tests that exercise floating point, integer, bit manipulation and control flow and over 20 virtual memory stress tests [14].

2.8 Related Works

This section will give a birds eye view on prior researches done on fields related to the project. The related works are going to by presented in three categories; Autonomous VM Migration, Temperature Aware Thermal Management in Data Centers and Power Consumption Reduction in Data Centers.

2.8.1 Autonomous VM Migration

An autonomous migration of virtual machines is designed to increase the overall resource utilization on a cluster of servers.

The proposed approach in [11] uses learning framework that autonomously finds and adjusts thresholds at run-time for different computing requirements that consider previous migration history to find the one that cope up with the current situation based on CPU utilization. Experimental results showed that their approach autonomously finds thresholds close to the best ones for different computing scenarios.

In [12] the authors extended the learning framework from their previous work in [11] to handle additional resource types i.e, memory utilization, and propose new proactive learning where they can examine the best combination computing environment thresholds and resource weight. Workload was also considered on both static and random distributions. The experimental results were presented in four parameters: resource type with fixed threshold, resource size, learning method, and workload distribution showing the impact of both CPU and memory utilization on learning patterns.

In [16] the authors propose; an autonomous network aware VM migration strategy and showed that an autonomous agent can learn to utilize available network resources and do a migration. They implement a dynamic reinforcement learning approach. While live migrating, they focus on the current network traffic demand. They argued that time to migrate VMs from an under utilized host can have significant impact on the current cloud system performance in terms of resource consumption. Sequential migration was implemented in their approach i.e, one after the other and they considered CPU utilization. Experimental results showed that RL agent can learn to migrate when utilization of network traffic is low and improve network resource usage at off peak hours.

2.8.2 Thermal Management in Data Centers

Various studies have been done and are still being conducted on thermal management that focus on temperature aware approaches.

The proposed solution by [18] involve control-theoretical thermal balancing (CTB) algorithm that provide online feedback for different servers in a server clusters. The CTB algorithm was engaged to monitor temperature and CPU utilization of servers in their server cluster. They implement two thermal balancing algorithm designs. The first algorithm, CTB-T, uses processor temperature as a feedback whereas, the second one; CTB-UT uses both temperature and CPU utilization. In their algorithm design, they also considered the thermal dynamics of the servers so that it can handle uncertain thermal characteristics such as fluctuating power consumption and thermal flaws. Their simulation result showed the maximum temperature difference among the servers has been minimized to 0.2% in both algorithms with the second algorithm being able to converge quickly than the first one.

Meanwhile, in [32] the authors showed that temperature aware workload placement is crucial in data centers to reduce cooling cost and increase reliability. In their work they suggested implementing thermal balance by smart workload placement based on temperature or CPU utilization might not give the best results. Additionally, how the data center is designed and which areas are exposed to high power utilization depending on the air flow and the cooling capacity should be considered. They proposed a way data centers architecture should be designed to get the best out of thermal and power optimization solutions in general. They also argued that an intelligent resource provisioning together with smart workload placement algorithm that consider both heat flow and thermal dynamics has the potential to reduce cooling infrastructure cost.

Temperature aware workload placement that considers scheduling workloads was proposed by [30]. This approach has shortcomings when it comes functioning on a data center when the utilization is 0% or

100%. The authors presented two scheduling algorithms called zone based discretization (ZBD) and minimize heat re circulation (MinHR) and tested them in real world data center. The algorithms are based on air flow in data centers so that they can keep server inlet temperature within the threshold and maximize the temperature that is pumped into the data center by the Computing Room Air Conditioning (CRAC). The results showed that DigitalMinHR was successful in highly reducing the cooling cost compared to ZBD even though it took 56 hours to converge, and ZBD turned out the persistent solution with only half an hour of convergence time.

2.8.3 Power Consumption Reduction in Data Centers

The authors in [42] pointed out that to reduce power consumption of a data center, both workload on the servers and air flow should be controlled. Their approach takes multiple parameters including outlet air temperature, power consumption of servers and sensor values for certain amount of time and predict the temperature distribution based on that monitored values. Their approach was examined in an experimental data center and resulted in a maximum of 30% power consumption reduction when the air conditioning was controlled by implementing temperature distribution prediction.

In [33] the authors present strategy to reduce data center power consumption by implementing cooling and load management together. Their initial point was based on the fact that most data centers control cooling and computational subsystems independently. Where, the cooling subsystem works to keep the whole data center infrastructure below critical temperature limit. On the other hand, the computational subsystem works to gain high performance and minimize the overall server energy consumption. With their study, they come up with a strategy to implement both cooling and load management together by coordinating the two subsystems. The results from their approach achieved better power management than the traditional approach.

The idea of server consolidation intend to put the load of multiple servers to one server and do a clean shutdown on unused servers intern, reduce power consumption. The proposed strategy in [45] begin with identifying servers based on their workload and usage. They categorized them into three resource pools; innovation, production and mission critical. Afterwards, server consolidation was applied to each of the categories. The experiment result showed that their approach increased the utilization ratio up to 50% saving huge amount of energy.

When data centers are designed there are some locations that generate higher heat than the others because of the air flow and the implemented cooling capacity [32, 38]. The proposed solution by [38] examines two methods to dig into redistributing workloads and its potential gain. The approaches are called Row-Wise Thermal Management and Regional

Thermal Management. Given that the racks at the end of a row in a data center are 10$^\circ C$ higher than row the ones at the middle, implementing the load distribution considered both thermal dynamics and workloads. They stated, the two approaches can also be implemented together. 14% energy consumption was reduced by workload redistribution using the two approaches when experiments were conducted.

Part II

The project

Chapter 3

Approach

This chapter outlines the proposed solution in order to answer the problem specified in the problem statement: "*How to achieve thermal balance on virtualized servers by autonomous migration of Virtual Machines running on them based on the server's temperature reading*".

The proposed solution to the given problem is organized in 3 phases:

(i) Design.

(ii) Implementation.

(iii) Experimentation and Analysis.

The coming consecutive chapters will cover every bits and pieces of each phase in detail, and this chapter gives an idea of how the components of this study are organized in those chapters.

3.1 Objectives

As stated in the introduction chapter, this thesis aims to achieve a thermal balance among virtualized servers. The proposed solution will be based on autonomous migration of virtual machines hosted on servers in a server cluster. Different tools and technologies described in the background chapter will be used.

In this thesis, autonomous migration of VMs will be implemented. The experiments will be conducted using two different approaches. The first approach will enable an autonomous migration of VMs based on the VMs global information of all physical server's CPU temperature. The second, alternative approach will be based on partial information, where the VMs will only know the temperature of the physical server they are hosted on. Two different algorithms will be implemented to test the functionality of the proposed approaches and results will be evaluated.

3.2 Design

This phase of the solution is where all the planning and setting up the test environment happens. It is basically where the blue print of the project will be sketched. This involves 5 main tasks:

(i) Physical machine selection and configuration.

(ii) Virtual machine selection and configuration.

(iii) Workload configuration.

(iv) Designing the two approaches.

(v) Designing the algorithm according to the design plans.

The above main tasks will have detailed explanation in the next chapter, for now a short and brief introduction on what to expect in the details will be given in 4 consecutive sub sections.

3.2.1 Environmental Setup

For successfully designing the proposed solution, three physical servers will be configured. From the presented technologies and tools in the background chapter, the three physical servers will use:

(1) *KVM*: as virtualization technology.

(2) *Libvirt*: as virtual machines managing solution.

(3) *sensors*: to print temperature readings from the servers.

(4) *Bokeh*: to monitor the temperature readings at the servers in real time.

(5) *python* and *bash*: as scripting language.

When it comes to the VMs, a *micro core Linux* from the core project will be used to handle a given workload. The customized core VM will have:

(1) *SSH*: to access and be accessed by the physical servers securely.

(2) *stress-ng*: to create a realistic workload on the VMs.

(3) *python* and *bash*: as scripting language.

Regarding the workload that the VMs handle, it will be designed in three ways. The first set will have a uniform workload distribution where as the second and the third will be of varying load. The three workloads will be designed as:

(1) *Uniform Load*: workload occupying full percentage of CPU utilization.

(2) *Variable Load*: workload occupying different percentages of CPU utilization.

(3) *Dynamic Load*: workload that change percentages of CPU utilization through time.

3.2.2 Algorithm Design

There will be two algorithm designs corresponding to the two approaches. The first algorithm, as in the first approach, will be designed so that the VMs will be feed temperature information from all servers in the cluster. The second algorithm will be based on reinforcement learning and learning automata concepts and involve VM's partial knowledge of its environment in relation to temperature readings at the physical servers.

The two algorithms will be named:

(1) *Choose Coolest Server*: Algorithm based on calculating average temperature and choosing the "coolest" server.

(2) *Learn to Balance*: Algorithm based on learning automata concepts that learn its environment so as to make the best response to it.

3.2.3 Temperature Information Exchange

In the case of the first approach, the servers will exchange and update their temperatures regularly. Every server in the cluster will send its temperature reading to all other servers. Hence, every running VM will receive all the updated server temperatures form the server it is hosted on. On the other hand, the second approach does not involve this. Since, VMs only need the temperature information of the server they are hosted on, they will only require the host server's temperature.

3.2.4 Autonomous Migration

The VMs will be designed to make an autonomous choice of whether to migrate and where to migrate based on temperature data from the physical servers in decentralized approach. The Core VMs will be programmed to execute a specific code that will keep making the decision of migration based on the provided information. The VM management including the migration will be handled using Libvirt VM managing solution explained in the background chapter.

3.3 Implementation

The next phase coming after the completion of the design is implementation where all designed tasks will hit the ground and start functioning. The implementation phase includes:

(i) Integrating the underlying tools and technologies.

(ii) Organizing set of deployment scripts.

(iii) Implementation of algorithms.

A short introduction will be given on each tasks of the Implementation phase in the coming 3 consecutive sub sections.

3.3.1 Underlying Tools and Technologies

The tools and technologies used for implementation will be installed, configured and tested during this phase. There will be number of installed packages on the servers and VMs as well. Necessary packages that will be installed on the physical machines includes; sensors, python, Libvirt, and bokeh. On the other hand the VMs will be running SSH, stress-ng and python.

3.3.2 Deployment Scripts

A number of python and bash scripts will be implemented to fulfill the required functionality and to run successful tests. The deployment scripts will be used in three places. The first set include server side scripts that will run on the servers. The second set of deployment scripts are for the client side and will be implemented on the VMs. The third are scripts implemented on an external machines for the sake of monitoring the setup. The monitoring will include temperature and power readings of the servers in real time.

3.3.3 Algorithm Implementation

The designed algorithms will be implemented in python. This script will be included when the core VMs are customized so that it runs in the background at all times. There will be multiple Core ISO images based on the two implemented algorithms and the workloads handled. When VM provisioning happen the specific ISO image for the experiment will be used accordingly.

3.4 Experimentation, Result and Analysis

This phase is where the setup would be tested and the results be presented and analyzed. The two implemented approaches will also be compared and the better solution will be selected in terms of maximum temperature differences and convergence time. This phase includes:

(i) Set of conducted experiments.

(ii) Presentation of experiment results.

(iii) Analysis and comparison of results.

3.4.1 Experiments and Results

In order to achieve the best accurate solution, the temperature readings will be calibrated at the beginning of all experiments. There will be a real time temperature and power readings monitoring using tools Bokeh and Kibana. Bokeh setup will be in place to visualize the VMs activities in a number of aspects and Kibana for the power consumption. Although their might be additional experiments or changes in the layouts the following test run cases are planned to be included:

(i) *case 1* - Uniform workload distribution with choose coolest server algorithm.

(ii) *case 2* - Variable workload distribution with choose coolest server algorithm.

(iii) *case 3* - Dynamic workload distribution with choose coolest server algorithm.

(iv) *case 4* - Uniform workload distribution with learn to balance algorithm.

(v) *case 5* - Variable workload distribution with learn to balance algorithm.

(vi) *case 6* - Dynamic workload distribution with learn to balance algorithm.

3.4.2 Data Analysis and Comparison

Based on the results from the experimental tests conducted, the visualized data will be organized for further analysis and statistical evaluations. Different methods of data presentation will be used so that the results can easily be understood by readers. In addition comparison to the methods will be given and a discussion will be included to initiate further studies based on the findings from this work.

Chapter 4

Design and Implementation

This chapter covers how the working environment will be designed and configured based on the proposed tasks in chapter 3. It consists of environmental setups and configurations on both the physical servers and the VMs together with the algorithm design. There will be two design approaches for the proposed solution, and will be presented in detail. Moreover, as part of the implementation, steps undertaken will be presented and the necessary tools and technologies integrated for implementation of the solution will be described as well.

4.1 Environmental Setup

Designing the experimental environment requires physical servers that provision Virtual Machines (VMs) with a given workload and VMs that are able to make autonomous decision of whether to migrate and where to migrate based on temperature readings of those physical servers, the goal being thermal balance across the server cluster.

In the next consecutive sections; physical machine, VM and VM workload configurations will be described. The design for autonomous VM migration will also be included.

4.1.1 Physical Server Specification and Configuration

The experimental environment includes three physical servers located at Oslo Metropolitan University (OsloMet). The servers that are used in this thesis are homogeneous, hence all the three servers have the same specification as given in Table 4.1.

Architecture	x86_64
Model name	AMD Opteron^{TM} Processor 6234
Operating System	Ubuntu 16.04.3 LTS
CPU op-mode(s)	32-bit, 64-bit
CPUs	48
On-line CPU(s) list	0-47
NUMA node(s)	8

Table 4.1: Physical Server Specification

Advanced Micro Devices (AMD) is the world's second largest manufacturer of x86 architecture processors. The AMD Opteron ^{TM} Processor 6234 lies within 6200 Series and it is characterized with large memory footprints [3]. Regarding temperature, AMD machines report two types, *Core Temperature* and *CPU Temperature*. The CPU Temperature is read by a sensor in the socket of the motherboard. It is the physical temperature and is more precise at low levels, whereas Core Temperature is a non physical arbitrary scale designed to imitate a temperature sensor [4, 20].

Table 4.2 shows the Non Uniform Memory Access (NUMA) nodes and CPUs found on a single physical server. The NUMA architecture has been explained in the background chapter section 2.5. At every server, there are eight NUMA nodes and at each NUMA node there are six CPUs and a single temperature sensor. Thus, there will be 48 CPUs in total on a single server. In this kind of setups with more than 12 CPUs, NUMA architecture is the recommended way of organizing the processors because, it reduces the number of CPUs competing for a shared memory bus.

In this thesis, the workload utilizing the CPUs will be generated using the VMs running on the servers. The VM and workload configuration will be explained on the next sub sections.

NUMA node0 CPU(s)	0,4,8,12,16,20
NUMA node1 CPU(s)	24,28,32,36,40,44
NUMA node2 CPU(s)	2,6,10,14,18,22
NUMA node3 CPU(s)	26,30,34,38,42,46
NUMA node4 CPU(s)	3,7,11,15,19,23
NUMA node5 CPU(s)	27,31,35,39,43,47
NUMA node6 CPU(s)	1,5,9,13,17,21
NUMA node7 CPU(s)	25,29,33,37,41,45

Table 4.2: List of NUMA Nodes and CPUs

4.1.2 Virtual Machine Configuration

The light weight variant of Tiny Core called Micro Core Linux (Core) described in the background chapter section 2.7.1 will be used as a virtual machine. It provides a small minimal Linux distribution that can be used for any purpose with customized configurations. The customization will be done using a tool called ezremaster. This tool provides a Graphical User Interface (GUI) to specify what the custom ISO image is supposed to include. The remastered Core VM for this thesis will be set to run SSH server and Python will be installed as well. In addition, whenever a VM is provisioned, it will implement a specific algorithm and a given workload set to be handled. The base image of any Core VM is 11MB, after remastering and including needed packages and scripts, the final image occupied 226MB.

4.1.3 Workload Configuration on VMs

As a workload generator, stress-ng tool will be used on the VMs. The stress-ng tool as explained in the background chapter section 2.7.5, loads and stresses a server in various selected ways. The workload will be designed with CPU stress method that provide a combination of memory, cache and floating point operations. The workload will be designed in three ways. The first set of workloads will occupy full percentage of CPUs and the second will occupy either of the four percentages (25, 50, 75 and 100) of CPU load. The last one will have dynamic workload distribution where, all VMs start with the same percentage of workload and then alter to a dissimilar percentage. In order to have a realistic workload distribution among VMs, this three ways of workload generation will be used and tested.

4.2 Design

There will be two designs proposed for implementation. Each design will be described accordingly together with their corresponding algorithms in subsequent sections.

4.2.1 Choose Coolest Server

In this design approach, every sever will exchange its total temperature regularly, which will then be used by the VMs to decide whether to migrate and where to migrate. In order to exchange this temperature readings and other information needed for visualization, the communication will be through a User Datagram Protocol (UDP), even though there is a working SSH configuration between the servers and the VMs in place. The temperature readings will be structured in a JavaScript Object Notation(JSON) format. JSON formatting is an open source file formatting that consists of attribute-value pairs and array data types. In addition to being human readable text, it is also language independent [44].

VM provisioning will take place from all the servers, where provisioned VMs will handle a given workload. In order to see the thermal effect, different number of VMs will be provisioned from each server, so that the servers temperature will be different. The provisioned VMs will initiate a UDP communication with the server they are hosted on, requesting temperature of all servers in the cluster. Shortly after they receive the response from the hosting server, VMs will implement the algorithm in place and autonomously migrate if migration is decided else, they will request for updated temperature readings. As soon as the VM decide to migrate and select destination, server with the lowest temperature, it will send a UDP packet to the server it is hosted on. This packet will include migration requesting message having the VM's MAC address and the host name of the server it intended to be migrated to. This way the migration of VMs happen. On the other hand, if a VM decides to stay there at the hosting server, it will request for an updated temperature readings and it continues this way.

For this design to be implemented, two communications are needed. The first will be communication between the physical servers and the second between a physical server and the VMs. The next sub section presents how this communications is designed.

