A CLASS-BASED VIRTUAL MACHINE CONSOLIDATION FOR IMPROVED QUALITY OF SERVICE AND ENERGY CONSERVATION IN CLOUD COMPUTING

BY

AJAYI, OLASUPO OPEYEMI

NOVEMBER, 2017

A CLASS-BASED VIRTUAL MACHINE CONSOLIDATION FOR IMPROVED QUALITY OF SERVICE AND ENERGY CONSERVATION IN CLOUD COMPUTING

By

AJAYI, Olasupo Opeyemi

030805018

B.Sc. Computer Science (2008), University of Lagos.

M.Sc. Computer Science (2012), University of Lagos.

A thesis submitted to the School of Postgraduate Studies, University of Lagos, Akoka, Lagos, Nigeria, in partial fulfilment of the requirement for the award of the degree of Doctor of Philosophy (Ph.D.) in Computer Science.

NOVEMBER, 2017

CERTIFICATION

School of Postgraduate Studies, University of Lagos.

This is to certify that the thesis entitled:

A CLASS-BASED VIRTUAL MACHINE CONSOLIDATION FOR IMPROVED QUALITY OF SERVICE AND ENERGY CONSERVATION IN CLOUD COMPUTING

Submitted to the School of Postgraduate Studies, University of Lagos, for the award of the degree of Doctor of Philosophy (Ph.D.) in Computer Science is a record of original research carried out by:

AJAYI, Olasupo Opeyemi

In the Department of Computer Sciences

Author's Name	Signature	Date
First Supervisor's Name	Signature	Date
Second Supervisor's Name	Signature	Date
External Examiner's Name	Signature	Date
First Internal Examiner's Name	Signature	Date
Second Internal Examiner's Name	Signature	Date

DEDICATION

To my mother, I know you always wished you could, now you can be proud that your son did. This is for you momi.

To my father who made reading compulsory.

To my grandmother (mama) who on leaving for school, quietly made me promise her that "ibi to ga ju ninu iwe kika, omo mi ba wan debe" – this is promise fulfilled ma.

To my god-mother, I miss you so much big momi.

ACKNOWLEDGEMENTS

I have come to understand that without doubt, my life and everything about it is purely at God's discretion, I am therefore, truly grateful for His grace. It has simply been the grace of God in my life that has made this and every other thing in my life possible. Baba God, it's been You all the way.

To my family, my parents, Mr. and Mrs. O. A. Ajayi, I say thank you for all your support, love, prayer and patience. It is my prayer that God would continually bless you and reward you abundantly. My siblings, Mrs. O. O. Dada, Mr. O. O. Ajayi, and Mr. A. O. Arowolo, thank you, you have been awesome.

To my supervisor, Professor C.O. Uwadia, words would be grossly insufficient to express my gratitude to you sir. You have been the perfect pseudo-father to me and have taught me so many things, within and outside academia. Despite your busy schedule, you always found time. No matter how tight your week was, you always squeezed out time to see me and review my work. Your belief in me often times amazes me and I sincerely appreciate your time, efforts, advice, encouragement, support and love. Thank you very much and God bless you sir.

To my co-supervisor and my school mother, Dr. (Mrs.) F. A. Oladeji, I say thank you ma. I always ran to you when I got stuck on any aspect of my work and you always seemed to have a solution or knew someone who did. Ma, your efforts and time are highly appreciated, may God continue to bless and strengthen you ma.

My gratitude goes to Professor T. O. Ogundipe, I still do not understand how you saw through me back then but I truly believe it was God working through you. Thank you very much sir and yes I am glad I embarked on this journey.

In a very special way, I am truly thankful to Emeritus Professor O. Abass, who took keen interest in my work and offered subtle advices which shaped this research work. I am grateful sir.

The efforts and contributions of my teachers, Professor J. O. A. Ayeni, Dr. E. P. Fasina, Dr. B. A. Sawyerr, and Dr. O. B. Okunoye are truly appreciated. You were tough and disciplined teachers but you instilled a lot in me. I say thank you sirs, your efforts were worth it.

Special thanks to Dr. A. P. Adewole, Dr. V.O. Odumuyiwa, Dr. A. O. Sennaike, Dr. (Mrs) C.O. Yinka-Banjo, Dr. N. A. Azeez and Dr. A. U. Rufai, you were always there; particular in

those low moments, your advice and words of encouragement kept me going. I am really thankful. I appreciate my friends and colleagues, Mr. S.E. Edagbami, Mrs. R. A. Ajetunmobi, Ms. R. O. Isimeto, Mrs. C.P. Ojiako, Mr. L. O. Ikuvwerha, Mrs. D.T. Afolabi, Mr. O. Olaitan, Mr. A. S. Adewunmi, Mr. E. A. Okhueleigbe, Mrs. Idowu and Ms. Nonye you made every day in academia interesting and fulfilling.

To my other colleagues, in the Department of Computer Sciences and the Faculty of Science, University of Lagos, I say thank you.

To my friends, Ms. O. I. Ogungbe, Ms. I. U. Chukwbueze, Mr. O. O. Onibile, Mr. O. O. Ajayi, and Mr. O. O. Oredugba, you encouragement were immeasurable. Thank you.

Finally, to Mr. Ayo Adefemi and Mrs. Adetutu Unuigboje, for your support and encouragement, I can only say thank you and God bless you.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Cloud computing is a model in which computer resources are provided as paid services to users. It has enjoyed wide spread acceptance in recent times due to its numerous advantages, particularly in terms of cost savings. Despite its advantages, it still has some challenges, such as Quality of Service (QoS) adherence, efficient resource utilization, and energy conservation. From the Cloud users' perspective, service provisioning and adherence to QoS are vital; while to the Cloud Service Providers (CSPs), efficient resource utilization and conservation of the energy consumed by Cloud data centres are key. Meeting these requirements collectively is a major Cloud computing challenge. Virtual Machine Consolidation (VMC) has become the *de facto* technique for addressing resource utilization and energy conservation, while efficient scheduling and migration techniques are used to address QoS. Power-Aware Best Fit Descending (PABFD) and Virtual Machine Consolidation with Multiple Usage Prediction (VMCUP-M) are examples of recent research works that used the combination of both approaches to address this challenge. However, in these works like many others, users were treated equally without regard to class of requirements. In a bid to address this shortcoming, this study proposed an approach that grouped workloads into classes, then used a class-based VMC scheme to ensure end-to-end adherence to QoS. It also improved on resource utilization and energy conservation by combining resource utilization prediction with a half-interval workload allocation scheme. The proposed approach is called Multi-Class Load Balancing (MC-BAL). MC-BAL was tested against PABFD and VMCUP-M, using three datasets and across static and dynamic PM threshold schemes. Obtained results show that MC-BAL met the set objectives by performing better than PABFD and VMCUP-M in terms of QoS adherence, by an average of 13 % and 26 % for both static and dynamic thresholds respectively. With respect to resource utilization, MC-BAL used at least 19 % less resources to accomplish the same tasks versus the other two approaches. In terms of energy conservation, MC-BAL consumed an average of 24.7 % less energy than both PABFD and VMCUP-M. MC-BAL is able to address critical requirements that are vital to Cloud stakeholders. For Cloud users, MC-BAL ensures satisfactory (quality) service delivery. Using MC-BAL, CSPs can manage their resources better, creating room to take on new customers and increase profit margins. To the society at large, MC-BAL lowers energy consumption of data centres and by extension carbon emission, thereby addressing one of the major concerns in the world today.