4.2.1.1 Server-to-Server Communication Design

The three physical servers will be computing their own total temperature gathered from the 8 temperature sensors at each NUMA node regularly. The communication between the physical servers will have a mesh topology where every server is communicating with every other server in the cluster. The UDP payload will be in JSON format. At the beginning of the communication, the servers will send their own temperature to other servers in the cluster, so that all the servers have updated temperature readings and then with consecutive communications, they will update their own temperature value and send to the other servers in the cluster.

4.2.1.2 VM-to-Server Communication Design

Each provisioned VM will send a UDP datagram to the hosting server regularly. First, requesting for the temperature of all the servers and after receiving it, the VM will process the temperature readings using the implemented algorithm and will make the smart decision of whether to migrate and where to migrate autonomously. If a decision to migrate is made, here goes the second message with request for migration. If not, that means the VM decided to stay where it is and it will again send UDP datagram requesting temperature again.

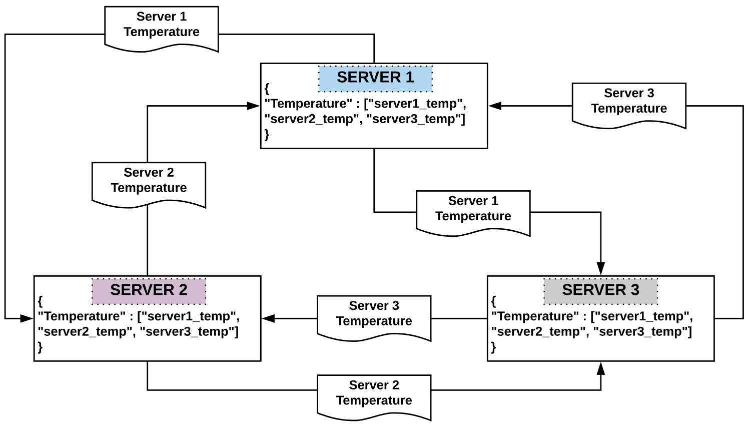
Server 1 Temperature
SERVER 1
{
"Temperature" : ["server1_temp", "server2_temp", "server3_temp"]
}
Server 3 Temperature
Server 2 Temperature
Server 1 Temperature
SERVER 2
{
"Temperature" : ["server1_temp", "server2_temp", "server3_temp"]
}
Server 3 Temperature
SERVER 3
{
"Temperature" : ["server1_temp", "server2_temp", "server3_temp"]
}
Server 2 Temperature

Figure 4.1: Server to Server Communication Design

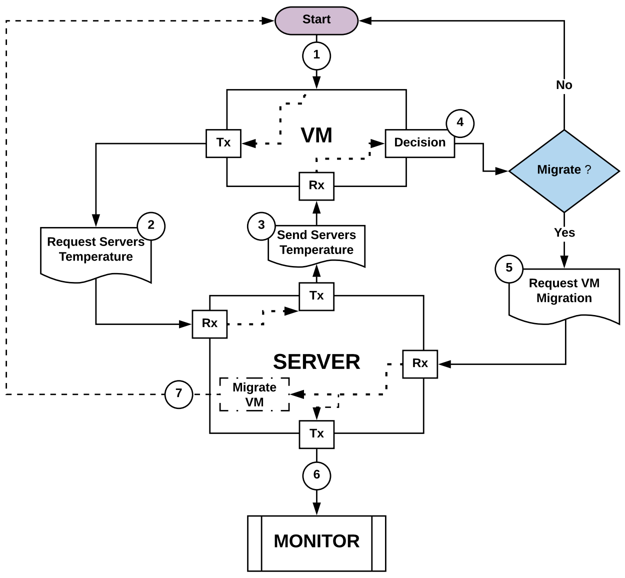

Figure 4.2: VM to Server Communication Design

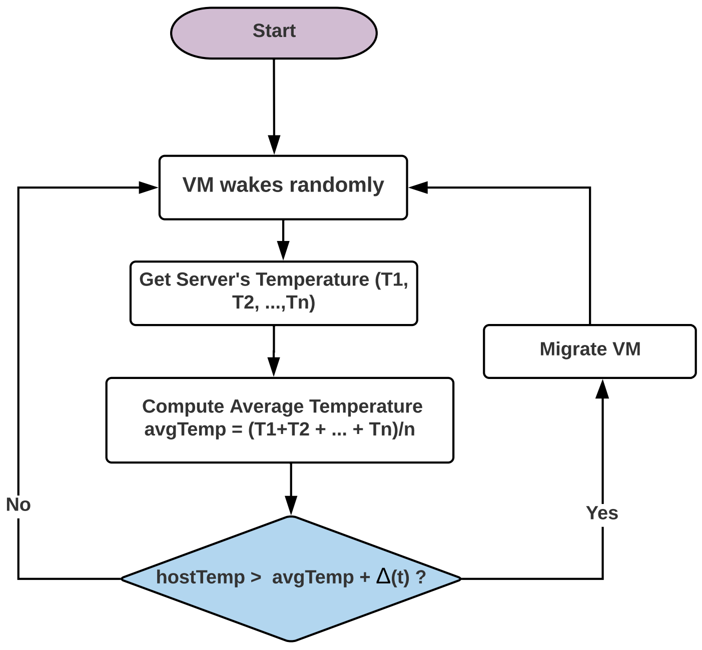
Start
VM wakes randomly
Get Server's Temperature (T1, T2, ...,Tn)
Compute Average Temperature avgTemp = (T1+T2 + ... + Tn)/n
hostTemp > avgTemp + Δ(t) ?
No
Yes
Migrate VM

Figure 4.3: Algorithm Design: Choose Coolest Server

4.2.1.3 Algorithm Design: Choose Coolest Server Algorithm

This algorithm requires a VM to make decision by computing average temperature of the servers by itself. After the average temperature is calculated, the VM will compare the value to the hosting server's temperature. If the hosting server's temperature is above the calculated average temperature, the VM will decide to migrate and choose the server with the lowest temperature. On the contrary, if the hosting server's temperature is below the average, it means the server is not as heated as the other two servers. Thus, VM decides to stay where it is. The algorithm design is given in 4.3.

Algorithm 1: Choose Coolest Server Algorithm

```
1 while true do
2     VM wakes up randomly between(20 and 120 secs);
3     Request temperatures of all the servers in the cluster;
4     Identify host temperature, hostTemp;
5     Compute average temperature of all servers, avgTemp;
6     if $hostTemp > avgTemp + \Delta(t)$ then
7         Migrate to the server with lowest temperature;
8     else
9         go to step 2
```

4.2.2 Learn to Balance

While the first design requires full information of temperature readings in server cluster, what differs in this one is, the VMs will have partial information. That is, the VM only have the temperature reading of the server it is hosted on currently. This algorithm is based on reinforcement learning techniques using learning automata. The learning automata concept was adopted from the active probing and learning algorithm provided in [25].

Here is how it is designed. After VM provisioning take place, a VM will initially take action and migrate to one of the physical servers based on set of uniform probability values. Then the learning automata probability values will be updated. As the VMs migrate, the probability of the destination server that a VM choose will be increased, based on a reward function, and at the same time the probabilities of the other servers will be decreased. The reward function is designed in such a way that, VMs migrated to servers having lower temperature value will be rewarded greatly, In the contrary they will be rewarded less. This way, a VM will keep on learning the environment better and make the right decisions that will lead to a balanced thermal state in the server cluster.

Different from the design approach presented in section 4.2.1, this design requires only communication between the host server and a VM running on it. This communication will be designed the same way as in the VM to server communication in the choose coolest server design, as shown in Figure 4.2.

4.2.2.1 Algorithm Design: Learn to Balance Algorithm

The designed algorithm for the above design involves reinforcement learning concepts and specifically, learning automata. Learning automata is explained in the background chapter section 2.6.2. Here in this context the feedback will be the temperature value and high temperature is considered as penalty whereas low temperature means a reward. The feedback is calculated as seen in Algorithm 2. The sum of the probabilities of moving to the three servers is one as in $P_1 + P_2 + P_3 = 1$. Therefor, a VM is rewarded for moving to a server means the probability value will increase for that machine and the probability of the others will decrease as a result.

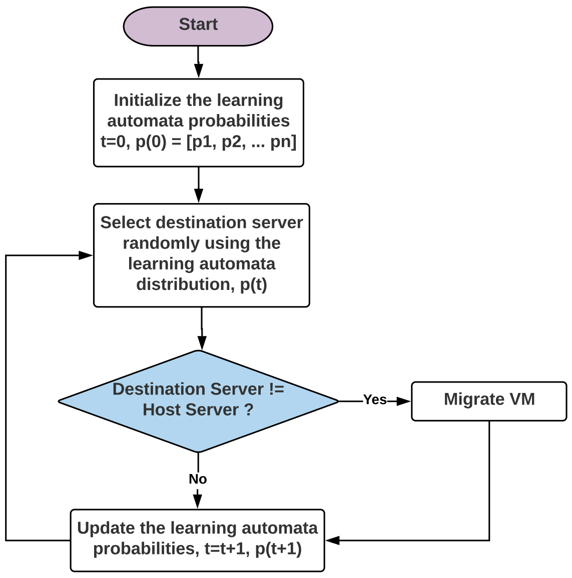
Start
Initialize the learning automata probabilities t=0, p(0) = [p1, p2, ... pn]
Select destination server randomly using the learning automata distribution, p(t)
Destination Server != Host Server ?
Yes
Migrate VM
No
Update the learning automata probabilities, t=t+1, p(t+1)

Figure 4.4: Algorithm Design: Learn to Balance

Algorithm 2: Learn to Balance Algorithm

```
1  Initialization: t=0 $\pi_{ij}(0) = 1/m,\ where;\ j \in \{1, ..., m\}\ is\ the\ set\ of\ servers\ and\ satisfies \sum \pi_{ij}(t) = 1, \pi_{ij} \rightarrow probability\ of\ VM_i\ to\ move\ to\ server\ j$
2  Select destination server $u$ according to distribution $\pi_i(t)$
3  if $i$=$u$ then
4      migrate VM$_i$ to server $u$
5  else
6      VM$_i$ stays on server $j$;
7  Compute the feedback function (reward strength) by
       $z(u,t) = 1 - \frac{T(u,t)}{T_{max}}$
   where, $T(u,t)$ - measured temperature of server $u$ and T$_{max}$ - pre defined maximum temperature;
8  Update the learning automata probabilities maintained at the source VM$_i$ i.e, probability of VM$_i$ to move to server $j$;
9  $\pi_{ij}(t+1) = \pi_{ij}(t) + G * z(u,t) * (\delta_{ju} - \pi_{ij}(t))$
10 where G is the learning gain and $\delta_{ju} = \begin{Bmatrix} 1 \text{ if } j = u \\ 0, \text{otherwise} \end{Bmatrix}$
11 t = t + 1
```

4.3 Implementation

This section provides the steps under taken to implement the two designs.

4.3.1 Customizing an ISO Image

This project requires autonomous VM migration. In order for this to happen, a custom Core ISO image was built on top of the official Core VM base image. GUI tool called ezremaster introduced in section 2.7.2 was used for doing the remastering. The result was a 226MB Core ISO image with SSH Server and Python installed on it, with scripts for autonomous VM migration and workload based on stress-ng utility tool.

The ISO image was created from a running TinyCore instance. The script attached on Appendix A.1 was used to take care of the installation and configuration of the customized core ISO image. Once the script completes, the newly created ISO image can be downloaded from the TinyCore instance using the Linux *scp* command.

4.3.2 Registering a script as a Linux systemd service

In order to keep the server script up and running at all times, it was registered as systemd service. systemd, as explained in section 2.7.4 of the background chapter, it is an initialization (init) system for Linux distributions. The systemd service guarantees the server script which does most of the tasks is always up and running unless intentionally stopped. This was achieved by registering the script as a service that will restart if it fails. The service was placed at */lib/systemd/system/* as *server.service*. The python code was named *server.py* and placed at */home*.

The content of the file *server.service* is given below:

```
[Unit]
Description=VM manager
After=multi-user.target

[Service]
Type=idle
ExecStart=/usr/bin/python3 /home/server.py
Restart=always

[Install]
WantedBy=multi-user.target
```

4.3.3 Generating Workload on Running VMs

The workload the VMs handle was implemented in three ways with regard to CPU load percentage using stress-ng tool. The first one aims to handle uniform workload set to utilize 100% of a CPU. The second one is with variable workload. The variable load was occupying 25, 50, 75 and 100 percentages of the CPU. The last one has dynamic workload distribution where all VMs start with uniform load and then change to either of the loads specified in the variable load.

The stress-ng tool was built and deployed in the VMs from source found in [24] and the command used to generate the workloads is:

```
stress-ng --cpu N --cpu-method matrixprod --cpu-load P
 ↪  --timeout T
```

The switch *–cpu* specifies the number of CPUs to stress on a single server and the switch *–cpu-method* in this command tells about a specific CPU stress method selected among the many that exist [15]. In the command what the selected *matrixprod* attribute does is, it will "*matrix product of two 128 × 128 matrices of double floats. Testing on 64 bit x86 hardware shows that this is provides a good mix of memory, cache and floating point operations and is probably the best CPU method to use to make a CPU run hot*" [15] whereas the *–cpu-load* sets the percentage of the CPU to load and stress. In addition a timeout can also be given by adding *–timeout*.

4.3.4 Deployment and Migration of Virtual Machines

The virtualization technique used in this project involves a number of technologies and tools. The setup was implemented on a KVM virtualization with Libvirt as VM managing solution. The core VM was selected for its capability of customization and small size. The Core VMs were deployed on a virtual network bridge on the subnet 192.168.122.1/24. In order to avoid collision of MAC and IP addresses of the VMs in the server cluster, the virtual bridge interfaces DHCP range has been assigned accordingly using this script.

```
networkname=virsh  net-list --all
virsh net-edit $networkname

virsh  net-destroy $networkname
virsh  net-start $networkname

ifconfig virbr0 down
ifconfig virbr0 up
```

The folllowing command was used to provision VMs:

```
virt-install --name=vm1 --cdrom=my.iso --virt-type=kvm
 ↪  --os-type=linux --ram=256 --network bridge=virbr0 --nodisk
 ↪  --graphics=none --noautoconsole
```

The result of the above command is:

```
Starting install...
Creating domain...

virsh list --all
 Id    Name                           State
----------------------------------------------------
 1    vm1                            running
```

The migration command the host server use to migrate a VM looks like this:

```
virsh migrate --live vm1 qemu+ssh://targetserver/system
 ↪  --undefinesource --persistent
```

The switch *–persistent* makes the migration persistent and *–undefinesource* lets for the VM guest definition on the source host to be deleted after a successful migration.

4.3.5 Monitoring Configuration

Bokeh was used to visualize the temperature readings of the servers in real time. Following the progress was a crucial part of the thesis, that is why bokeh has been selected to give interactive and real time readings. The plots were divide into four. The first plot shows the temperature readings in the server cluster. The second one tracks the number of VMs running at each server, whereas the third plot shows the workload distributions. Lastly, for the learn to balance algorithm experiments, additional plot to show the learning automata probability of a VM is provided.

Kibana setup was in place to visualize the power consumption readings of the servers at Oslo Metropolitan University (OsloMet) beforehand, and that was adapted to this thesis work and used to visualize how the power consumption reading is responding to the implemented design approaches.

Chapter 5

Results and Analysis

This chapter presents various experiments conducted in this thesis. Experimental results will also be analyzed. In addition, comparison of results from the implementation of choose coolest server and learn to balance algorithms will be included.

As stated in chapter 4 section 4.2.1.3 the first implemented algorithm was the choose coolest server algorithm. This section presents the preliminary experiment and subsequent successful experiments. In addition, analysis based on the acquired results will be included.

Core VMs from the Core project were used for remastering custom VMs, which handles different types of workloads; uniform, variable and dynamic, using *stress-ng* utility tool. Uniform workload occupies full percentage of the CPU utilization and variable workload occupies a set of predefined CPU utilization percentages, whereas dynamic workload changes the CPU utilization percentages over time.

Three physical servers with specifications described in section 4.1.1 were used for the experiments. The physical servers has a NUMA architecture with 8 NUMA nodes, where there is one CPU temperature sensor per node. This temperature readings were calibrated at the beginning of the experiments in order to achieve the best possible accuracy.

The temperature, number of hosted VMs and workload distributions of all servers were monitored using Bokeh web application, whereas power consumption readings using Kibana web application in real time. The default readings of both temperature and power are shown in Figure 5.1 and 5.2.

The experimental results will be presented in terms of:

(i) Workload distribution in CPU utilization percentage.

(ii) Number of VMs running.

(iii) Temperature readings relative to the calibrated temperature.

(iv) Power consumption readings in Watts.

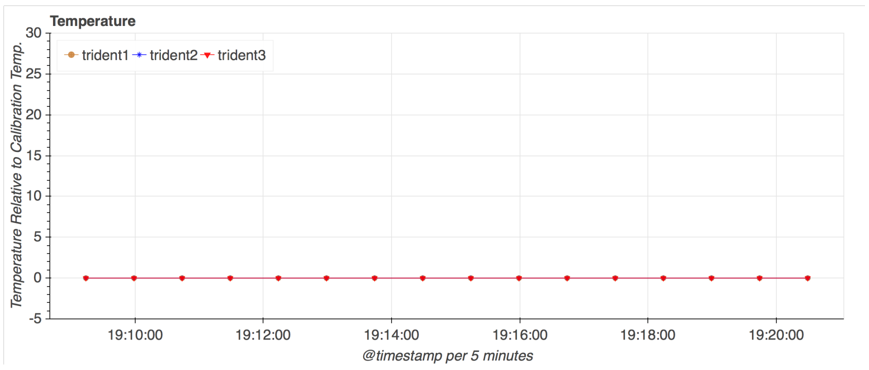

Figure 5.1: Calibrated server temperature readings

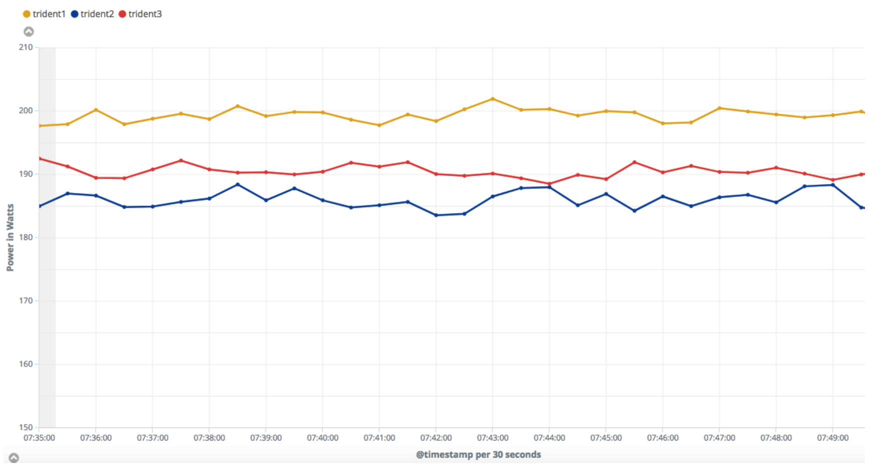

Figure 5.2: Idle server power consumption readings

5.1 Experimental Results: Choose Coolest Server Algorithm

The choose coolest server algorithm implementation was based on the VMs ability to do an autonomous migration relying on the provided temperature readings from all servers in the cluster. Experiments were carried out by turning on and off the algorithm to present the obtained thermal balance in a more clear way.

5.1.1 Preliminary Experiment

In this very first experiment, 45 VMs were provisioned with uniform workload on the three physical servers with load distribution shown in Figure 5.3. To have thermal imbalance among the servers, different number of VMs where provisioned. This way it is quite clear to see the achievement of the autonomous migration.

The results from this experiment are given in Figure 5.4, showing the number of VMs running at the servers, Figure 5.5, showing the temperature readings and Figure 5.6, showing the power consumption readings.

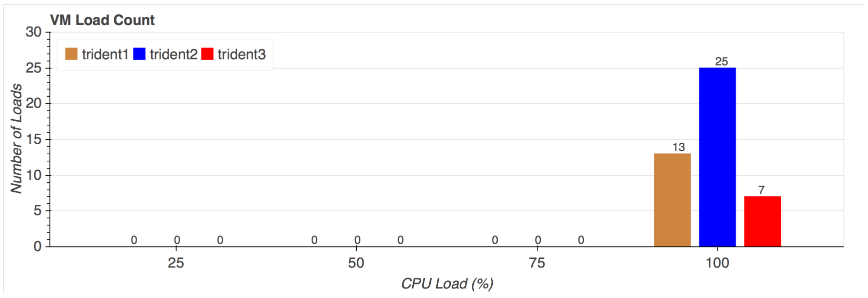

Figure 5.3: Initial uniform workload distribution

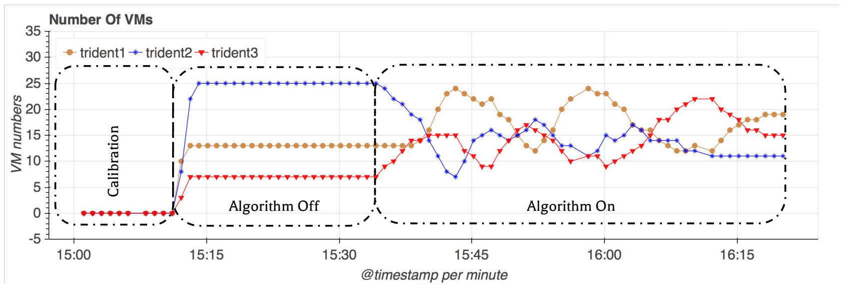

Figure 5.4: Number of running VMs with uniform workload

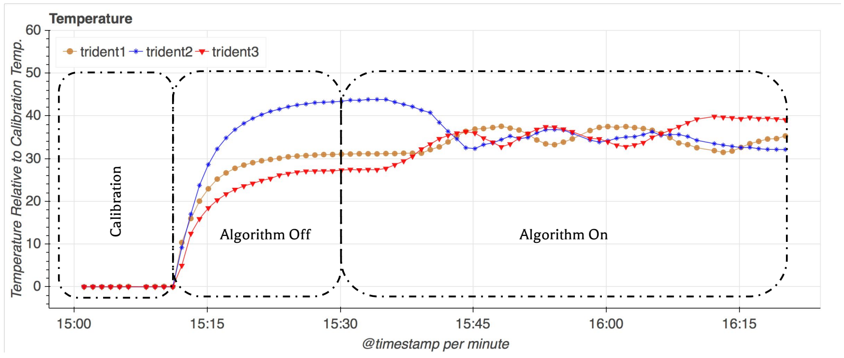

Figure 5.5: Temperature with uniform workload

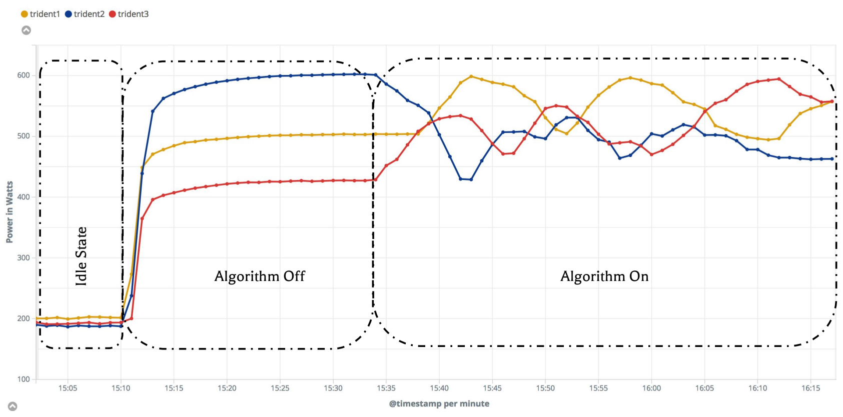

Figure 5.6: Power consumption with uniform workload

As shown in the preliminary experiment, the ability to make autonomous migration decisions was achieved, but there was no convergence that looks like, equalizing the temperature to attain the expected thermal balance. The VMs were going back and forth, due to the temperature imbalance among the servers. Naturally, server's temperature takes time to regulate, because they respond slowly to temperature variations before and after change has occurred.

In the subsequent experiments this situation was avoided by applying delta temperature value, where VMs will stop moving around after the servers reach to a specific temperature difference.

5.1.2 Uniform Workload Results

The provisioned VMs has uniform workload with 100% CPU utilization. After the VMs were provisioned, they wake up randomly and do the autonomous migration. The initial workload distribution was set as shown in Figure 5.3. The results are shown in Figures 5.9, 5.7, 5.8 and 5.10 in terms of workload distribution after thermal balance, number of running VMs, temperature and power readings respectively.

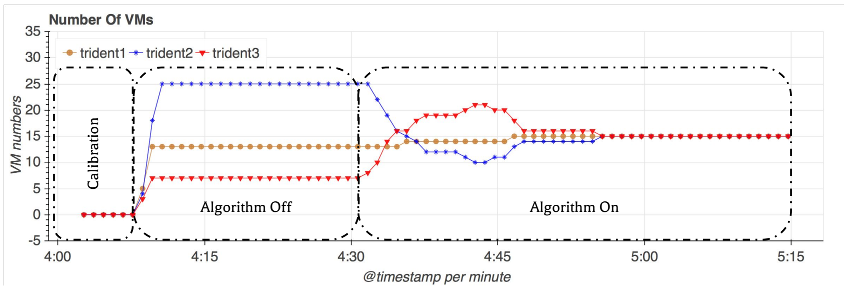

Figure 5.7: Number of running VMs with uniform workload

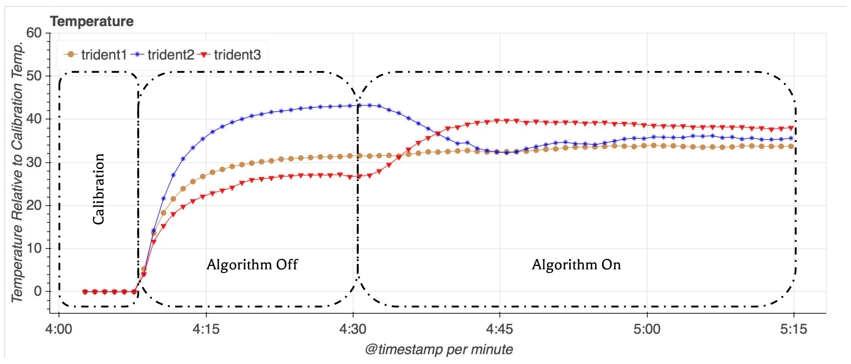

Figure 5.8: Uniform workload temperature readings

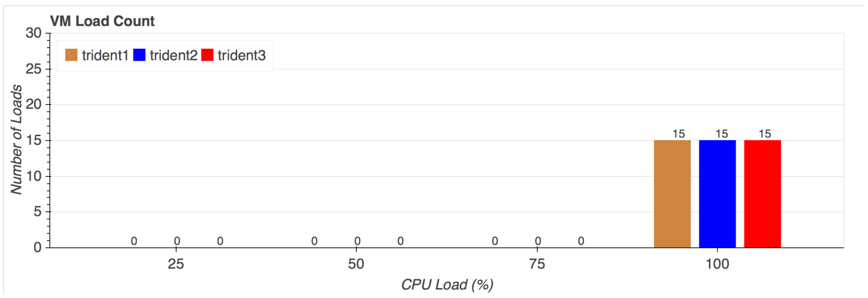

Figure 5.9: Uniform workload distribution after thermal balancing

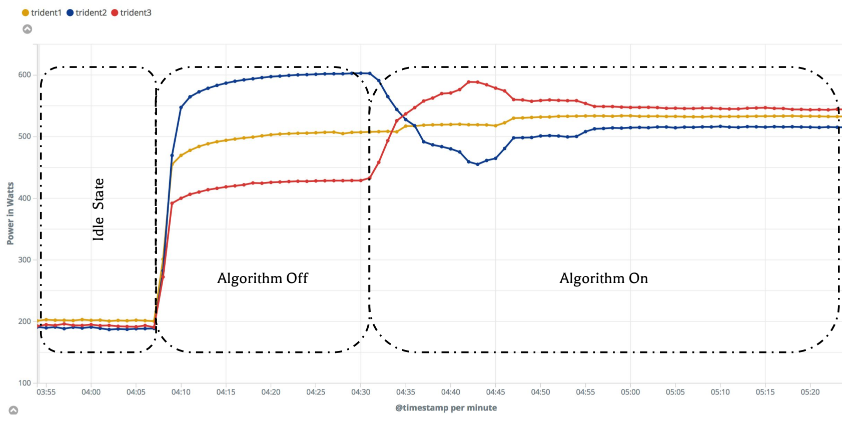

Figure 5.10: Uniform workload power consumption reading

5.1.3 Uniform Workload Analysis

The ping pong situation observed in the preliminary experiment was fixed by introducing a time varying delta temperature value. Where, this time varying delta value was set to start with Δ_{max} and reach to Δ_{min} progressively. Afterwards the VMs were able to stop migration when Δ_{min} temperature difference was attained among the servers in the cluster.

The need for a time varying delta temperature was for the fact that servers heat up and cool down slowly, and the effect of the VMs movement is reflected by the servers gradually. Thus having the time varying delta temperature improved the decision making by catching relatively accurate temperature changes when VMs come and leave.

The result with this experiment showed the VMs were able to do the autonomous migration and maintain thermal balance among the servers. As stated in Table 5.1 the convergence time taken for this algorithm was fairly short as compared to the initial workload distributions variation among the servers.

Choose Coolest Server Algorithms - Uniform Workload	
Number of Running VMs	45
Convergence Time(mins)	50
Relative Temperature Diff. before thermal balance	16.6
Relative Temperature Diff. after thermal balance	3.8

Table 5.1: Analysis: Choose Coolest Server with Uniform Workload

As part of the analysis, box plot representation of the temperature readings is presented in Figures 5.11 and 5.12 to indicate the variations before and after thermal balancing.

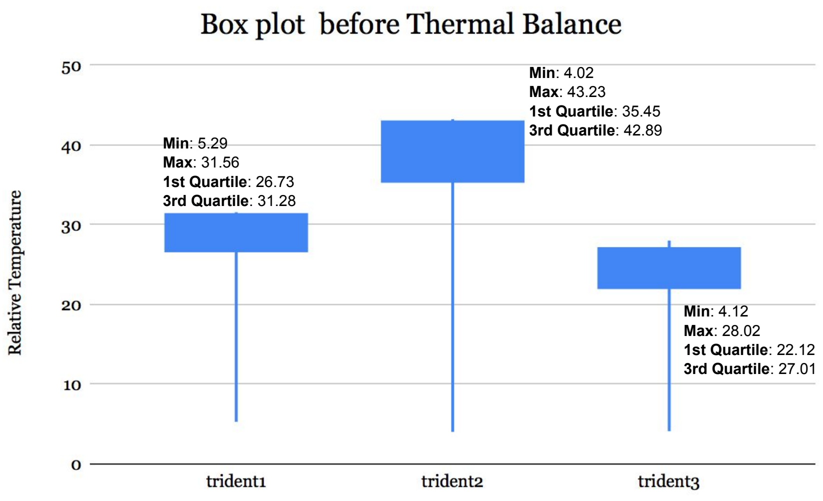

Figure 5.11: Box plot for uniform workload temperature readings before thermal balance

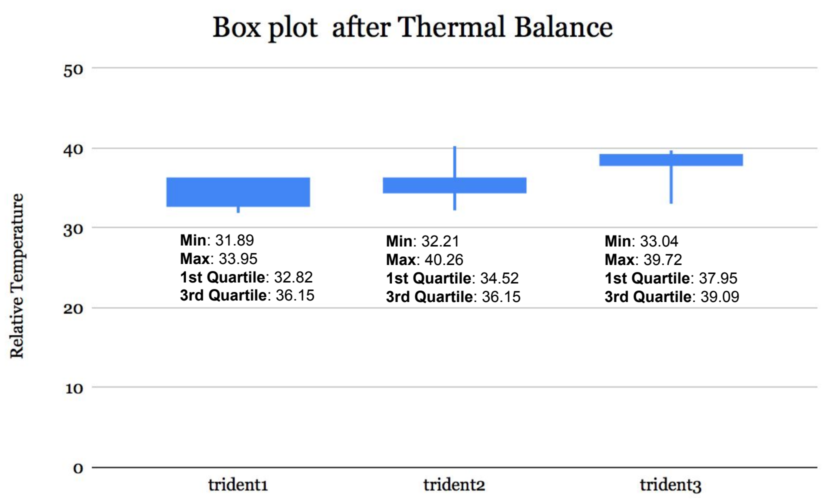

Figure 5.12: Box plot for uniform workload temperature readings after thermal balance

5.1.4 Variable workload Results

This experiment aims to examine and evaluate the implementation results for variable workloads. The workloads were adjusted to occupy either 25%, 50%, 75% or 100% of CPU utilization. The initial workload distribution on the servers can be seen in Figure 5.14 and the results from Figures 5.15, 5.16, 5.17 and 5.13.

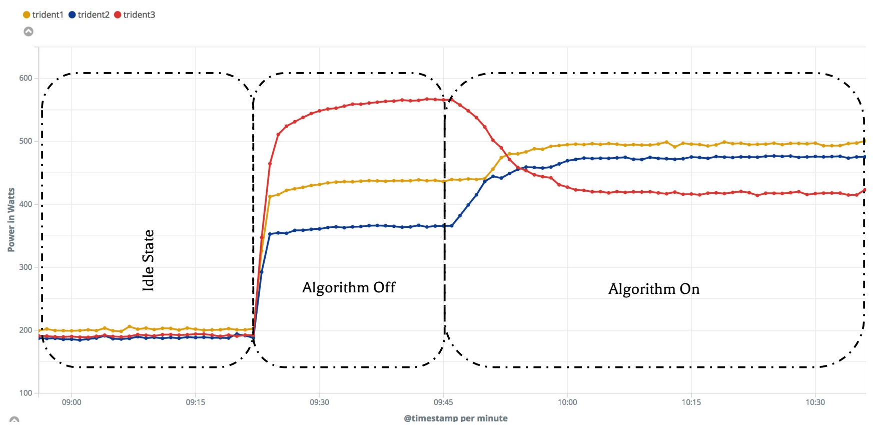

Figure 5.13: Power consumption with variable workload

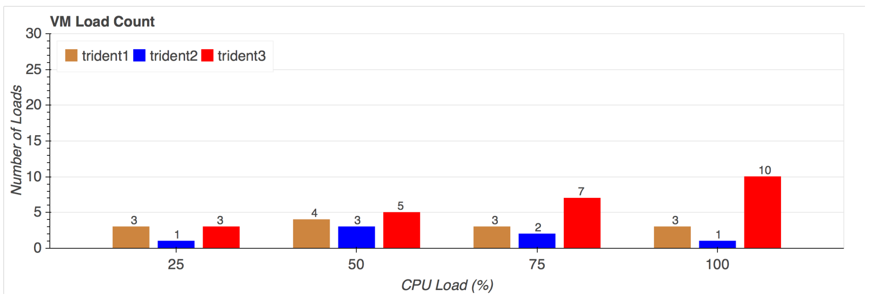

Figure 5.14: Initial variable workload distribution

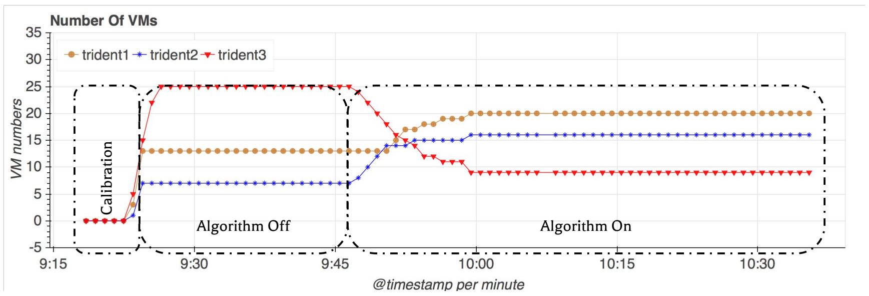

Figure 5.15: Number of running VMs with variable workload

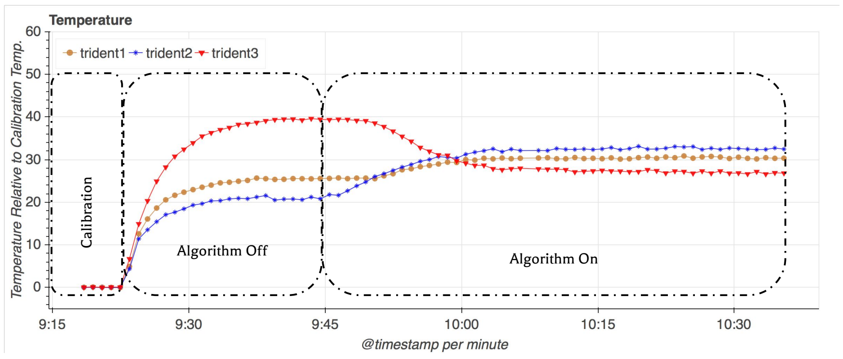

Figure 5.16: Temperature readings with variable workload

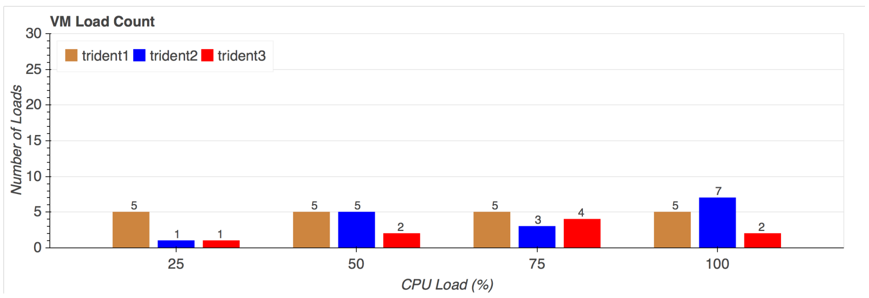

Figure 5.17: Variable workload distribution after thermal balancing

5.1.5 Variable workload Analysis

The experiment with variable workload was also a success with regard to achieving thermal balance across servers by the implemented autonomous VM migration. The experiment was conducted on 45 VMs as in the case of uniform workload. The summary of the results in terms of convergence time and maximum temperature difference is given in Table 5.2