Keywords: Cloud Computing, Energy Conservation, Quality of Service, Resource Management, Virtual Machine Consolidation.

AUTHOR'S STATEMENT

I hereby agree to give the University of Lagos through University of Lagos Library, a non-exclusive, worldwide right to reproduce and distribute my thesis and abstract (hereinafter "the Work") in whole or in part, by any and all media of distribution, in its present form or style or in any form or style as it may be translated for the purpose of future preservation and accessibility provided that such translation does not change its content.

By the grant of non-exclusive rights to University of Lagos through the Library under this agreement, I understand that the rights of the University of Lagos are royalty free and that I am free to publish the Work in its present version or future versions elsewhere.

Warranties

I further agree as follows:

i. That I am the author of the Work and I hereby give the University of Lagos the right to make available the Work in the way described above after a three (3) year period of the award of my doctorate degree in compliance with the regulation established by the University of Lagos Senate.
ii. That the Work does not contain confidential information which should not be divulged to any third party without written consent.
iii. That I have exercised reasonable care to ensure that the Work is original and it does not to the best of my knowledge breach any Nigerian law or infringe any third party's copyright or other Intellectual Property Right.
iv. That to the extent that the Work contains material for which I do not hold copyright, I represent that I have obtained the unrestricted permission of the copyright holder to grant this license to the University of Lagos Library and that such third party material is clearly identified and acknowledged in the Work.
v. In the event of a subsequent dispute over the copyrights to material contained in the Work, I agree to indemnify and hold harmless the University of Lagos and all its officers, employees and agents for any uses of the material authorized by this agreement.
vi. That the University of Lagos has no obligation whatsoever to take legal action on my behalf as the Depositor, in the event of breach of Intellectual Property Rights, or any other right, in the material deposited.

Author's Name	Signature / Date	Email
Supervisor's Name	Signature / Date	Email
Supervisor's Name	Signature / Date	Email

CHAPTER ONE

INTRODUCTION

1.1 Background of the Study

Over the years, computers have evolved through computer generations from large mainframe systems that occupied entire rooms to small sized super-fast devices that are inter-networked and use multiple processors for simultaneous processing of programs. Today, computing is being offered and used as a commercial service, one in which users need not own physical infrastructure or software applications, but simply log into remote devices to store or use their data on a pay-as-you-use basis (Furht and Escalante, 2010). This is what is referred to as Cloud computing (CC).

There are numerous definitions of CC, but the widely accepted definition is that of the National Institute of Science and Technology (NIST), defined by Mell and Grance (2011) as a model that enables convenient and ubiquitous access to a shared pool of configurable computing resources that can be rapidly setup or torn down on user demand with little or no service provider intervention. It is a model that offers computing to users as a paid service. Resources in CC could be at the hardware level: Infrastructure-As-A-Service (IAAS), at the software level: Software-As-A-Service (SAAS) or at a developer level: Platform-As-A-Service (PAAS) and deployed either as a private, public, community or hybrid model.

As with most things in life, there is no perfect system and CC is one of such. CC is plagued with numerous challenges some of which include adherence to Quality of Service (QoS), resource management, energy conservation, security, data ownership, service availability and excess carbon emission. QoS adherence, efficient resource utilization and energy conservation are key to this study as they directly affect both the users and Cloud Service Providers (CSPs) alike. A lot of research work has been carried out in a bid to ensure an efficient way of managing these challenges while delivering services to users. The works of Fang *et al.* (2010), Lu *et al.* (2011), Beloglazov and Buyya (2012), Xu *et al.* (2014), Hieu *et al.* (2015), Mosa and Paton (2016), Bermejo *et al.* (2016) and Hieu *et al.* (2017) are notable examples which focus on the balancing Quality of Service (QoS), efficient resource utilization with energy conservation.

A number of researchers have proposed various scheduling algorithms to allocate user workloads onto Cloud resources in such a way that pre-signed Service Level Agreements (SLAs) are not violated. Though effective, scheduling algorithms focus on admission control and seek to reduce the delay experienced by users while waiting for a resource to become available to service their requests.

Load balancing through the use of Virtual Machine Consolidation (VMC) is an approach that has been utilized to address the issue of resource utilization. It is a mechanism that attempts to distribute user workloads (on virtual machines) evenly across all the Cloud resources in a bid to avoid a situation where some resources are heavily overloaded while others are idle. It is a scheme that seeks to improve the overall performance and resource utilization of the system by consolidating workloads onto the fewest possible number of Physical Machines (PMs).

Different schemes have also been proposed by researchers to address energy efficiency, with the two notable ones being Dynamic Voltage-Frequency Scaling (DVFS) (Holzle and Barroso, 2007) and PM sleep-states (Meisner *et al.*, 2009). DVFS manages energy by throttling the voltage consumption in proportion to Central Processing Unit (CPU) frequency. PM sleep-state on the other hand actively monitors CPU status of PMs and switches underutilized or idle PMs to sleep-mode. Most recent works employ the PM sleep-states approach considering it more effective than DVFS, as DVFS only focuses on CPU, ignoring other components such as memory, storage and peripheral devices present in the PM.

A mix of effective admission control, load balancing (VMC) and/or energy conservation schemes has been the approach taken by most researchers when proffering solutions to the aforementioned challenge. A survey of some of these hybrid approaches has been done by Ajayi *et al.* (2015) and Ullrich *et al.* (2016). Two works of particular interest to this study are Power Aware Best Fit Descending (PABFD) (Beloglazov and Buyya, 2012) and Virtual Machine Consolidation with Multiple Usage Prediction (VMCUP-M) (Hieu *et al.*, 2017). These works proposed models that addressed the three challenges of QoS adherence, resource utilization and energy conservation in CC. The latter (VMCUP-M) improved on the former (PABFD) by introducing resource utilization prediction. This addressed the shortcomings of the former by reducing the number of Virtual Machine (VM) migrations and resulted in better resource utilization and QoS adherence. This study improved on both of these works, by proposing and implementing an alternative QoS, resource and energy-aware scheme for resource management in CC.

1.2 Statement of the Problem

Cloud Compuing has emerged as the next Information Technology (IT) revolution. It represents a paradigm shift from conventional personal computer (desktop or laptop) to remote computing, whereby data are stored in locations separate from their users. This enables Cloud users have access to their data from any device and from any location in the world as long as they can access the Internet. Apart from ubiquitous access to data, the burden of purchasing expensive IT infrastructure, maintaining high availability, flexibility and security of data are taken off the customer and borne by the Cloud provider. The advantages of the Cloud, have led to many individuals and companies the world over adapting it at a phenomenal rate. However, the same cannot be said about Africa and her developing countries. Developing countries are faced with an ever increasing energy challenges; thus situating state-of-the-art Cloud data centres in these countries is almost an inconceivable thought, as there would hardly be power to keep them running uninterrupted (Ani *et al.*, 2012).

On the other hand, irrespective of the location or challenges faced by the Cloud Service Providers (CSPs), the expectations of their customers remain the same - customers would continue to demand for service, and submit workloads with diverse requirements which the CSPs are expected to meet. In practice, Cloud resources do not grow in equal proportion to Cloud users; hence CSPs are faced with a challenge of how to efficiently utilize their limited resources in such a way as to continuously provide services to users in a profitable manner without violating SLAs.