Choose Coolest Server Algorithms - Variable Workload	
Number of Running VMs	45
Convergence Time(mins)	45
Relative Temperature Diff. before thermal balance	17.5
Relative Temperature Diff. after thermal balance	4.2

Table 5.2: Analysis: Choose Coolest Server with Variable Workload

The same time varying delta temperature was in place as in the uniform workload. The workload variations were chosen to reflect realistic workload distributions as much as possible.

In addition, a box plot representation of the temperature readings can be found in Figures 5.18 and 5.19 representing the variations before and after the attained thermal balance respectively.

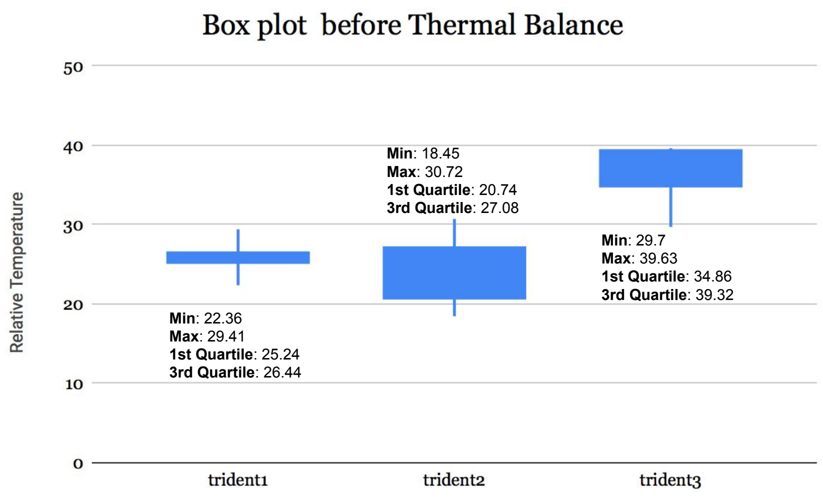

Figure 5.18: Box plot for variable workload temperature readings before thermal balance

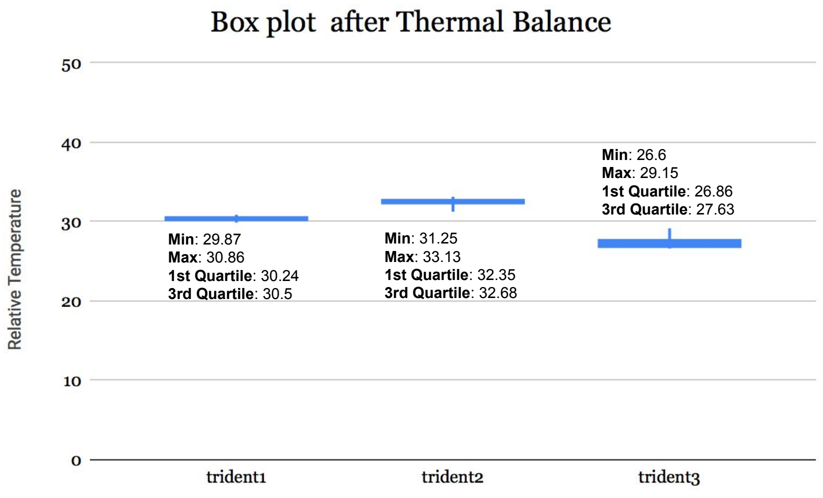

Figure 5.19: Box plot for variable workload temperature readings after thermal balance

5.1.6 Dynamic workload Results

The objective of this final experiment with the choose coolest server algorithm was to prove that the implemented algorithm works as expected when there is a dynamic workload variation.

The experiment was conducted using 45 VMs with dynamic workload that started with uniform workload distribution that used 100% of CPU utilization and changed to either 25%, 50% or 75% accordingly. The initial workload distribution at the servers can be seen in Figure 5.21 and the results are presented in Figures 5.22, 5.23 and 5.20.

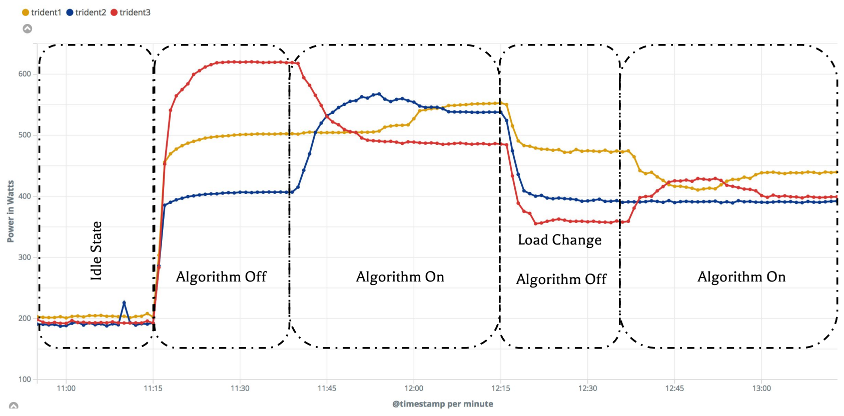

Figure 5.20: Power consumption with dynamic workload

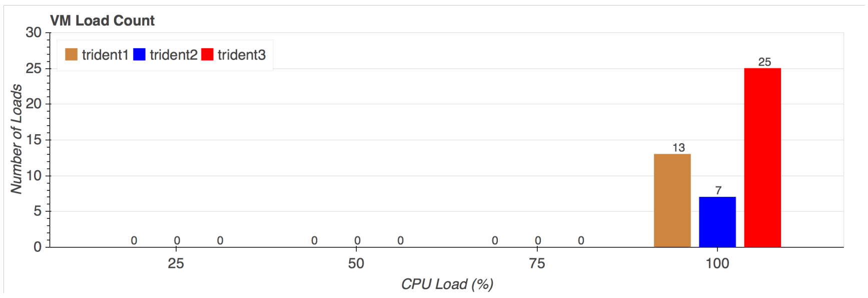

Figure 5.21: Initial dynamic workload distribution

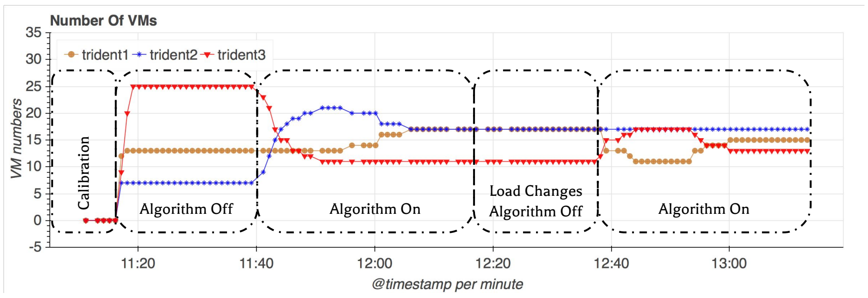

Figure 5.22: Number of running VMs with dynamic workload

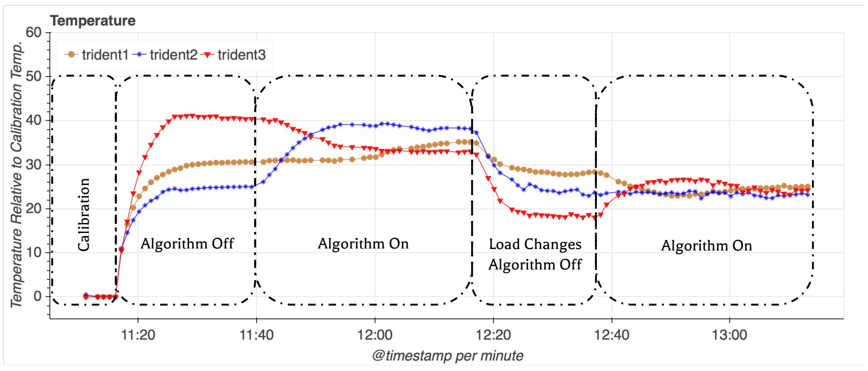

Figure 5.23: Temperature readings with dynamic workload

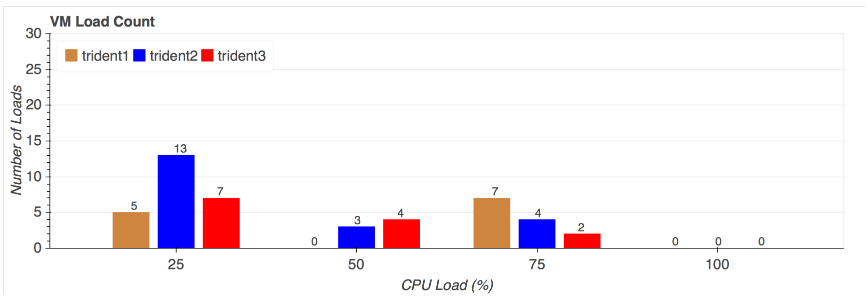

Figure 5.24: Dynamic workload distribution after thermal balancing

5.1.7 Dynamic workload Analysis

The objective of this experiment was to test the VMs intelligence in coping up with the environment whenever there is change and it worked well.

This experiment has employed dynamic workload and has achieved thermal balance among the servers as expected. The experiment was conducted on 45 VMs the same way as the previous experiments. The summary of the results in terms of convergence time and maximum temperature difference is provided in Table 5.3.

Choose Coolest Server Algorithms - Dynamic Workload	
Number of Running VMs	45
Convergence Time(mins)	40
Relative Temperature Diff. before thermal balance	16.3
Relative Temperature Diff. after thermal balance	1.2

Table 5.3: Analysis: Choose Coolest Server with Dynamic Workload

5.2 Experimental Results: Learn to Balance Algorithm

The learn to balance algorithm was based on a reinforcement learning techniques, where each VM locally runs an independent learning automata. Thermal balancing between all servers in the cluster is learned using distributed learning automata.

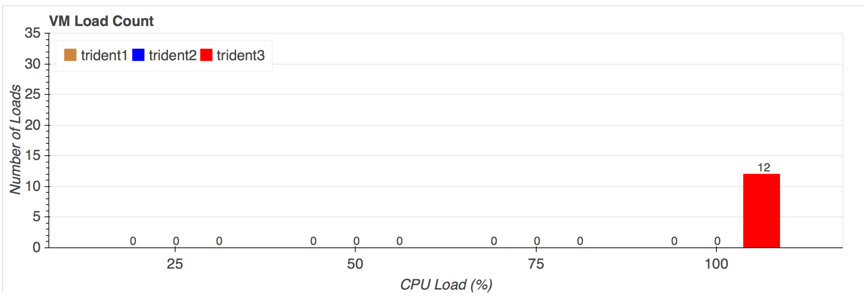

Figure 5.25: Initial uniform workload distribution

5.2.1 Uniform Workload Results

This experiment was conducted on VMs with uniform workload.The initial workload distribution is shown in Figure 5.25. The total number of VMs used were 12. The results from this experiment are given in Figures 5.27, 5.28, 5.29 and 5.26 representing the number of running VMs, temperature readings, final workload distribution and power consumption respectively.

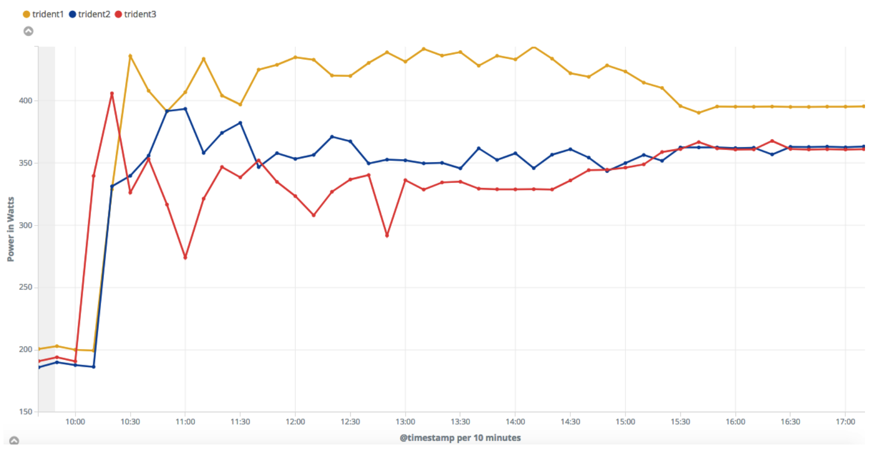

Figure 5.26: Power consumption with uniform Workload

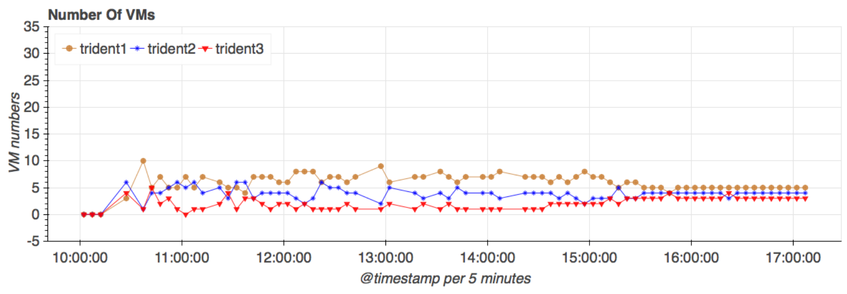

Figure 5.27: Number of running VMs with uniform workload

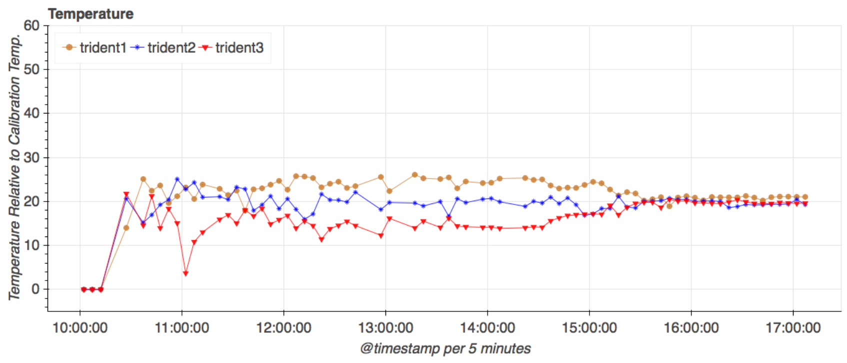

Figure 5.28: Temperature readings with uniform workload

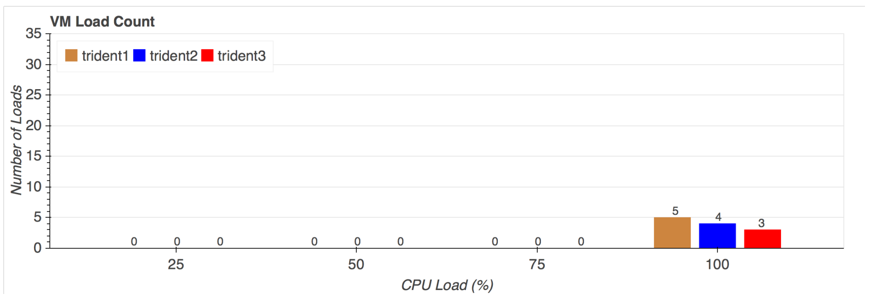

Figure 5.29: Uniform workload distribution after thermal balancing

5.2.2 Uniform Workload Analysis

The results from the implemented learn to balance algorithm, was conducted with 12 VMs provisioned on single server. The VMs have handled workloads that occupied 100% of CPU utilization. The implemented autonomous VM migration has maintained the expected thermal balance among all servers in around 6 hours.

The convergence time and the maximum temperature differences in the server cluster is given in Table 5.5.

Figure 5.30 shows the learning automata probability of a single VM. The VMs learned their environment by trial and error that brought the thermal balance among servers.

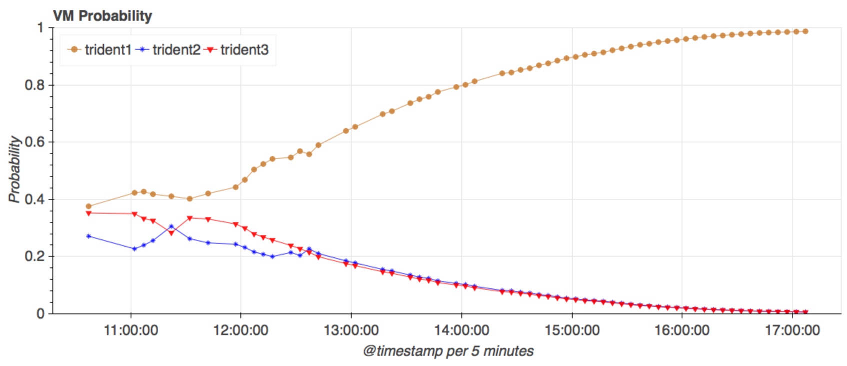

Figure 5.30: Learning Automata Probabilities

Box plot representation is also included in Figures 5.31 and 5.32 to show the temperature variation before and thermal balance attained after.

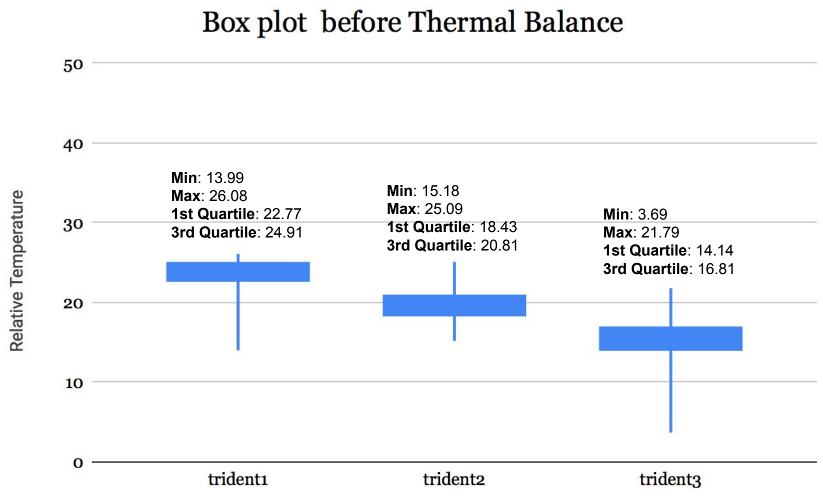

Figure 5.31: Box plot for Temperature Difference with Uniform Workload

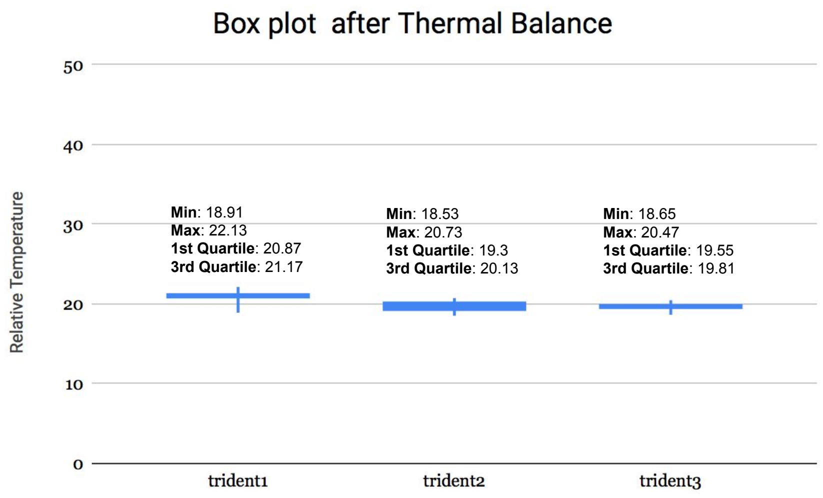

Figure 5.32: Box plot for Temperature Difference with Uniform Workload

5.2.3 Variable Workload Results

The experiment was conducted on VMs with uniform workload provisioned from the servers with a distribution of workloads shown in Figure 5.33. The total number of VMs was 12 as in the case for uniform workload. The results from this experiment are provided in Figures 5.34, 5.35, 5.36 and 5.37 representing the number of running VMs, temperature readings, final load distribution after thermal balance and power consumption readings respectively.

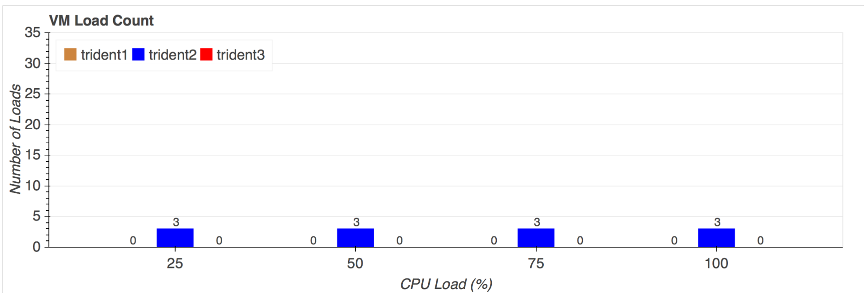

Figure 5.33: Initial variable workload distribution

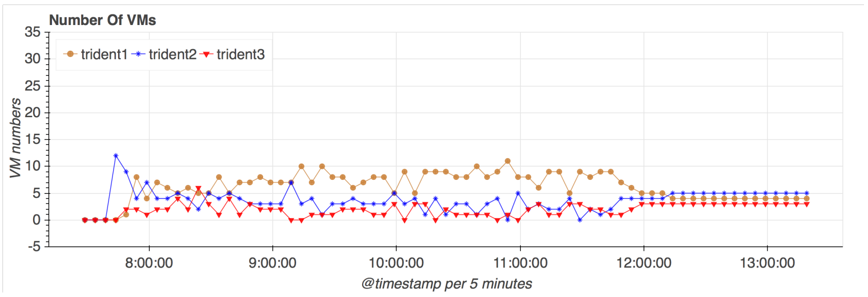

Figure 5.34: Number of running VMs with variable workload

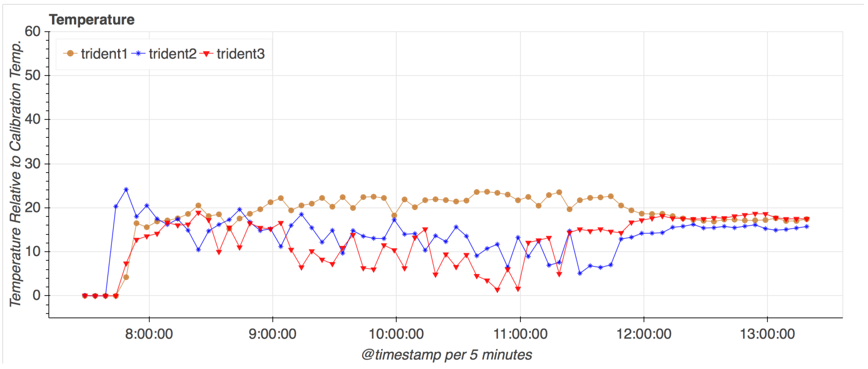

Figure 5.35: Temperature readings with variable workload

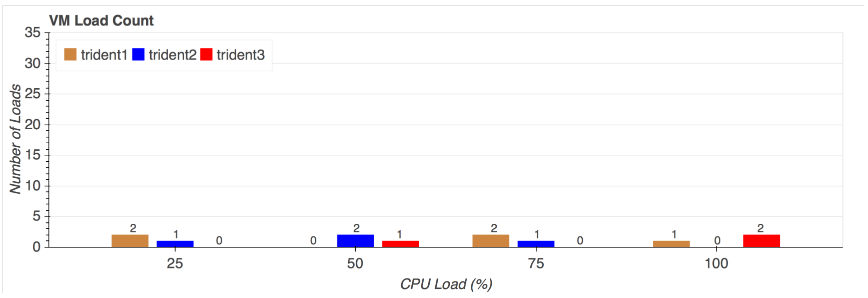

Figure 5.36: Variable workload distribution after thermal balancing

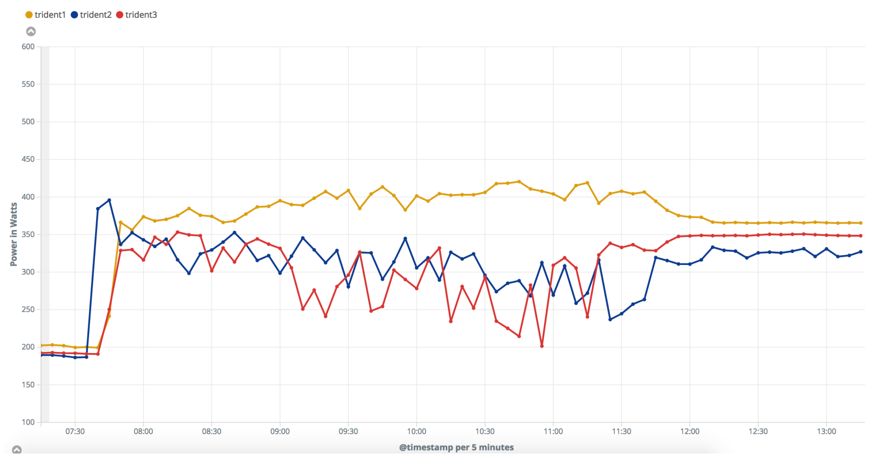

Figure 5.37: Power consumption readings with variable Workload

5.2.4 Variable Workload Analysis

The results from implementation of the learn to balance algorithm showed that, 5 hours were spent by the 12 VMs to learn their environment, by making autonomous migration choices, that lead the servers to attain thermal balance.

Table 5.5 presents the time taken to reach to the thermal balance and the maximum temperature differences between the servers before and after the thermal balance.

Figure 5.38 shows the learning automata probability of a single VM. The VMs learned their environment by trial and error, and its convergence is slow.

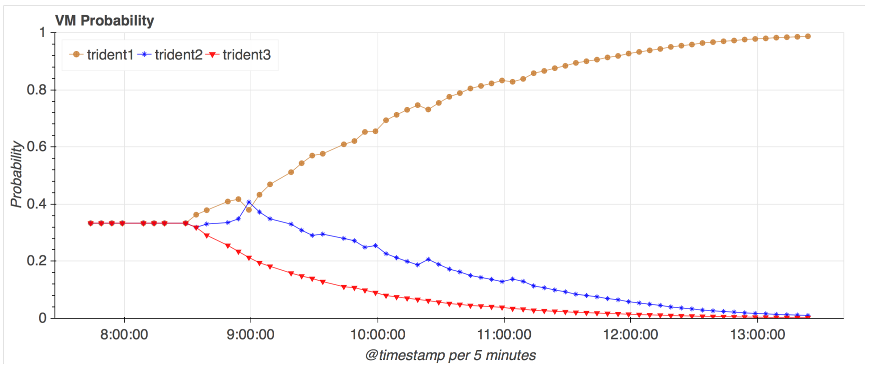

Figure 5.38: Learning Automata Probabilities

The box plot representation in Figures 5.39 and 5.40 shows the temperature variations before and after the thermal balance.

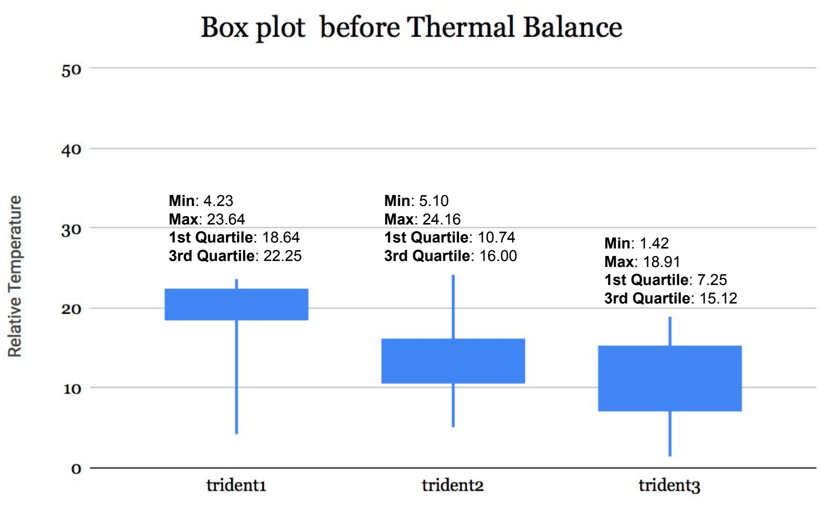

Figure 5.39: Box plot for Temperature Difference with Variable Workload

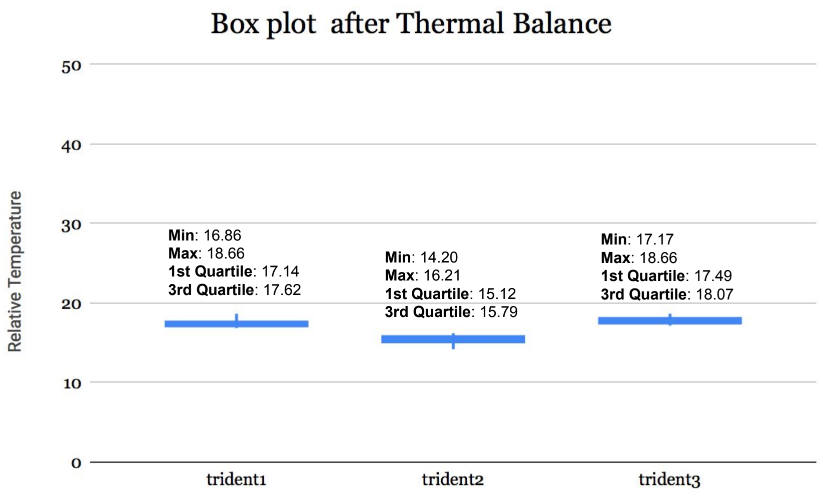

Figure 5.40: Box plot for Temperature Difference with Variable Workload

5.3 Algorithms Comparison

Since the experiment with learn to balance algorithm was executed using 12 VMs, in order to compare the result with the choose coolest server algorithm, there was additional experiment conducted. Appendix section C subsections C.1 and C.2 presents the results from this additional experiment using 12 VMs handling uniform and variable workload distributions respectively.

This section provides a comparison between the two implemented algorithms on the conducted experiments with 12 VMs. Table 5.4 and 5.5

shows the summary of time taken for attaining thermal balance and temperature difference of the servers.

Choose Coolest Server Algorithms		
Workload Type	Uniform	Variable
Number of Running VMs	12	12
Convergence Time(mins)	35	35
Relative Temperature Diff. before thermal balance	26.6	23.7
Relative Temperature Diff. after thermal balance	3.4	3.9

Table 5.4: Analysis: Choose Coolest Server with Uniform and Variable Workloads

Same way as the previous experiments, the results showed the attained thermal balance among the servers by the autonomous migration of VMs.

The experiment with choose coolest server algorithm took around 35 minutes to attain the thermal balance. Whereas, the learn to balance algorithm took 360 minutes for the same experiment. When it comes to the relative temperature differences, learn to balance algorithm achieved the lowest values.

Learn to Balance Algorithms		
Workload Type	Uniform	Variable
Number of Running VMs	12	12
Convergence Time(mins)	360	309
Relative Temperature Diff. before thermal balance	28.4	22.8
Relative Temperature Diff. after thermal balance	1.4	1.9

Table 5.5: Analysis: Learn to Balance with Uniform and Variable Workloads

Box plot representation of the variation in temperature before and after the maintained thermal balance are shown in Figures 5.43 and 5.44 for the variable workloads and 5.41 and 5.42 for the uniform workload using choose coolest server algorithm.

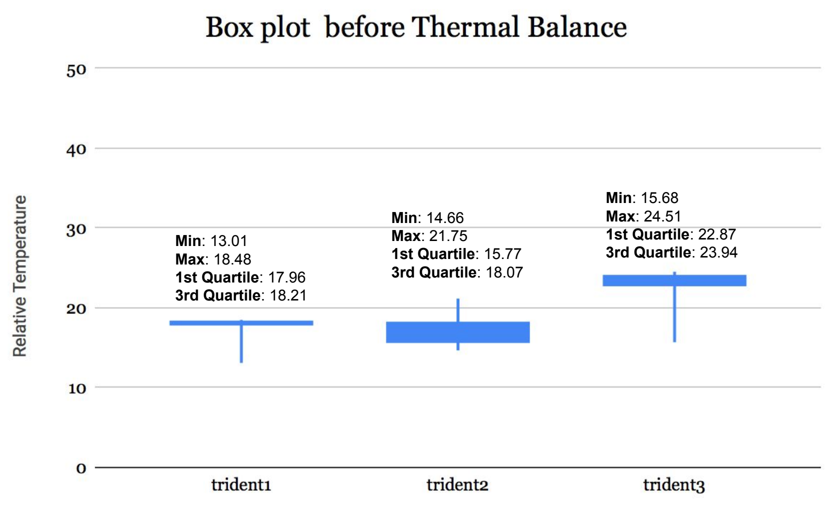

Figure 5.41: Box plot for Temperature Difference with Uniform Workload

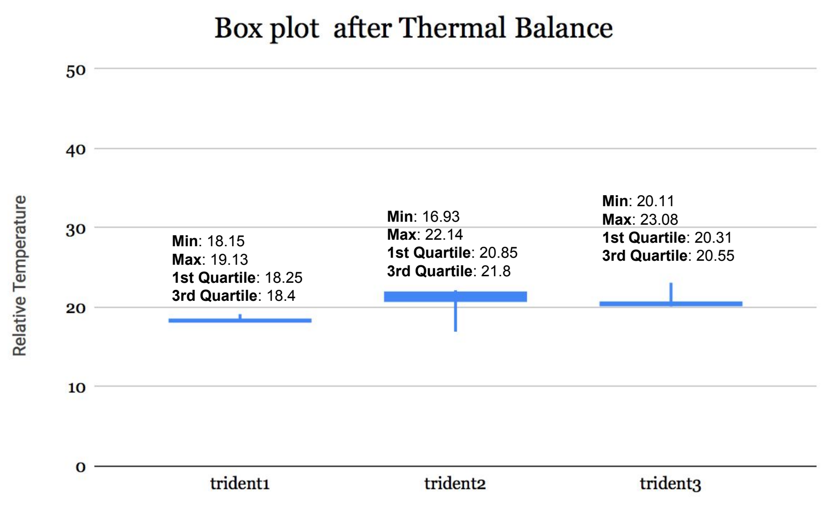

Figure 5.42: Box plot for for Temperature Difference with Uniform Workload

Since the number of VMs used for the experiment were small, provisioning was done from a single server. Thus, temperature imbalances between the servers was created, besides it made the spread of VMs quite noticeable.