Balancing these challenges has therefore led to the problem of providing services to numerous Cloud users using limited Cloud resources without violating QoS requirements pre-agreed with the users; yet utilizing these limited CSPs' resources efficiently and in a manner that consumes the least amount of energy.

1.3 Aim and Objectives

The aim of this study is to develop a resource management model that simultaneously adheres to pre-set QoS requirements, efficiently utilizes resources and conserves the total energy consumed in Cloud data centres.

The specific objectives of the study are to:

1. develop a load balancing scheme that ensures adherence to end-to-end, pre-set QoS while providing services to Cloud users.
2. develop a scheme that efficiently utilizes Cloud resources while providing services to Cloud users.
3. improve the overall energy consumption of Cloud data centres.

1.4 Scope and Delimitation of Study

Classification of user workloads into classes is solely based on workload burst and response times. In this study, response times have been taken to be "Best Effort", "Some Delay Permitted" or "No Delay Permitted" which correspond to the three classes of user workloads, viz. Bronze, Silver and Gold respectively. No further analysis was done in classifying user workloads into groups.

VM reuse is not considered as this study assumes that a unique VM is created for each user's workload; once completed, the associated VM is destroyed.

The actual content or type of a user's workload is not considered. This study simply focuses on the user specified requirements and concentrates on providing resources that can cater for these requirements.

Though MC-BAL can use multiple resources, only CPU utilization is used in determining the status of a PM. This is because the CPU alone has been shown to account for up to 70 % of the overall power consumed by a computer system (Pennsylvania, 2013).

1.5 Significance of the Study

Three major Cloud computing challenges have been identified viz. QoS adherence, efficient resource utilization and energy conservation. To Cloud users, adherence to QoS is of upmost importance, while to the CSPs, efficient resource utilization and energy conservation are paramount. The significance of this study therefore was to develop a resource management scheme that ensured that requirements which are vital to both the Cloud users (QoS) and the

CSPs (efficient resource utilization and energy conservation) were met. In effect, the study developed a scheme that benefits both the users of the Cloud and the CSPs at the same time.

1.6 Definition of Terms

Adherence: Compliance with pre-set terms and/or conditions.

CloudSim: A toolkit for simulating Cloud infrastructure and services.

Consolidation: Process of combining or packing multiple entities into a single whole.

Conservation: Preservation and efficient management of resource(s).

Data centre: A building where a large number of inter-connected computer servers are operated.

Data set: A log of users' workloads submitted to a Cloud data centre.

Delay: Time spent waiting (on queue) for service.

Efficient: Performing or functioning in a manner that minimizes wastage or effort.

End-to-End: Comprehensive and all encompassing.

Energy: Electrical power consumed per unit time in a data centre.

Load Balancing: Re-distribution of workloads among PMs to prevent a situation whereby certain PMs are over-worked while others are idle.

Multi-tenancy: A CC model that allows numerous users share resource(s).

Physical Machine: A high-end computer system, usually a server.

Quality of Service: A users' perception of the level of service being received from a service provider.

Queue: A line of workloads awaiting PM allocation.

Service Level Agreement:	A contract between a user and service provider, which documents the services and standards expected by the user from the provider alongside the expectations from the user.
Service Provider:	An individual or organization that provides certain services to users, usually at a fee.
Sleep-State:	Power-saving or low activity mode.
Utilization:	The act of making use of something.
Users:	Cloud customers or subscribers.
Violation:	Breaking or dishonouring an agreement.
Virtualization:	Process of creating a software-based replica of a physical device.
Virtual Machine:	An emulated computer system. Multiple virtual machines run on a PM.
Workload:	A user's service or job, which is submitted to the service provider for execution.
Workload Class:	Category or group of workloads with similar service requirements.
Workload-Migration:	Moving workloads between PMs in a data centre.

1.7 List of Abbreviations and Acronyms

2DHIS:	Double-Depth Half Interval Search
CC:	Cloud Computing
CPU:	Central Processing Unit
CSP:	Cloud Service Provider
DC:	Data Centre

DVFS:	Dynamic Voltage-Frequency Scaling
FCFS:	First Come First Serve
GCD:	Google Cluster Dataset
GTC:	Google Test Cluster
IAAS:	Infrastructure-As-A-Service
IT:	Information Technology
IQR:	Inter-Quartile Range
LR:	Linear Regression
MAD:	Mean Absolute Deviation
MC-BAL:	Multi-Class Load Balancing
PAAS:	Platform-As-A-Service
PABFD:	Power-Aware Best Fit Descending
PM:	Physical Machine
PSC:	Power State Change
QoS:	Quality of Service
RBT:	Red-Black Tree
SAAS:	Software-As-A-Service
SLA:	Service Level Agreement
THQ:	Static Threshold
VM:	Virtual Machine
VMC:	Virtual Machine Consolidation
VMCUP:	Virtual Machine Consolidation with Usage Prediction
VMCUP-M	Virtual Machine Consolidation with Multiple Usage Prediction
VMM:	Virtual Machine Migration

CHAPTER TWO
LITERATURE REVIEW

2.1 Introduction

There are numerous CC challenges, which include but not limited to data management, energy conservation, high availability, interoperability and standards, resource management, QoS, security, privacy and legal issues. Of particular interest to this study however are QoS, resource utilization and energy conservation as they relate to CC. There is therefore the need to elaborate on each of these concepts upon which this study is based. This chapter thus focuses on these and is organized as follows: an introduction to Cloud computing, its service delivery models and architecture is done in section 2.2. Quality of Service in CC is discussed in section 2.3, while a brief justification for the need of resource management in Cloud computing is done in section 2.4. Resource utilization and the various ways of achieving efficient utilization are discussed in section 2.5. Energy conservation as it relates to CC are discussed in section 2.6, while related works wherein these three challenges are considered are discussed in section 2.7. In section 2.8, a detailed analysis of works against which this study is benchmarked is done. The chapter is then concluded in section 2.9.

2.2 Cloud Computing

Computing and data processing have in recent times moved from personal or super computers located within homes or offices to large data centres geographically dispersed around the world (Furht and Escalante, 2010). There has been a paradigm shift in the way computers and computing resources are used. Today, computing is now being offered and used as a commercial resource whereby users pay the provider(s) on a pay-as-you-use model similar to other utilities such as electricity, water, gas etc. (Voorsluys *et al.*, 2011).

2.2.1 Definition of Cloud Computing

A Cloud according to Buyya *et al.* (2009) is defined as a parallel and distributed computing system consisting of a pool of inter-connected and virtualized computers that are dynamically provisioned and presented as a single computing resource to the users based on pre-agreed SLAs. Sidhu and Kinger (2013), defines it as a framework for enabling a suitable on-demand network access to a shared pool of computing resources (such as networks, servers, storage, applications, services etc.) that can be provisioned and de-provisioned quickly with minimal

management effort or service provider interaction. It is defined in Shahzad (2014) as a disruptive technology that has the potential to enhance collaboration, agility, scaling, and availability, and provides the opportunities for cost reduction through optimized and efficient computing. CSA (2011) defines Cloud computing as a technology that has the potential to transform the world into one in which components can be rapidly provisioned, decommissioned, scaled up or down in a bid to provide an on-demand utility-like model of allocation and consumption of Computing resources. From a technological perspective Cloud Balancer (2015) classified CC as a technological evolution in computing. These evolutionary phases are depicted in Figure 1. Finally, Mell and Grance (2011) defines it as a model that enables convenient and ubiquitous access to a shared pool of configurable computing resources. Cloud computing can thus be defined as a transition from computers as a product to computing as service rendered to paying users.