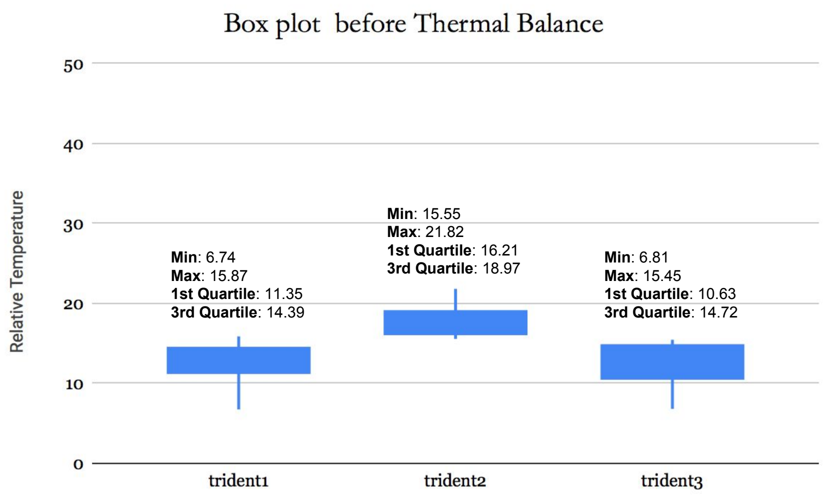

Figure 5.43: Box plot for Temperature Difference with Variable Workload

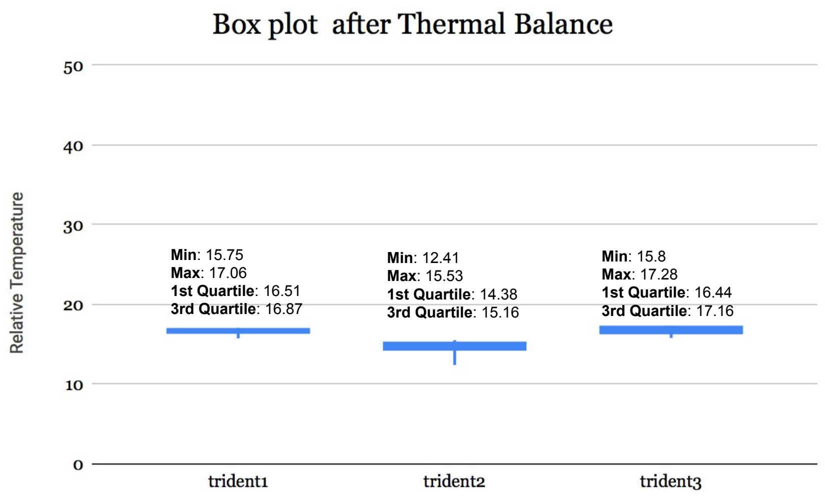

Figure 5.44: Box plot for for Temperature Difference with Variable Workload

Box plot representation of a data gives a summary of distributions in a data set. The data set to be compared in this thesis is relative temperature readings of servers. In the box plot, *min* and *max* values are the minimum and maximum temperature readings that appeared in the experiment. *Q1* shows 25% of the readings fall below it, whereas *Q3* shows 75% of the readings fall below it. Finally, the *median* is the mid point of the readings.

Tables 5.6 and 5.7 are compiled box plot presentations when the experiments were carried out implementing the two algorithms with uniform and variable workloads respectively.

Algorithms	**Q1**	**Min.**	**Median**	**Max.**	**Q3**
Choose Coolest Server	18.25	18.15	18.34	19.13	18.40
Learn to Balance	20.87	18.91	20.98	22.13	21.17

Table 5.6: Comparison of Algorithms: Box plot Analysis for Uniform Workload

Algorithms	**Q1**	**Min.**	**Median**	**Max.**	**Q3**
Choose Coolest Server	16.51	15.75	16.70	17.06	16.87
Learn to Balance	17.14	16.86	17.29	18.66	17.62

Table 5.7: Comparison of Algorithms: Box plot Analysis for Variable Workload

Part III

Conclusion

Chapter 6

Discussion and Conclusion

This chapter aims to give the full picture of the project. It is based on the initial problem statement and how it was addressed and answered through the whole process of the thesis work. Furthermore, the issues encountered in accomplishing the work and what can be done in the future based on the findings will be presented.

6.1 Background

Lots of researches have been studied and designs implemented to reduce the heat disposal in data centers. This include infrastructure level solutions, such as smart data center design [34], system level improvements, such as temperature aware workload placement [30] and IT level solutions such as virtualization.

Virtualization plays a huge role in improving data center power and heat management with all the features it posses. In relation to workloads servers handle, the existence of VMs ease hardware and maintenance cost minimization. Furthermore, VMs can be used for services and applications both in cloud and physical data centers.

In addition, virtualization provides a way to manage physical servers based on the VMs they hosted by implementing a consolidation or migration of VMs from one server to another as intended. By doing so, a complete shutdown on some of the physical servers is even possible, which highly minimize the cooling cost and intern reduce the total power consumption. This will enhance failure and emergency governance [30] and also increases the reliability of the whole infrastructure.

The goal of this thesis was to attain a thermal balance in a server cluster, using VMs that are configured to do an autonomous migration based on the hosting servers temperature readings. Autonomous VM migration works in decentralized approach, where the drawbacks of implementing centralized approach such as, single point of failure and resource competition can be eliminated.

Standing out advantage of thermal balancing as a thermal management technique is, its potential to balance temperature of servers by making sure there is no overheated one among them. And good enough, it does this without performance downsides and it can be applied on heterogeneous environments even though this advantage has not been experimented in this thesis.

In data centers, both load balancing and thermal balancing should go hand in hand. Because, in most of the cases assuring a balanced workload among servers might not necessarily create a thermal balance among them. In a typical data center, servers temperature vary in relation to their physical placement. This is due to the fact that exhaust air temperatures of racks at the end of a row are higher than the row's middle rack [38].

Having the thesis goal in mind, a research has been done to look for different approaches and two approaches were considered for further implementation. The first approach aimed to test the VMs ability to make the autonomous migration choices independently by selecting destination server based on global information, whereas the second approach does this by trial and error to learn the environment with only partial information provided to it.

6.2 Problem Statement

The problem statement this project tried to address was "*How to achieve thermal balance on virtualized server cluster by autonomous migration of Virtual Machines hosted on them, based on the servers temperature readings*". The state of the art in thermal balancing was revised, and two algorithms were developed to do an autonomous VM migration in order to attain a thermal balance in a server cluster.

The introduction chapter have set questions that needed to be answered through the process of this thesis. This section provides how they were answered through the design and implementation of the proposed solutions.

- **How to remaster a custom VM based on light weight Linux distribution with a given workload?**

  The question of doing an autonomous migration was one of the key tasks of this thesis. The core VM from the core project described in section 2.7.1 was selected to do the task for two reasons:

  (i) Its ability to be customized.

  (ii) Its small size.

After the Core VM was selected, the next task was to search for a virtualization technology and VM management tool. KVM described in section 2.3, for virtualization and Libvirt described in section 2.2, for VM management has been found appealing for their handy features and vast management utilities.

Afterwards, remastering of the custom Core VM has been accomplished using a remastering tool described in section 2.7.2, with all the necessary packages stated in section 3.2.1 and scripts attached in Appendix section B. The workloads were configured with multiple CPU utilization percentages to show a realistic workload distribution using the stress-ng tool, built from source code, described in section 4.3.3.

In order to provision VMs from different servers in the cluster, each VMs had to be assigned unique MAC and IP address. This has been accomplished by editing the VM virtual bridge interface's DHCP range of each server as stated in section 4.3.4. The requirement was to only specify range of reserved IPs at the servers so that the given IPs to the VMs by Libvirt does not clash when the VMs migrate.

- **How to equip VMs with autonomous decision making capability?**

  Two approaches were implemented based on full and partial temperature information of all servers in the cluster. By the implemented autonomous VM migration, the VMs were supposed to make decisions independently and without a centralized controller and the implemented algorithms were able to give this intelligence to the VMs.

  The first algorithm implemented was choose coolest server, that had used temperature readings of all servers in the cluster. Based on those readings, the VMs compute average temperature independently and migrates to the coolest server if it is hosted on a server with higher temperature than the calculated average temperature, otherwise it will stay were it was.

  The second algorithm used a reinforcement learning technique, where VMs learn their environment using a distributed learning automata. The learning automata concept was adopted from the active probing and learning algorithm provided in [25].

  At the beginning, the learning automata probabilities were initialized uniformly. Then VMs migrate independently to chosen destination server randomly, according to their learning automata probability. The visited server's temperature is then used to reinforce this chosen server, by increasing the probability of choosing it again. Simultaneously, the probability of migrating to the rest of the servers in the cluster will be decreased. The reinforcement technique allows the servers with low temperature to be visited too often. Besides, since all VMs runs independent learning automata locally, it is easily scalable.

- **How to monitor and visualize VMs activities in server cluster in real time?**

  In order to monitor and visualize the temperature readings, number of running VMs and workload distributions, Bokeh tool was configured and used as seen from the experimental results gathered in chapter 5. The average temperature readings from the 8 sensors placed inside NUMA nodes of the servers has been plotted. Above all, the real time monitoring made it easy to track what has happened in terms of the monitored parameters in the server cluster. The power consumption visualizations, on the other hand, were collected from Kibana web application that was already in place at Metropolitan University (OsloMet).

6.3 Results and Analysis

The experimentation phase employed the above mentioned design and implementation core concepts and took place in many distinct test runs. The final results presented in the results and analysis chapter, chapter 5 are outcomes of this diverse experiments. As the goal of this thesis requires balancing temperatures of virtualized servers by doing autonomous VM migration, the results will be evaluated with regard to achieving them.

The implemented approaches where tested on three physical servers housed in a server room at OsloMet. These servers were placed on top of each other in a rack with other servers. As part of the experiment, the physical placement of the servers was swapped to see if that would have an effect on the power consumption readings. When this thesis started, there was a huge power consumption difference between the servers and exchanging the placement helped in minimizing that difference. But still the server at the top i.e, *trident1* is the most consuming one compared to the other two, even in an idle situation as seen in Figure 5.2.

When it comes to the VMs, the customized VMs in this thesis were able to make autonomous migration decisions. In the real world, when choosing cloud service providers, it is crucial to consider various aspects including pricing and security. Whenever a customer is not satisfied with the services provided by the service providers at data centers, it would be easier to migrate services across different platforms if the ability to do autonomous VM migration is applied.

Furthermore, three types of workloads were used, dynamic, variable and uniform. Dynamic workloads used, as shown in section 5.1.6, were to test if the implemented algorithms can function well with changing environments. In data centers, handled workloads by servers differ from one another, in addition there is a varying workload based on time and service that servers provide. For example, web servers are usually idle in the day time and highly utilized during night. Therefore, experimenting with a dynamic workload distributions reflects the real deal at data centers.

Since the concern of this thesis is not load balancing, rather balancing the temperatures, it was also necessary to test it with variable workloads as well, as presented in Figure 5.14. This also reflects the diverse workload that servers handle in data centers.

The uniform workload on the other hand was set to test the maximum utilization of the servers with full percentage CPU utilization, which might be the case occasionally even in data centers.

In addition, these workloads were distributed, by running different number of VMs on the servers as seen in Figures 5.7, 5.15, 5.22, 5.27 and 5.34, so as to create a noticeable temperature imbalances on the servers when the experiments were conducted.

The experimental results showed that the goal of attaining a thermal balance in a server cluster using VMs capable of making autonomous migration was a success story from the conducted experiments.

This achievement can scale and be used in real data centers. The autonomous VM migration can highly benefit and improve data centers VM management. Regarding the thermal balance, since the temperature readings used in this thesis were physical temperatures, they do not provide accurate temperature readings of the servers. Therefore, the same approach can be used and tested with data from real temperature monitoring tools in a data center.

When this thesis was started there was a hope that it would reduce power consumption, but it turns out there was no significant promotion or reduction in the power consumption. This was because the used CPU temperatures were not able to relate to the monitored power consumption readings, in addition there was a limit on the number of VMs used, that prevented conducting experiments with higher than 45 VMs, as there was unstable server in the cluster.

Nonetheless, the two algorithms have ensured thermal balancing by autonomous VM migration. They both have got their own advantages and disadvantages as well. The next section provides a comparison between them.

6.4 Algorithm Comparison

The experimental results showed that choose coolest server algorithm converges faster than the learn to balance algorithm. Appendix section C subsection C.8 and Figure 5.35 are evident for this. Since the choose coolest server algorithm provide the VMs full knowledge of their environment, it had been easier to make the right decisions and reach to a thermally balanced state faster.

The learn to balance was an advanced algorithm based on reinforcement learning technique. The partial information provided for the VMs brought an issue when it comes to convergence, since the VMs were moving with trial and errors to learn their environment, in order to reach at a balanced state.

Another parameter to compare the algorithms with is, the maximum temperature difference between the servers after the attained thermal balance. With this regard, the results showed a bit higher temperature differences among the servers compared to the learn to balance algorithm as given in the box plot representation Figures 5.42 and 5.32 captured for the uniform workload distribution with 12 VMs. keep in mind that, the choose coolest server algorithm used time varying delta temperature value, in which VMs stay where they are when a specific temperature difference was reached and thermal balance was attained.

The temperature difference when the thermal balance was attained and the convergence time taken for both algorithms can be seen from the summary provided in Tables 5.4 and 5.5 for variable and uniform workload distributions with 12 VMs respectively.

6.5 Future Work

Since data centers are increasing in number and size, with all the growing server density and all the huge number of facilities they comprehend, there is a huge demand for thermal management. On this thesis thermal balancing by autonomous VM migration has been experimented.

Another thing to consider would be CPU utilization. Focusing only on the CPU temperature readings was not good enough, because the thermal effect stays unchanged even after some VMs has migrated already. As a future work, if CPU utilization and Temperature readings would be considered together in the design of the autonomous migration, that will address the issue experimented in this thesis from multiple perspectives.

In addition, the focus of this thesis was only on autonomous migration of VMs in a server cluster and it doesn't consider the option of scheduling VMs on NUMA nodes of the server. This was left to the OS scheduler and any future work might consider that.

6.6 Conclusion

The goal of this thesis was to attain a thermal balance in a virtualized server cluster, by doing an autonomous migration of VMs, based on real time CPU temperature readings of the servers.

This thesis has proposed two algorithms, choose coolest server and learn to balance, in order to equip VMs with an intelligence to do autonomous migration decisions to maintain a thermal balance in a server cluster.

Multiple experiments conducted have revealed the VMs ability to do an autonomous migration to attain a thermal balance among servers in the cluster. Among the two implemented algorithms, the choose coolest server algorithm has been found better in terms of time it take to attain a thermal balance. In contrary, the learn to balance algorithm has shown a better performance with regard to thermal variation among servers in the cluster.

Bibliography

[1] Mariana Agache and B John Oommen. 'Generalized pursuit learning schemes: New families of continuous and discretized learning automata'. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 32.6 (2002), pp. 738–749.

[2] Kishwar Ahmed et al. 'Online Resource Management for Carbon-Neutral Cloud Computing'. In: *Handbook on Data Centers*. Springer, 2015, pp. 607–630.

[3] *AMD Opteron™ 6000 Series Platform*. URL: https://products.amd.com/en-us/search/CPU/AMD-Opteron%E2%84%A2/AMD-Opteron%E2%84%A2-6200-Series-Processor/6234/33 (visited on 11/02/2018).

[4] *AMD Temp Information and Guide*. URL: http://www.overclock.net/forum/10-amd-cpus/1128821-amd-temp-information-guide.html (visited on 29/09/2011).

[5] Heni Ben Arab. 'Virtual Machines Live Migration'. In: *PDF document. Available at* (2017).

[6] Maria Avgerinou, Paolo Bertoldi and Luca Castellazzi. 'Trends in Data Centre Energy Consumption under the European Code of Conduct for Data Centre Energy Efficiency'. In: *Energies* 10.10 (2017), p. 1470.

[7] Jeff Beckham. *Cloud Computing vs. Virtualization: The Differences and Benefits*. URL: https://blogs.cisco.com/smallbusiness/cloud-computing-vs-virtualization-the-differences-and-benefits (visited on 04/02/2018).

[8] Jeffrey Kramer Ben Cutler Spencer Fowers and Eric Peterson. 'Want an Energy-Efficient Data Center? Build It Underwater'. In: (2017).

[9] Bokeh. *User Guide to Bokeh*. URL: https://bokeh.pydata.org/en/latest/docs/user_guide.html (visited on 04/03/2018).

[10] Susanta Nanda Tzi-cker Chiueh and Stony Brook. 'A survey on virtualization technologies'. In: *Rpe Report* 142 (2005).

[11] Hyung Won Choi et al. 'Autonomous learning for efficient resource utilization of dynamic vm migration'. In: *Proceedings of the 22nd annual international conference on Supercomputing*. ACM. 2008, pp. 185–194.

[12] Hyung Won Choi et al. 'Enabling Scalable Cloud Infrastructure Using Autonomous VM Migration'. In: *High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), 2012 IEEE 14th International Conference on*. IEEE. 2012, pp. 1066–1073.