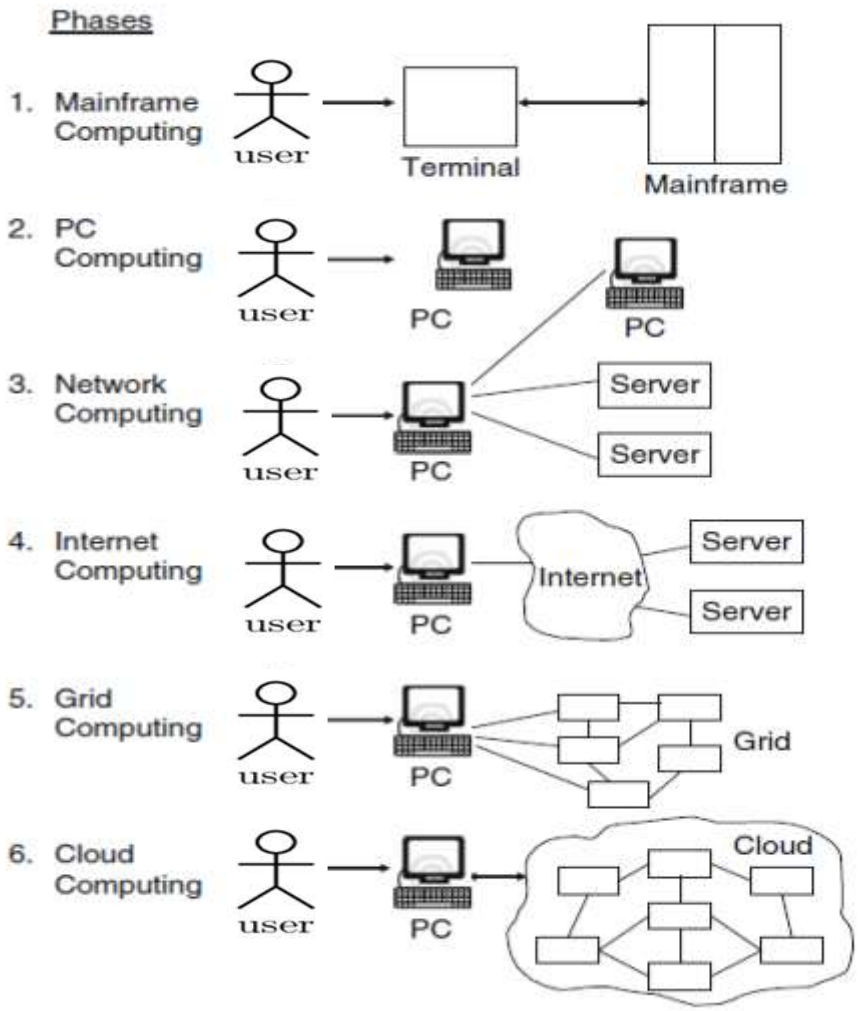


Figure 1: Paradigm shift in Computing (Furht and Escalante, 2010)

These pools of computing resources are offered to customers as a paid service. A generalized model for the Cloud as proposed by the NIST is depicted in Figure 2.

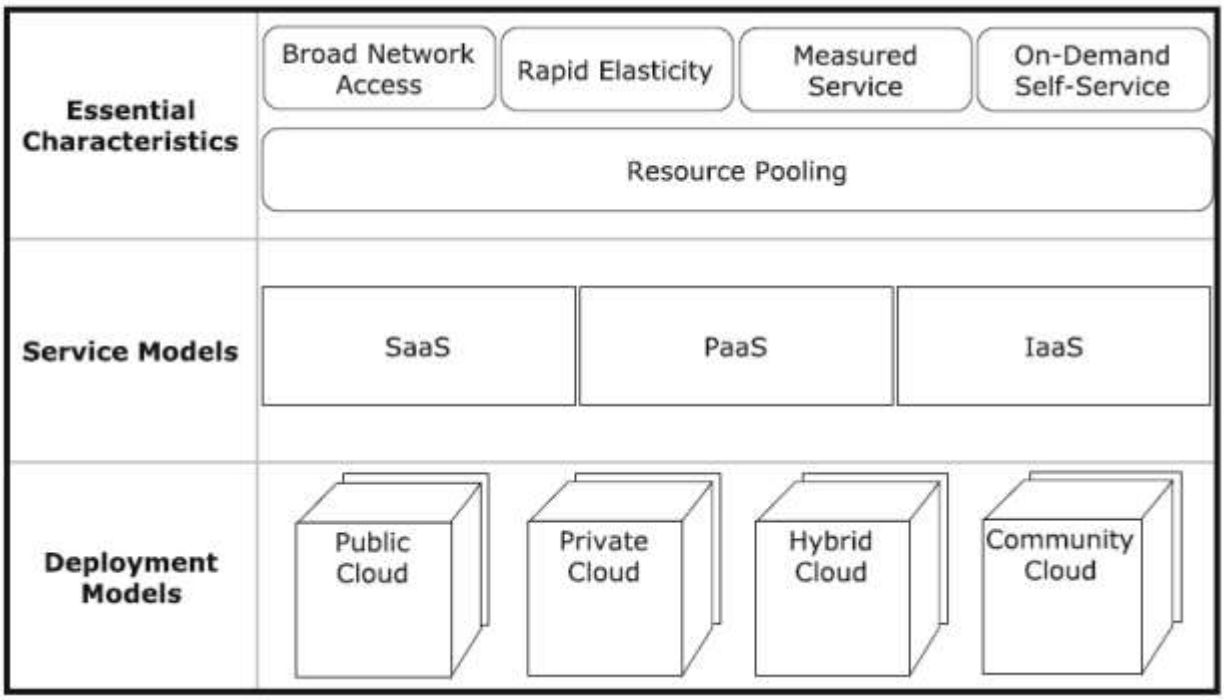


Figure 2: A visual model of Cloud Computing (Mell and Grance, 2011)

2.2.2 The Cloud Computing Architecture

In Calheiros *et al.* (2011), CC is depicted as a layered structure as shown in Figure 3. The lowest (System) layer consists of a pool of physical resources, which are made available to users via virtualization. This is the layer offered to customers in the IAAS Cloud model. Above this layer is the Core Middleware layer and is concerned with the provision of services to users. It is split into two parts, a lower layer which interfaces with the System layer, providing services such as VM deployment and management; and an upper layer which interfaces with the User-Level Middleware and providing services such as QoS and SLA maintenance, billing and usage monitoring. This Core Middleware layer is offered to users as a PAAS. Potential customers here are application developers requiring a plethora of tools and interfaces for application development. The User-Level Middleware sits above the Core Middleware Layer and is responsible for providing complete software solutions to customers. This is offered as the SAAS service model to Cloud users.

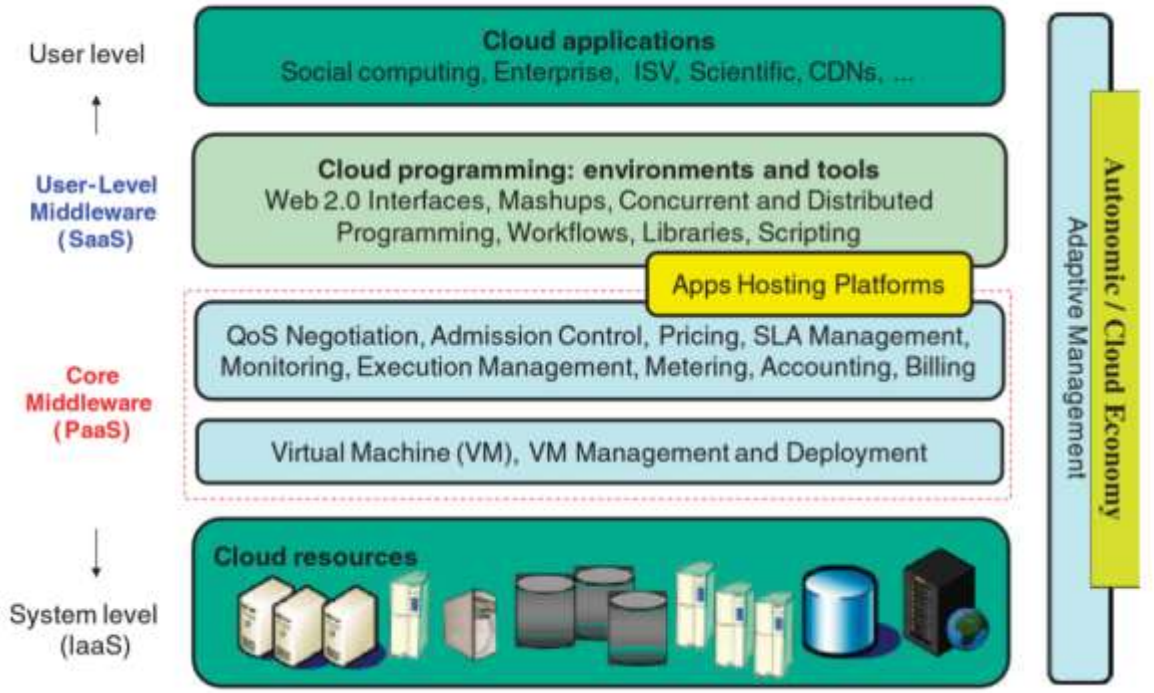


Figure 3: Cloud Computing Architecture (Calheiros *et al*., 2011)

2.3 Quality of Service (QoS)

Cloud computing offers users cheap and pay-as-you-use computing resources (Patel *et al.*, 2009). This has resulted in a tremendous user adoption rate with recent reports showing that there are over 140 million Cloud users (Cisco, 2016) with a growth rate of about 17.2 % per annum (Gartner, 2016). With more users adopting the Cloud, it becomes paramount for CSPs to be able to satisfactorily meet these users' workload requirements. The term QoS describes the particular requirement of a user's workload from a CSP during the time the required service is being provided to the user. Service Level Agreements (SLAs) are used to spell out QoS parameters required by users and expected from CSPs (Buyya *et al.*, 2011). These parameters enable CSPs measure their delivery on expectations as perceived by their users and vice versa.

Han *et al.* (2013) proposed an approach to workload scheduling based on QoS. The work introduced a hybrid of Suffrage and Min-Min allocation algorithms and split workloads into high and low groups based on their QoS requirements. Workloads with high QoS requirements were scheduled using the Suffrage algorithm while those with low QoS requirements were scheduled using Min-Min algorithm. The proposed approach ensured optimal QoS satisfaction with respect to the class of requirements. Simulation results showed that the proposed approach out performed each of the constituent algorithms in terms of Makespan, however, a comparison

of the Makespan of the high QoS workloads versus those of low QoS requirements was not done, hence the fairness of the approach cannot be ascertained.

Banerjee and Adhikari (2013) proposed an approach to task scheduling in Cloud computing that improves the overall QoS. Using Best Fit Descending, VMs were placed in descending order of their capacities. Workloads were continually allocated to a given VM so long as the VM's Remaining Load Capacity (RLC) was greater than the Maximum Load Capacity of other VMs in the system. Simulation were carried out using CloudSim and the proposed approach was compared with Round-Robin and Conductance algorithm (Chatterjee *et al.*, 2014). Obtained results showed that the proposed approach reduced the overall Makespan and completion time of workloads. The work focused mainly on VMs, ignoring external factors especially those relating to the PM upon which the VMs ran. The work also relied heavily on VM re-use, and as at the time of writing the efficacy of reusing a VM versus spawning new ones is still an open research question.

Patel *et al.* (2009) presented a mechanism for measuring the SLA between Cloud users and their CSPs. Observing the varied nature of QoS expectations, the authors drew an analogy between Cloud computing and Service Oriented Architecture used in web technologies and proposed a Web Service Level Agreement (WSLA) to manage these expectations. The proposed WSLA framework helped formalized SLA enforcement. The WSLA framework was made up of the following: management services, which measured the CSPs performances; condition evaluation services, which evaluated the CSP's performances against a set of SLA metrics and the implementation services, which took action such as imposing penalties in cases of SLA violations. WSLA however becomes difficult to implement when there are multiple parties involved in the Cloud service provisioning. Also the lack of a unified set of standards and metrics makes it difficult to actually assess CSPs' performances.

A formal model for SLA negotiation was proposed by Baig *et al.* (2017). The authors presented an agent-based multiple round SLA negotiation model. Like the work of (Patel *et al.*, 2009) this model was also based on Web Service Agreement but targeted at SLA management for Cloud services provided by multiple parties. The model incorporated multiple runs, multiple providers and multiple negotiation round. At each negotiation round a happiness factor was computed. The happiness function was used to represent the level of compromises each party (user agent and CSPs agents) could make and was used to determine the point at which satisfactory SLA could be made. The work assumed that QoS requirements of each party could

be prioritized and that these parties were willing to make compromises on certain QoS requirements.

There is still a major lack of standards with respect to Cloud computing, but certain efforts are being put in place to achieve this. The Multi-Tier Cloud Security (MTCS) of the Singapore government is one such effort (MTCS, 2015). Though primarily aimed at Cloud Security, it is a certification that guides users in choosing CSPs based on the level of data, platform or infrastructural security they provide. It incorporates multiple guidelines which can be used to formulate SLAs between users and their CSPs. MCTS offers three certification levels and has been applied in numerous areas particularly in health care (Tao and Lee, 2017).

Using Multi-tenancy (Le-Quoc *et al.*, 2013), public Cloud providers are able to host workloads of numerous users on the same PM. Virtualization and VMs are used to segment different users' jobs on these single PMs thereby giving an illusion of dedicated PMs to users. However, in reality this segmentation is not perfect as shared resources can sometimes be fiercely contented by users' workloads requiring similar resources (Xu *et al.*, 2014). Workloads are often widely dispersed with different (sometimes conflicting) resource requirements. Therefore, the one cap fits all approach applied by numerous authors wherein assumptions that all tasks require similar resources are made, might be inappropriate. To this end, a few research work have taken an alternative approach of differentiating user workloads into groups or classes for the purpose of ensuring QoS. A few of these work are now discussed.

In Goudazi and Pedram (2011) user workloads were split into two groups – Gold and Bronze based on user required response times; while in the works of Karthick *et al.* (2014) and Rajeshram and Shabarran (2015) user tasks were grouped into three groups – Short, Medium and Long based on the user indicated burst time of each tasks. In Gouda *et al.* (2013) and Pawar and Wagh (2012) the authors used multiple user supplied criteria for classification of tasks into groups. Though these works focused on tasks pre-emption, they had to classify tasks to determine priority of pre-emption. You *et al.* (2014) proposed a resource based classification of servers using RAM, CPU and bandwidth, and allocated user tasks onto the server that offered these tasks the minimum completion time. In the works of Wu *et al.* (2011) and Macias and Guitart (2012) though multiple SLA parameters such as product type, account type, request type, and response time were considered, all tasks were eventually classified into three groups – Small, Medium and Max or Gold, Silver and Bronze respectively.