[13] Christopher Clark et al. 'Live migration of virtual machines'. In: *Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation-Volume 2*. USENIX Association. 2005, pp. 273–286.

[14] Amos Waterland Colin King. *How To Stress Test CPU and Memory (VM) On a Linux and Unix With Stress-ng*. URL: https://www.cyberciti.biz/faq/stress-test-linux-unix-server-with-stress-ng/ (visited on 28/03/2018).

[15] Amos Waterland Colin King. *stress-ng Manual*. URL: http://kernel.ubuntu.com/~cking/stress-ng/stress-ng.pdf (visited on 28/03/2018).

[16] Martin Duggan et al. 'An autonomous network aware vm migration strategy in cloud data centres'. In: *Cloud and Autonomic Computing (ICCAC), 2016 International Conference on*. IEEE. 2016, pp. 24–32.

[17] Richard Evans and Jim Gao. 'DeepMind AI Reduces Google Data Centre Cooling Bill by 40%'. In: (2016).

[18] Yong Fu, Chenyang Lu and Hongan Wang. 'Robust control-theoretic thermal balancing for server clusters'. In: *Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on*. IEEE. 2010, pp. 1–11.

[19] Brice Goglin and Nathalie Furmento. 'Enabling high-performance memory migration for multithreaded applications on linux'. In: *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*. IEEE. 2009, pp. 1–9.

[20] Whitson Gordon. *How to Monitor Your Computer's CPU Temperature*. URL: https://www.howtogeek.com/howto/windows-vista/ever-wonder-what-temperature-your-cpu-is-running-at/ (visited on 21/02/2018).

[21] Martijn Groot. *A Primer in Financial Data Management*. Academic Press, 2017.

[22] Chung-hsing Hsu and Wu-chun Feng. 'A power-aware run-time system for high-performance computing'. In: *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*. IEEE Computer Society. 2005, p. 1.

[23] Bernadette Johnson. *How Data Centers Work*. URL: https://computer.howstuffworks.com/data-centers.htm.

[24] Colin King. *Stress-ng*. URL: http://kernel.ubuntu.com/git/cking/stress-ng.git/ (visited on 28/03/2018).

[25] Hong Li, Lorne Mason and Michael Rabbat. 'Learning minimum delay paths in service overlay networks'. In: *Network Computing and Applications, 2008. NCA'08. Seventh IEEE International Symposium on*. IEEE. 2008, pp. 271–274.

[26] Vaishakhi Maheshwari and Mohit Patel. 'Live Migration using VM/TPM Protocol of Virtual Machine on Private Cloud'. In: (2017).

[27] Nakul Manchanda and Karan Anand. 'Non-uniform memory access (numa)'. In: *New York University* 4 (2010).

[28] Daniel A Menascé. 'Virtualization: Concepts, applications, and performance modeling'. In: *Int. CMG Conference*. 2005, pp. 407–414.

[29] Mayank Mishra et al. 'Dynamic resource management using virtual machine migrations'. In: *IEEE Communications Magazine* 50.9 (2012).

[30] Justin D Moore et al. 'Making Scheduling" Cool": Temperature-Aware Workload Placement in Data Centers.' In: *USENIX annual technical conference, General Track*. 2005, pp. 61–75.

[31] Justin Moore and Parthasarathy Ranganathan. *Thermal management of data centers*. US Patent 7,620,613. Nov. 2009.

[32] Justin Moore et al. 'Going beyond CPUs: The potential of temperature-aware solutions for the data center'. In: *Proc. 2004 First Workshop Temperature-Aware Computer Systems (TACS-1) Held in Conjunction with ISCA-31*. 2004.

[33] Luca Parolini, Bruno Sinopoli and Bruce H Krogh. 'Reducing data center energy consumption via coordinated cooling and load management'. In: *Proceedings of the 2008 conference on Power aware computing and systems, HotPower*. Vol. 8. 2008, pp. 14–14.

[34] Chandrakant D Patel et al. 'Smart cooling of data centers'. In: *ASME 2003 International Electronic Packaging Technical Conference and Exhibition*. American Society of Mechanical Engineers. 2003, pp. 129–137.

[35] The Core Project. *Ezremaster Overview*. URL: http://wiki.tinycorelinux.net/wiki:remastering_with_ezremaster (visited on 14/02/2018).

[36] The Core Project. *Introduction to Core*.

[37] Disha Sangar. 'Will You Carry Me'. MA thesis. 2017.

[38] Ratnesh K Sharma et al. 'Balance of power: Dynamic thermal management for internet data centers'. In: *IEEE Internet Computing* 9.1 (2005), pp. 42–49.

[39] James E Smith and Ravi Nair. 'The architecture of virtual machines'. In: *Computer* 38.5 (2005), pp. 32–38.

[40] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. Vol. 1. 1. MIT press Cambridge, 1998.

[41] Richard S Sutton and Andrew G Barto. 'Reinforcement learning: An introduction'. In: (2017).

[42] Yuya Tarutani et al. 'Reducing power consumption in data center by predicting temperature distribution and air conditioner efficiency with machine learning'. In: *Cloud Engineering (IC2E), 2016 IEEE International Conference on*. IEEE. 2016, pp. 226–227.

[43] Morgan Tatchell-Evans et al. 'An experimental and theoretical investigation of the effects of supply air conditions on computational efficiency in data centers employing aisle containment'. In: *Thermal Measurement, Modeling & Management Symposium (SEMI-THERM), 2017 33rd*. IEEE. 2017, pp. 100–107.

[44] *The JSON Data Interchange Syntax*. URL: http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf (visited on 12/2017).

[45] Mueen Uddin and Azizah Abdul Rahman. 'Server consolidation: An approach to make data centers energy efficient and green'. In: *arXiv preprint arXiv:1010.5037* (2010).

[46] Fredrik Meyn Ung. 'Towards efficient and cost-effective live migrations of virtual machines'. MA thesis. 2015.

[47] Anis Yazidi. 'Intelligent learning automata-based strategies applied to personalized service provisioning in pervasive environments'. In: (2011).

[48] Jie Zheng et al. 'Comma: Coordinating the migration of multi-tier applications'. In: *ACM SIGPLAN Notices*. Vol. 49. 7. ACM. 2014, pp. 153–164.

Appendices

Appendix A

Environmental Setup

A.1 Creating the customized Core .iso

```
#!/bin/sh

# Install openssh and ezremaster
tce-load -iw openssh.tcz ezremaster.tcz python.tcz
 ↪ python-numpy.tcz python3.4.tcz

# Start the SSH server
sudo /usr/local/etc/init.d/openssh start

# wget the Core-current.iso file to /tmp
wget
 ↪ http://distro.ibiblio.org/tinycorelinux/6.x/x86/release/Core-current.iso
 ↪ -P /tmp

# ezremaster walkthrough
read -p "Open ezremaster. Click on the ezremaster icon (looks
 ↪ like a CD with 'ez' on it) at the bottom of the screen."
read -p "Use ISO Image, specifying the /tmp/Core-current.iso
 ↪ file"
read -p "Next, Next"
read -p "Click load under the 'Extract TCZ in to initrd'"
read -p "Remove everything except openssh.tcz"
read -p "Next until you can Create ISO (BUT DON'T CREATE ISO
 ↪ YET)"
read -p "Press Enter to continue..."

###########################
# Modifying the new build #
###########################

# Build stress-ng
sudo cp /tmp/stress-ng /tmp/ezremaster/extract/usr/bin/
```

```
# Edit the isolinux.cfg file to change the boot timeout from
  300 (30 seconds) to 10 (1 second)
sudo cp /tmp/ezremaster/image/boot/isolinux/isolinux.cfg
  /tmp/ezremaster/image/boot/isolinux/isolinux.cfg.backup
sudo sed -i 's/timeout 300/timeout 10/'
  /tmp/ezremaster/image/boot/isolinux/isolinux.cfg

# Add the SSH keys generated when TinyCore installed SSH.  Not
  required, but otherwise every reboot will generate new
  keys.
sudo cp -f /usr/local/etc/ssh/ssh_host_*
  /tmp/ezremaster/extract/usr/local/etc/ssh

# Edit the SSH server configuration
sudo cp /tmp/ezremaster/extract/usr/local/etc/ssh/sshd_config
  /tmp/ezremaster/extract/usr/local/etc/ssh/sshd_config.backup
sudo sed -i 's/#PermitRootLogin/PermitRootLogin/'
  /tmp/ezremaster/extract/usr/local/etc/ssh/sshd_config
sudo sed -i 's/#GatewayPorts no/GatewayPorts yes/'
  /tmp/ezremaster/extract/usr/local/etc/ssh/sshd_config

# Ensure the correct file permissions for the SSH keys
sudo chown root
  /tmp/ezremaster/extract/usr/local/etc/ssh/ssh_host*
sudo chmod 644
  /tmp/ezremaster/extract/usr/local/etc/ssh/ssh_host*pub
sudo chmod 600
  /tmp/ezremaster/extract/usr/local/etc/ssh/ssh_host*key

# Start the SSH server on boot
sudo cp /tmp/ezremaster/extract/opt/bootlocal.sh
  /tmp/ezremaster/extract/opt/bootlocal.sh.backup
sudo echo "/usr/local/etc/init.d/openssh start" >>
  /tmp/ezremaster/extract/opt/bootlocal.sh

# Start custom workload on boot
sudo cp /tmp/workload.py /tmp/ezremaster/extract/opt/
sudo chmod +x /tmp/ezremaster/extract/opt/workload.py
sudo echo "/opt/workload.py &" >>
  /tmp/ezremaster/extract/opt/bootlocal.sh

sudo cp /tmp/client.py /tmp/ezremaster/extract/opt/
sudo chmod +x /tmp/ezremaster/extract/opt/client.py
sudo echo "/opt/client.py &" >>
  /tmp/ezremaster/extract/opt/bootlocal.sh

sudo cp /tmp/__init__.py /tmp/ezremaster/extract/opt/
sudo cp /tmp/rl.py /tmp/ezremaster/extract/opt/
sudo cp /tmp/utils.py /tmp/ezremaster/extract/opt/

# Put bootlocal in .filetool.lst
sudo echo "/opt/bootlocal.sh" >> /tmp/ezremaster/extract/opt/.filetool.lst

# Give the "tc" user a password
passwd tc

# Change root user password
sudo passwd root

# Copy the /etc/shadow & /etc/passwd files (which have the new tc and root passwords) from the current TinyCore operating system to the new Core build
sudo cp -f /etc/shadow /tmp/ezremaster/extract/etc/shadow
sudo cp -f /etc/passwd /tmp/ezremaster/extract/etc/passwd

read -p "Now click on Create ISO...script is done. File location: /tmp/ezremaster/ezremaster.iso"
```

Appendix B

Developed Scripts

B.1 client.py

```
#!/usr/local/bin/python
# -*- coding: utf-8 -*-

import socket
import json
import time
import random
from copy import deepcopy
from utils import *
from rl import RlAgent

''' Client
'''
class Client(object):
def __init__(self):
self.rlAgent = RlAgent()
self.sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
self.port = 10000
self.client_message = deepcopy(CLIENT_MESSAGE)
self.client_data = {
        "hostName" : "",
        "ip" : "",
        "mac" : "",
        "load" : 0
}

def run(self):
# Get Client data
self._getClientData()

# Wait 5 minutes
time.sleep(300)
```

```python

while True:
# VM wakes randomly
rand_time = random.randint(60, 180)
print("Rand Time = {}".format(rand_time))
time.sleep(rand_time)

# Choose action
action = self.rlAgent.takeAction()
migrate = True if action != self.client_data["hostName"] else
 ↪  False
print("migrate? = {}, action = {}".format(migrate, action))

# Migrate
if(migrate):
        self.client_message["request"]["login"] = False
        self.client_message["request"]["migration"] = True
        self.client_message["request"]["temperature"] = False
        self.client_message["vm"]["mac"] =
         ↪  self.client_data["mac"]
        self.client_message["vm"]["target"] = action
        self.client_message["vm"]["load"] =
         ↪  self.client_data["load"]
        self.client_message["prob"]["trident1.vlab.cs.hioa.no"]
         ↪  = self.rlAgent.prob["trident1"]
        self.client_message["prob"]["trident2.vlab.cs.hioa.no"]
         ↪  = self.rlAgent.prob["trident2"]
        self.client_message["prob"]["trident3.vlab.cs.hioa.no"]
         ↪  = self.rlAgent.prob["trident3"]
        self.sock.sendto(json.dumps(self.client_message).encode('utf-8'),
         ↪  (self.client_data["ip"], self.port))
        print("sent: {} to {}".format(self.client_message,
         ↪  action))

        # Sleep for 60 sec, for migration to complete
        time.sleep(60)
else:
        print("VM choose to stay")

try:
        # Update Client status
        self._getClientData()

        # Request Temperature
        self.client_message["request"]["login"] = False
        self.client_message["request"]["temperature"] = True
        self.client_message["request"]["migration"] = False
        self.client_message["vm"]["mac"] =
        ↪  self.client_data["mac"]
        self.client_message["vm"]["target"] =
        ↪  self.client_data["hostName"]
        self.client_message["vm"]["load"] =
        ↪  self.client_data["load"]
        self.client_message["prob"]["trident1.vlab.cs.hioa.no"]
        ↪  = self.rlAgent.prob["trident1"]
        self.client_message["prob"]["trident2.vlab.cs.hioa.no"]
        ↪  = self.rlAgent.prob["trident2"]
        self.client_message["prob"]["trident3.vlab.cs.hioa.no"]
        ↪  = self.rlAgent.prob["trident3"]
        self.sock.sendto(json.dumps(self.client_message).encode('utf-8'),
        ↪  (self.client_data["ip"], self.port))
        print("sent: {}".format(self.client_message))

        # Receive response
        print("waiting to receive")
        self.sock.settimeout(15.0)
        data, server = self.sock.recvfrom(1024)
        self.sock.settimeout(None)
        server_message = json.loads(data.decode('utf-8'))
        print("received: {} from {}".format(server_message,
        ↪  server))

        # Learn
        maxTemp = server_message["maxTemp"]
        hostTemp = server_message["hostTemp"]
        self.rlAgent.learn(action, maxTemp, hostTemp)
except:
        print("Socket timeout")

def _getClientData(self):
# Send client data request
while True:
try:
        # Get client data
        self.client_data["hostName"] = getHostName()
        self.client_data["ip"] =
        ↪  getHostIp(self.client_data["hostName"])
        self.client_data["mac"] = getVmMac()
        self.client_data["load"] = getLoad()
        print('VM is at: {}, on
        ↪  {}'.format(self.client_data["hostName"],
        ↪  (self.client_data["ip"], self.port)))

        # Send Request
        self.client_message["request"]["login"] = True
```

```
                self.client_message["request"]["temperature"] = False
                self.client_message["request"]["migration"] = False
                self.client_message["vm"]["mac"] =
                ↪  self.client_data["mac"]
                self.client_message["vm"]["target"] =
                ↪  self.client_data["hostName"]
                self.client_message["vm"]["load"] =
                ↪  self.client_data["load"]
                self.client_message["prob"]["trident1.vlab.cs.hioa.no"]
                ↪  = self.rlAgent.prob["trident1"]
                self.client_message["prob"]["trident2.vlab.cs.hioa.no"]
                ↪  = self.rlAgent.prob["trident2"]
                self.client_message["prob"]["trident3.vlab.cs.hioa.no"]
                ↪  = self.rlAgent.prob["trident3"]
                self.sock.sendto(json.dumps(self.client_message).encode('utf-8'),
                ↪  (self.client_data["ip"], self.port))
                print("sent: {}".format(self.client_message))

                # Receive response
                print("waiting to receive login request")
                self.sock.settimeout(15.0)
                data, server = self.sock.recvfrom(1024)
                self.sock.settimeout(None)
                server_message = json.loads(data.decode('utf-8'))
                print("received: {} from {}".format(server_message,
                ↪  server))
                break
        except:
                print("Login Request Socket Timed out, Retrying ...")
                time.sleep(30)

'''
Main
'''
if __name__ == "__main__":
Client().run()
```

B.2 server.py

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

import socket
import json
import threading
import time
import sys
```

```python
import os
import signal
import logging
from copy import deepcopy

from utils import *
import calibrate

logging.basicConfig(filename="/var/tmp/server.log",
level=logging.DEBUG,
format="%(asctime)s:%(levelname)s:%(message)s")

''' Server
'''
class Server(object):
def __init__(self, initialTemp, maxTemp):
# Calibration temperature
self.calibrationTemp = initialTemp

# Register Signal Handler
signal.signal(signal.SIGINT, self._signal_handler)

# Server message containing max and host temperature
self.server_message = deepcopy(SERVER_MESSAGE)
self.server_message["maxTemp"] = maxTemp

# List of VMs running on server with their corresponging load
self.vms = {}

# Python create mutex
self.my_mutex = threading.Lock()

# Create a UDP socket and bind the socket to the port
self.client_socket = socket.socket(socket.AF_INET,
 ↪ socket.SOCK_DGRAM)
self.plot_socket = socket.socket(socket.AF_INET,
 ↪ socket.SOCK_DGRAM)

# Ports
self.client_port = 10000
self.plot_port = 10002

self.client_socket.bind(("", self.client_port))
self.plot_socket.bind(("", self.plot_port))

def _signal_handler(self, signal, frame):
logging.info('Signal Interrupt Caught!')
self.client_socket.close()
self.plot_socket.close()
sys.exit(0)

def __del__(self):
logging.debug("Server Exited")

def run(self):
# Handle login
loginThread = threading.Thread(target=self._handleLogin)
loginThread.start()

# Handle plot
plotThread = threading.Thread(target=self._handlePlot)
plotThread.start()

# Handle client
clientThread = threading.Thread(target=self._handleClient)
clientThread.start()

clientThread.join()
plotThread.join()
loginThread.join()

def _handleLogin(self):
while True:
logging.debug("---------------------- handleLogin
 ↪  ---------------")