From literature it can be concluded that classification of user workloads into classes is mostly based on user specified requirements, the same approach is thus applied in this study.

2.4 Resource Management in Cloud Computing

Resource Management is a critical aspect of CC. Despite the appearance of unlimited resource perceived by Cloud users, the reality is that resources in the Cloud are not unlimited. The growth of pervasive computing, drop in prices of computing devices and a reduction in cost of Internet access has led to more users connecting to the Internet and by extension the Cloud. Managing these ever growing users with varied levels and types of demand is a major challenge in CC; as resources do not grow in equal proportion with demand. This therefore emphasizes the need for efficient resource management. In this study, the only resource considered are the Cloud servers (PMs). Managing these with respect to QoS adherence and minimizing their energy consumption are the crux of this study. A review of related literature is done in the subsequent sub-sections.

2.5 Resource Utilization

Resources in CC include but are not limited to processors, memory, storage and bandwidth (Membrey *et al.,* 2012). These resources are aggregated as a pool and made available to numerous users on a pay per use basis. CSPs are concerned with efficient utilization of these resources and consequently reduction of PM sprawl. PM sprawl is a situation in which there are multiple underutilized PMs occupying DC space and accounting for more utility bill than can be justified by their performance (Bigelow, 2012). Through the use of VMs, and multi-tenancy, CSPs are able to host multiple users' workloads on the same PM, thus increasing throughput (Xu *et al.,* 2014). Ultimately, efficient resource utilization, enables CSPs provide better services to customers while making higher profit. Discussed in the subsequent subsections are some of the underlining techniques for efficient resource utilization.

2.5.1 Virtualization

Virtualization is defined by Hashizume *et al.* (2013) as a technique used in CC which allows pools of resources to be made available to numerous users simultaneously through the use of Virtual Machines (VMs). Voorsluys *et al.* (2011) defines it as a means of sharing pools of resources among users by means of partitioning physical machine(s) into numerous independent virtual machines for each user. VMs are emulated computer systems that run

simultaneously on a single PM. These VMs are often customized with different specifications and offered to users as a paid service (Zhang *et al.*, 2014). The ability to configure customized VMs to utilize different virtual partitions of resources on a single PM is one of the greatest benefits of virtualization in terms of resource utilization. This is as illustrated in Figure 4. VMs allocated to users are independent of each other and can thus be started and stopped dynamically by users without the CSPs' intervention (Mell and Grance, 2011). This enables the Cloud meet the ever changing resource demand level (Buyya *et al.*, 2011). Eucalyptus (Nurmi *et al.*, 2009), OpenStack (OpenStack, 2010), Apache VCL (VCL, 2010), and Aneka (Buyya *et al.*, 2011) are some examples of platforms currently being used for managing the utilization of Cloud resource pool; some of which have been reviewed by Sempolinski and Thain (2010).

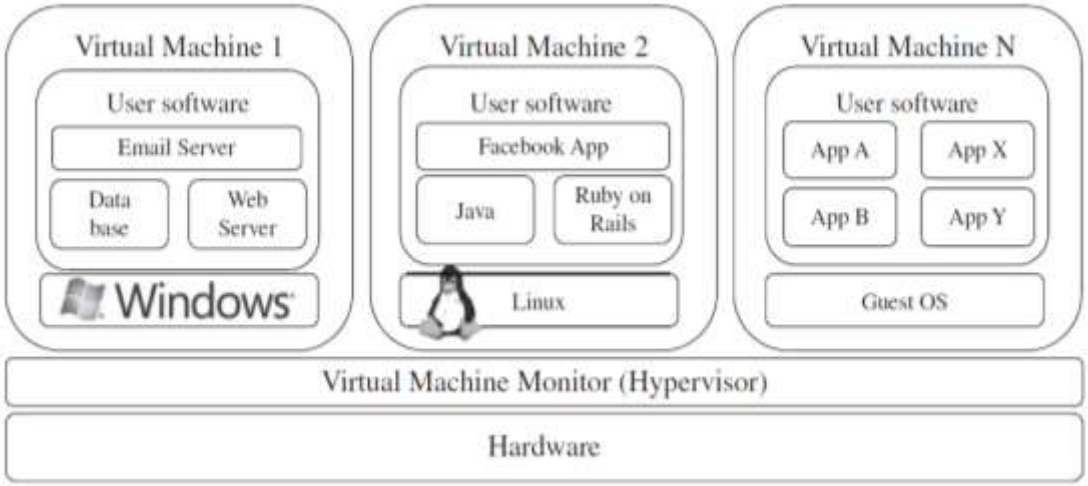


Figure 4: Multiple VMs running on a single PM (Voorsluys *et al*., 2011)

2.5.2 Virtual Machine Migration (VMM)

On one hand, it is almost impossible for users to know exactly how much resources their workloads would need during execution, while on the other hand CSPs need to be able to reallocate resources from less demanding VMs to more demanding ones. CC caters for these through a process known as dynamic scaling. Dynamic scaling is the ability to grow or shrink resource levels in response to demand. This is made possible through the use of VMM. VMM is defined as moving a VM from one PM to another for the sole purpose of meeting resource demand (Candler, 2014). In order to avoid SLA violations, these migrations have to be seamless with as little service interruption as possible (Galvin, 2009).

Akshat and Sanchita (2014) proposed a model for resource allocation and optimization using simulated annealing. The proposed model was one in which VMs were migrated based on pre-set threshold value. Power consumptions levels were used as basis for VM migration. These migrations were from PMs that consumed above and below pre-set maximum utilization and minimum utilization thresholds respectively to other PMs. In this work, the cost and effect of VM migrations were not considered; also CPU utilization was the sole criteria used for measuring PM utilization. The authors assumed a purely linear relationship between CPU utilization and power consumption.

Principled Technologies (2011) compared two virtualization platforms viz. vSphere (Setty, 2011) and Hyper-V (Microsoft, 2009), based on live VMM times and application stability after migrations. It was concluded that migration on vSphere not only took less time to complete but active VMs also suffered minimal disruptions in service while VMM was taking place.

Salfner *et al.* (2011) conducted experiments to calculate downtime and effect of live migrations on applications running on migrated virtual machines. It was concluded that the memory load and memory access pattern of the guest systems are the most important factors to be considered when performing VMM.

2.5.3 Load Balancing and Virtual Machine Consolidation

Load balancing is defined by Effatparvar and Garshasbi (2014) as the technique for spreading work between multiple computing resources for the purpose of optimizing resource utilization, improving throughput and response time.

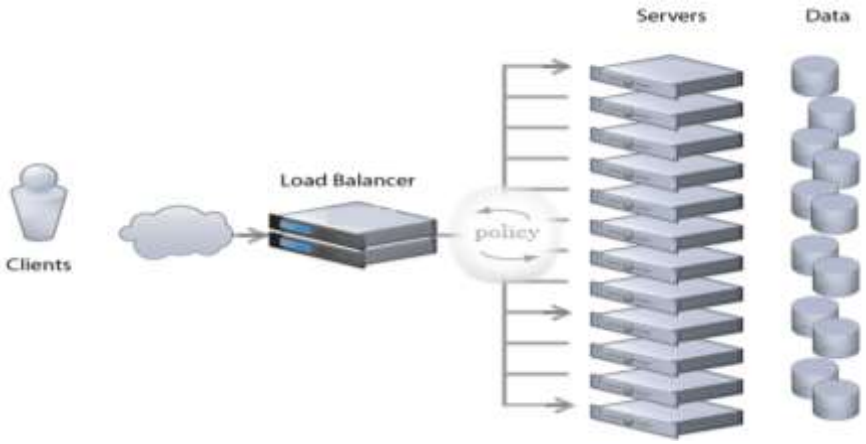


Figure 5: Load balancing of workloads among Servers (PMs)

It is a mechanism that redistributes workloads evenly across all PMs in a DC in order to avoid situations whereby some PMs are heavily loaded while others are under loaded or idle. Over working of PMs results in SLA violations (Beloglazov and Buyya, 2012); thus, load balancing is needed. Load balancing in Cloud computing is largely achieved through Virtual Machine Consolidation (VMC). VMC is the aggregation of all user workloads (VMs) on to the fewest possible number of PMs in a DC. It has become the *de facto* standard technique used by researchers seeking to improve resource utilization and energy conservation.

In the work of Fang *et al.* (2010) the authors proposed Task Scheduling Based on Load Balancing and this is as illustrated in Figure 6. User workloads were first allocated onto suitable VMs based on the user specified requirements. Subsequently, after each allocation a load balancing function (Equation 1) was computed and used to determine if the entire system was evenly balanced or not. Imbalances were resolved by migrating VMs off over-utilized PMs and consolidated on other PMs that could cater for them.

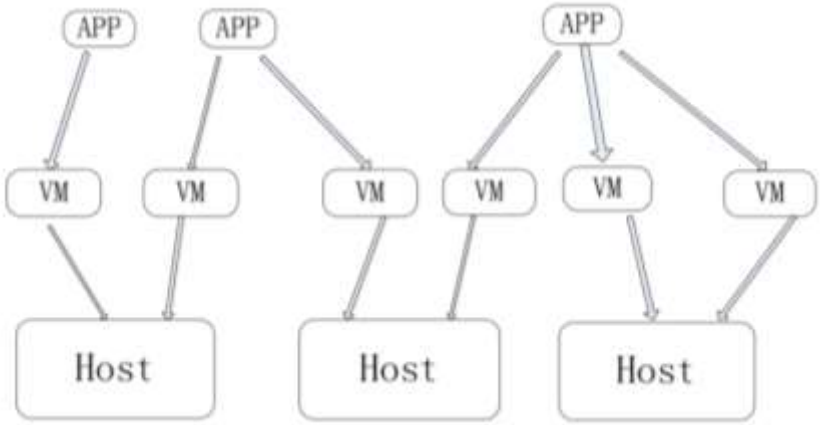


Figure 6: Task Scheduling Based on Load balancing (Fang *et al*., 2010)

$$B = \frac{\sqrt{\sum_{i=1}^{m}(H_i - avgl)^2}}{m} \quad (1)$$

$$avgl = \frac{\sum_{i=1}^{m} H_i}{m} \quad (2)$$

$$H_i = \frac{\sum_{j=1}^{n} V_j}{n} \quad (3)$$

Where V = VM load; H_i = load on a PM i; avgl = DC load, which is average load of all PMs in the DC; n = number of VMs; m = number of PMs; B = Load balancing evaluation value, smaller values of B implies better load balancing and vice versa.

Lu *et al.* (2012) proposed a novel load balancing algorithm called the Join Idle Queue (JIQ) and shown in Figure 7. The approach combined distributed dispatcher with Idle Queues (IQ). Each dispatcher had an IQ onto which idle PMs queued. On arrival of a task, the dispatchers checks if idle PM(s) are available on their IQ. If idle PMs were found, the task was dispatched to the first PM on the IQ otherwise, it was dispatched randomly to any PM. Idle PMs enqueued on dispatchers' IQ either randomly or on the shortest IQ. Though effective, the approach did not take workload imbalanced between busy PMs into consideration.

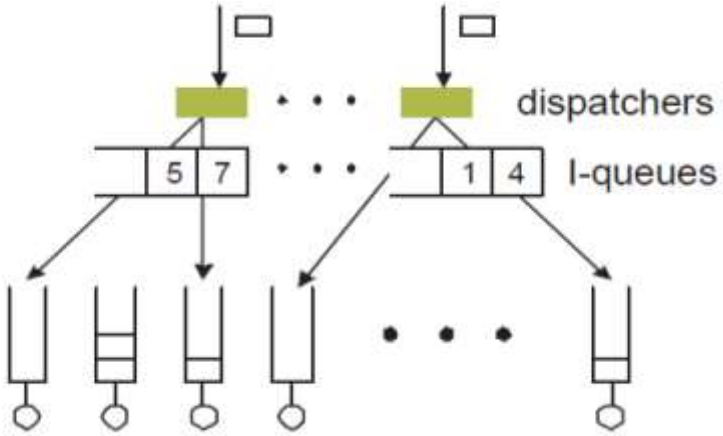


Figure 7: Join Idle Queue model proposed by (Lu *et al*., 2011)

Mahajan *et al.* (2013) discussed on a variant of round-robin called Round-Robin with Server Affinity (RRSA), which allocated workloads to PMs with a view of keeping a balance amongst the PMs. It distributed workloads using the conventional round robin algorithm but introduced a hash map and a PM state list which respectively stored information about the last allocated PM and the current state of PMs. Experimental results showed that when compared to the classic Round Robin, the response time improved as the number of data centres increased however processing time was only marginally better. There was also the high possibility of performance dips during the process of searching the hash map, prior to each allocation.

Dhinesh and Krishna (2014) proposed a nature inspired load balancing technique based on the behaviour of a colony of honeybees foraging for food. It was reported that this technique was

best suited for scenarios of heterogeneous service request types however the approach would lead to workload starvation as it is purely a priority based approach.

Choudhary (2015) improved on JIQ proposed by Lu *et al.* (2011) and presented a hybrid approach which combined SLA aware decentralized load balancing with Join-Idle-Queue (SLA+JIQ). One of the shortcomings of the original JIQ was that it ignored workloads on busy servers. This work addressed this shortcoming by monitoring PM status and redistributing workloads (VMs) based on Minimum Response Time (MRT) and minimum PM queue length. Though the work recorded improvement versus the original JIQ, numerous other works have shown that Minimum Completion Time (MCT) is a better criteria to use for VMC.

Bermejo *et al.* (2016) proposed a dual level approach to resource management. In their work, load balancing decisions were taken both within the PM and by a global controller. The PMs were autonomous and managed their local resource utilization levels independently. Each PM sent necessary load balancing information to the global controller. The global controller analysed all received status updates and made informed decisions for the global allocation or re-allocation of workloads with a view of stabilizing the entire system. A drawback of this work was that inter-nodal control messages contested with actual workload for network bandwidth. Also the need to get updates from all PMs prior to workload allocation and migration could have negative impact on QoS, especially if such updates arrived late.