# Delete VMs Loggedout
try:
vms = getVmsLoggedin()
self.my_mutex.acquire()
loggedoutVms = [vm for vm in self.vms if vm not in vms]
for vm in loggedoutVms:
logging.info("VM: {} Loggedout".format(vm))
del self.vms[vm]
self.my_mutex.release()
logging.debug("Logged in clients = {}".format(self.vms))
except:
logging.error("Exception in HandleLogin thread")
time.sleep(15)

def _handlePlot(self):
while True:
logging.debug("---------------------- handlePlot
 ↪  ---------------")
logging.info("waiting to receive server plot data request")
data, address = self.plot_socket.recvfrom(4096)
```

```
logging.info("Received server plot data request")

# Server plot data
try:
plotData = deepcopy(SERVER_PLOT_DATA)
hostTemp = (getHostTemp() - self.calibrationTemp) /
 ↪  getNumberOfNodes()
plotData["hostTemp"] = hostTemp if hostTemp >= 0.0 else 0.0
plotData["numVms"] = getNumberOfVms()

self.my_mutex.acquire()
for vm, value in self.vms.items():
plotData["vmLoads"][str(value["load"])] += 1
plotData["vms"].append(self.vms[vm])
self.my_mutex.release()

sent =
 ↪  self.plot_socket.sendto(json.dumps(plotData).encode('utf-8'),
 ↪  address)
logging.info("Sent {} to {}".format(plotData, address))
except:
logging.error("Exception in HandlePlot thread")

def _handleClient(self):
while True:
logging.debug("---------------------- handleClient
 ↪  ---------------")

# Get VM message
logging.debug("Waiting to receive client message")
data, address = self.client_socket.recvfrom(1024)
client_message = json.loads(data.decode('utf-8'))
logging.info("Received {} from {}".format(client_message,
 ↪  address))

try:
# Get VM Domain Name
vm = getVmName(client_message["vm"]["mac"])
vmLoad = client_message["vm"]["load"]

# Process Client Message
if vm != "":
if client_message["request"]["login"]:
# Register VM
if str(vmLoad) in SERVER_PLOT_DATA["vmLoads"]:
        logging.info("Added VM: {} Load: {}".format(vm,
         ↪  vmLoad))
        self.my_mutex.acquire()
        self.vms[vm] = {
                "vm" : vm,
                "load" : vmLoad,
                "prob" : client_message["prob"]
        }
        self.my_mutex.release()

        # Send Response
        sent = self.client_socket.sendto(json.dumps(client_message).encode('utf-8'), address)
        logging.info("Sent {} back to {}".format(client_message, address))
else:
        logging.error("VM Load not in SERVER_PLOT_DATA")
elif client_message["request"]["temperature"]:
# Send Response
hostTemp = getHostTemp() - self.calibrationTemp
self.server_message["hostTemp"] = hostTemp if hostTemp >= 0.0 else 0.0
sent = self.client_socket.sendto(json.dumps(self.server_message).encode('utf-8'), address)
logging.info("Sent {} back to {}".format(self.server_message, address))

# Update VM Load
self.my_mutex.acquire()
if str(vmLoad) in SERVER_PLOT_DATA["vmLoads"]:
        logging.info("Updated VM: {} Load: {}".format(vm, vmLoad))
        self.vms[vm]["vm"] = vm
        self.vms[vm]["load"] = vmLoad
        self.vms[vm]["prob"] = client_message["prob"]
else:
        logging.error("VM Load not in SERVER_PLOT_DATA")
self.my_mutex.release()
elif client_message["request"]["migration"]:
# MigrateVm
target = client_message["vm"]["target"]
migrationThread = threading.Thread(target=migrateVm, args=[vm, target])
migrationThread.setDaemon(True)
migrationThread.start()
migrationThread.join()
logging.debug("migrated {} to {}".format(vm, target))

# Delete VM Load
```

```
self.my_mutex.acquire()
if vm in self.vms:
        logging.info("Deleted VM: {} Load: {}".format(vm,
         ↪  self.vms[vm]["load"]))
        del self.vms[vm]
self.my_mutex.release()
else:
logging.error("Wrong VM Request Message")
else:
logging.error("VM Domain-Name did not deduced correctly")
except:
logging.error("Exception in HandleClient thread")

''' Main
'''
if __name__ == "__main__":
# Clear Log Content
with open('/var/tmp/server.log','w'): pass

# Get calibration temperature
initialTemp = calibrate.Calibrate(600).getCalibrationTemp()
maxTemp = 250 # for 10 VMs @100% load
logging.info("Initial Average Host Temperature = {} and maxTemp
 ↪  = {}".format(initialTemp, maxTemp))

# Start Server
Server(initialTemp, maxTemp).run()
```

B.3 workload.py

```
#!/usr/local/bin/python
# -*- coding: utf-8 -*-

from collections import deque
from threading import Timer
from random import shuffle
import subprocess

class PeriodicTask():
        def __init__(self, interval, callback, daemon=False,
         ↪  **kwargs):
                self.interval = interval
                self.callback = callback
                self.daemon = daemon
                self.kwargs = kwargs

                # Initialize random load from list
```

```python
                loadList = [25, 50, 75, 100]
                shuffle(loadList)
                self.load = deque(loadList)

        def run(self):
                load = self.load.pop()
                self.load.appendleft(load)
                self.callback(load, self.interval, **self.kwargs)
                t = Timer(self.interval, self.run)
                t.daemon = self.daemon
                t.start()

def job(load, timeout):
        cmd = "stress-ng --cpu 1 --cpu-method matrixprod --cpu-load %s --timeout %s &" % (load, timeout)
        output = subprocess.check_call(cmd, shell=True)
        return output

'''
        Main
'''
if __name__ == "__main__":
        task = PeriodicTask(interval=1800, callback=job)
        task.run()
```

B.4 utils.py

```python
# -*- coding: utf-8 -*-

import subprocess

SERVERS = {
        "trident1.vlab.cs.hioa.no" : "128.39.120.89",
        "trident2.vlab.cs.hioa.no" : "128.39.120.90",
        "trident3.vlab.cs.hioa.no" : "128.39.120.91"
}

SERVER_MESSAGE = {
        "maxTemp"  : 0.0,
        "hostTemp" : 0.0
}

CLIENT_MESSAGE = {
        "request" : {
                "login" : False,
                "temperature" : False,
            "migration" : False
        },
        "vm" : {
            "mac" : "",
            "target" : "",
            "load" : 0
        },
        "prob" : {
            "trident1.vlab.cs.hioa.no" : 0.0,
            "trident2.vlab.cs.hioa.no" : 0.0,
            "trident3.vlab.cs.hioa.no" : 0.0
        }
}

def getHostName():
    hostName = ""
    for key, value in SERVERS.items():
        cmd = "sudo traceroute -n %s | tail -n+2 | awk '{ print $2 }' | wc -l" % (value)
        try:
            if(int(subprocess.check_output(cmd, shell=True).decode('UTF-8').rstrip("\n")) == 1):
                hostName = key
                break
        except subprocess.CalledProcessError as e:
            print("ERROR: : {reason}".format(reason=e))
    return hostName

def getHostIp(hostName):
    hostIp = ""
    if hostName in SERVERS:
        hostIp = SERVERS[hostName]
    return hostIp

def getVmMac():
    try:
        mac = subprocess.check_output("sudo ifconfig | grep 'HWaddr' | awk '{print $NF}'", shell=True).decode('UTF-8').rstrip("\n")
        return mac.lower()
    except subprocess.CalledProcessError as e:
        print("ERROR: : {reason}".format(reason=e))
    return ""

def getLoad():
    try:
```

```
                load = subprocess.check_output("sudo ps | grep stress-ng | head -1 | awk '{print $NF}'", shell=True).decode('UTF-8').rstrip("\n")
                return int(load)
        except subprocess.CalledProcessError as e:
                print("ERROR: : {reason}".format(reason=e))
        return 0
```

Appendix C

Experimental Results

C.1 Choose Coolest Server Algorithm: Uniform Workload with 12 VMs

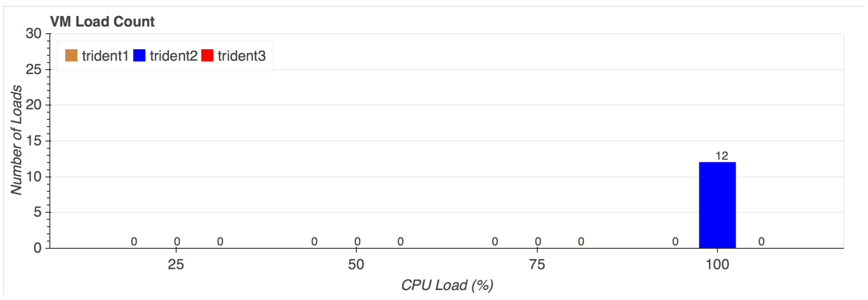

Figure C.1: Variable workload distribution after thermal balancing

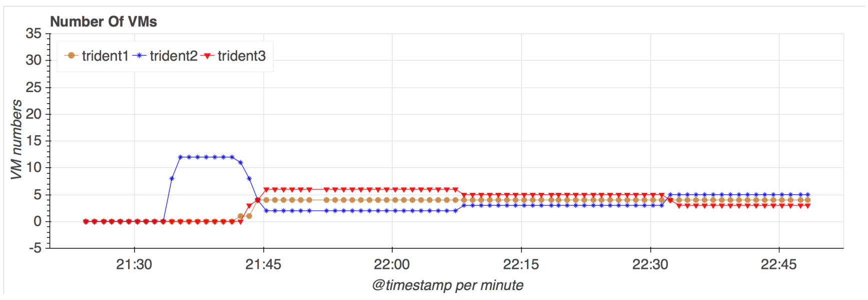

Figure C.2: Number of running VMs with uniform workload

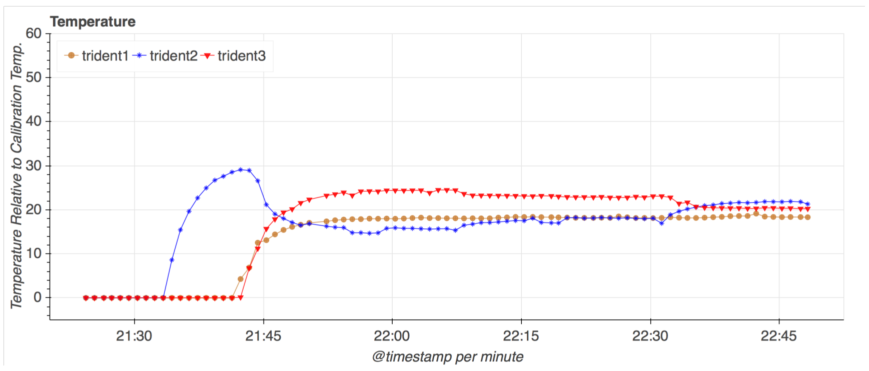

Figure C.3: Temperature readings with uniform workload

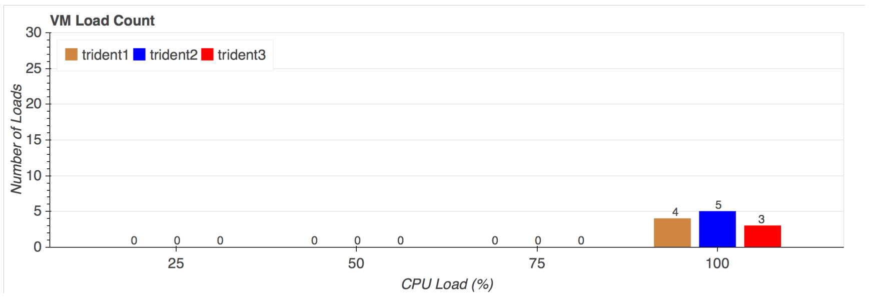

Figure C.4: Uniform workload distribution after thermal balancing

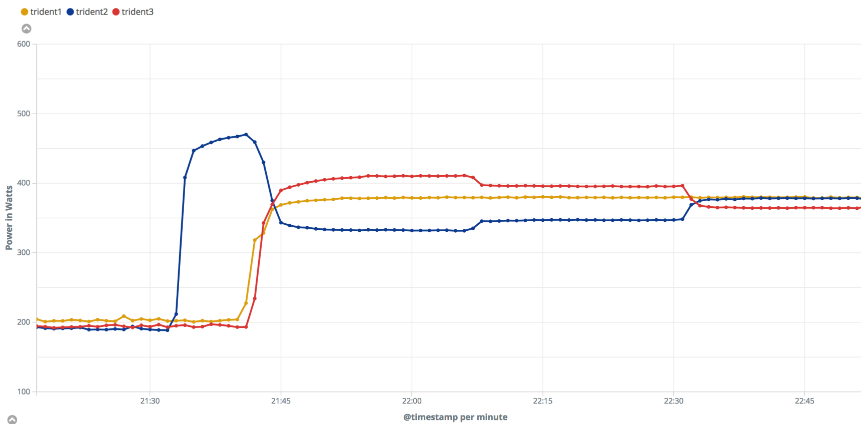

Figure C.5: Power consumption reading with uniform workload

C.2 Choose Coolest Server Algorithm: Variable Workload with 12 VMs

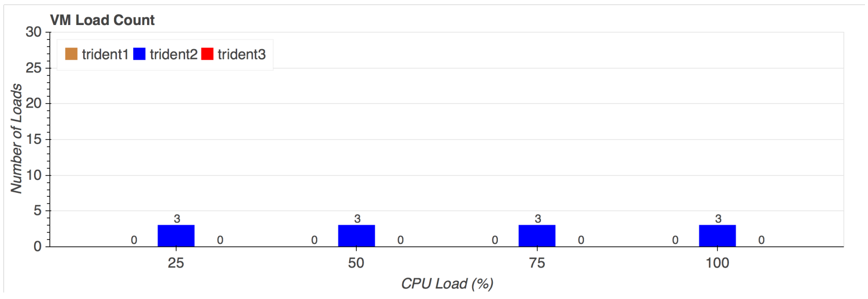

Figure C.6: Initial Variable workload distribution

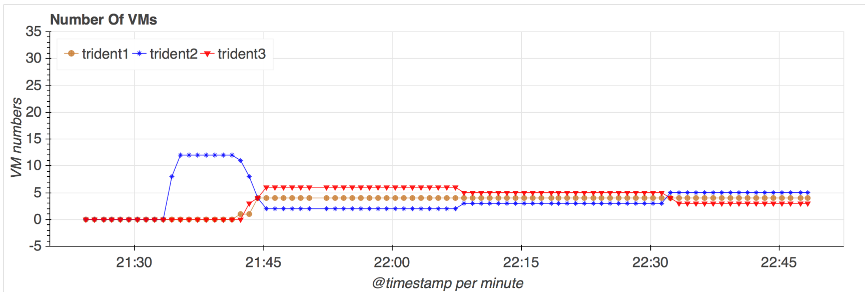

Figure C.7: Number of running VMs with variable workload

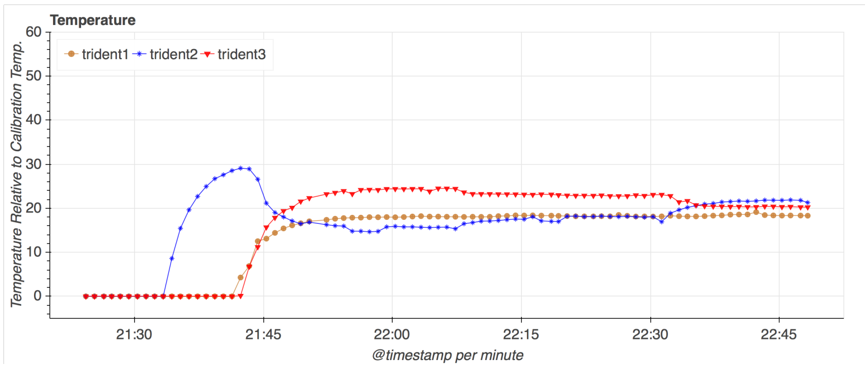

Figure C.8: Temperature readings with variable workload

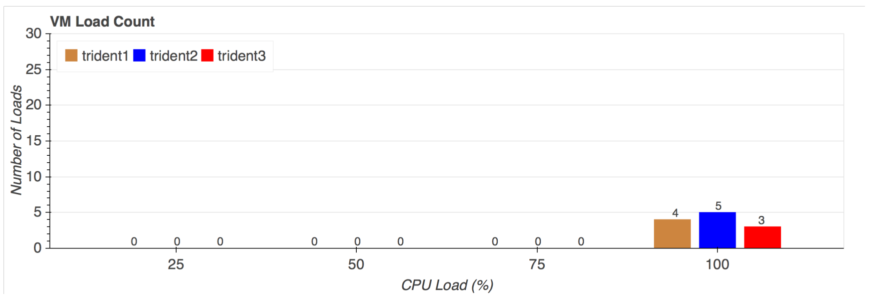

Figure C.9: Variable workload distribution after thermal balancing

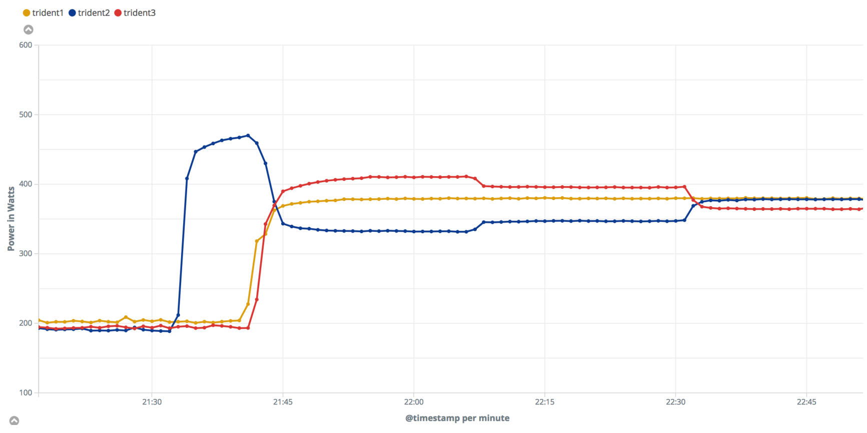

Figure C.10: Power consumption reading with variable workload