Farahnakian *et al.* (2016), presented an energy efficient VMC scheme based on utilization prediction. The authors used the PABFD algorithm proposed by Beloglazov and Buyya (2012) in allocating workloads to PMs and used Linear Regression (LR) and K-Nearest Neighbour (KNN) usage prediction models for VMC. Obtained results that the KNN approach performed better than the LR based approach. Compared to other heuristic approaches, the work reported improved energy conservation and resource utilization.

Numerous other authors have proposed various load balancing approaches some of which have been surveyed in Ajayi and Oladeji (2015).

2.6 Energy Conservation

Energy saving is a major concern in Cloud computing. It affects the CSPs, the Cloud Users and the society at large. Energy conservation with respect to CC, is a direct concern of the IAAS service provider. Recent studies have estimated that though there are about fourteen (14)

million active Cloud servers in the world (SPEC, 2016). These servers are responsible for about seventy billion kilo-watt-hour (70 Billion KWh) of energy consumption annually and this figure is projected to keep growing at a rate of four (4 %) per annum. (Shehabi *et al.*, 2016).

Numerous approaches have been proposed to manage the energy consumption of DCs, with each focusing on various levels. A review of these techniques is done in the following subsection.

2.6.1 Dynamic Voltage-Frequency Scaling (DVFS)

Dynamic Voltage-Frequency Scaling is a form of dynamic performance scaling PM power management technique. It dynamically adjusts the power consumption in proportion to the frequency of the CPU. A theoretical analysis of DVFS was done in (Beloglazov, 2013).

At the operating system level, Rajkumar *et al.* (2001) proposed four DVFS algorithms as options to managing energy consumption within PMs. The algorithms were SystemClock Frequency Assignment (Sys-Clock), Priority-Monotonic Clock Frequency Assignment (PM-Clock), Optimal Clock Frequency Assignment (Opt-Clock) and Dynamic PMClock (DPM-Clock). The operating system selected which to use based on application and overall system conditions. Simulation result showed a 50 % improvement in energy conservation when these algorithms are used within a DC.

Pallipadi and Starikovskiy (2006) developed an in-Kernel real-time power manager called the on-demand governor for Linux systems. The model actively monitored CPU utilization and throttles the CPU frequency and voltage with respect to overall system performance. Though efficient for single core CPUs, performance bottlenecks began to appear when used in systems with multiple CPU cores. Due to this heavy demand, one of the CPU cores had to be used to managing the CPU manager itself.

Vardhan *et al.* (2005), proposed a scheme that managed power consumption of a PM both at the CPU and network level. The scheme called the GRACE-2 project is an improvement on an earlier generation which focused only on CPU. GRACE-2 applied DVFS with respect to the applications running on the PM and managed energy at three levels - global, per-application basis and internal level. Experimental results showed that the per-application level management resulted in an average of 22 % improvement in energy versus the two other levels.

GRACE-2 however was only supported by few applications that had been built from ground-up with energy management in mind.

Though these DVFS schemes are effective in managing energy consumption, they primarily focus on scaling the power with respect to CPU alone. They ignored other components in a PM, such as memory, fan, disks etc. Of all the components within a PM, only the CPU is capable of multiple low power modes. This implies that when a PM is completely idle, and its CPU energy consumption throttled to almost zero using DVFS, other components within the PM are unaffected and would continue to run at full capacity. Using DVFS to manage energy would result in a completely idle PM still consumes about 70 % of its maximum power (Lefurgy *et al.*, 2007, Rice *et al.*, 2015). The use of the DVFS scheme was discouraged in the work of Fan *et al.* (2007), wherein the authors concluded that having idle or underworked PMs in a DC especially when DVFS schemes were used was highly undesirable. The authors then established a linear relationship between CPU utilization and power consumption. This relationship has since formed the basis upon which PM state switching schemes were built. A survey of approaches based on PM state switching are done in the next subsection.

2.6.2 PM State Switching

An approach to energy conservation in PM called Barely Alive was proposed by Anagnostopoulou *et al.* (2012). The authors in this work, focused on improving the time it takes to switch a PM from sleep-state to active state. They proposed five PM states Barely Alive 1 to 5. At each state various components of the PM were kept active, while others were turned off. The advantage of the model was that PMs were able to transition to and from states with little or no impact on energy. This is unlike other works, where transitioning between sleep-state and fully active states often times impact on energy.

Shehabi *et al.* (2016), had attributed the reduction in energy consumption of PMs in recent times to improved cooling techniques such as the use of liquid cooling (Lamke, 2016) and virtualization. It was reported that though these approaches could lead to about 20 % improvement in energy conservation they immensely complicate system administration and management.

In recent times, virtualization has been used to improve the overall energy consumption of data centres by consolidating PMs. In Beloglazov (2013), the author described a Virtual Machine Monitor (VMM) that actively monitored the performance of a PM hosting several VMs and

applied different power management techniques to adjust power with respect to performance. In the same work, a global resource manager was presented, which interacted with the local VMM on each PM, and decided when such PMs should be switched to sleep-state with respect to resource utilization. The combined use of local and global managers resulted a robust data centre power management scheme.

In merging QoS, resource utilization and energy conservation together, Beloglazov and Buyya (2012) had shown that running a PM above its upper capacity threshold can result in QoS violations while on the contrary underutilized PMs imply poor resource utilization which translated to energy wastage.

2.7 Related Works on Load Balancing, Energy Conservation and Quality of Service

Beloglazov and Buyya (2012) proposed an energy-aware approach to task allocation and load balancing in Cloud DCs, with a focus on conservation of energy, while minimizing SLA violations. The authors pioneered work in this research area and introduced numerous PM utilization detection and VM selection schemes. The scheme however required a probe of all PM state before workload allocations. It was also assumed that all users' workload requirements were the same and belong to a single class, which is not the case in practice. In works that focus on PM consolidation and multi-tenancy such as those of Le-Quoc *et al.* (2013) and Xu *et al.* (2014), virtualization and VMs were used to consolidate workloads onto PMs in a bid to reduce the total number of active PMs. Though VMC can lead to stiff competition for resources (Xu *et al.*, 2014), it is still the most used approach for efficient resource utilization and energy conservation.

Batista *et al.* (2015) conducted performance evaluation on some resource management schemes in CC and used results obtained to develop a dynamic model called Resource Management Module (ReMM). The resulting model ReMM, was able to guarantee QoS, efficient resource utilization and fair pricing. ReMM passively monitored the status of all allocated workload with respect to agreed SLA and then dynamically scaled available resource levels to meet demand. Passive monitoring however could results in significant QoS violations during and between checks. Finally, the work identified CPU as the most influential system resource in terms of QoS adherence.

The authors in (Hieu *et al.*, 2015) improved on the work of Beloglazov and Buyya, (2012) by proposing a predictive approach to resource management in CC called VMCUP. Rather than

checking for CPU utilization after allocation, as in the former, the work predicted the short-term future state of the PM and determined if such a PM would be over/under utilized. It was a preventive approach that sought to prevent over-utilization as against PABFD which was a corrective approach. The introduction of usage prediction however, slowed the allocation process and consequently impacted negatively on QoS.

Hu *et al.* (2016) proposed a Service-Oriented Resource Management (SORM) mechanism for improving the utilization of Cloud resources. The authors clustered PMs into three groups based on resources and then allocated Cloud services onto respective groups based on requests and resource consumption. SORM then increased the number of service instances in the system in response to service demands. This approach required active monitoring of workload status and also assumed that the CSPs have full purview of user workload particularly how they changed within the system; this might not be possible in practice.

Hieu *et al.* (2017), presented VMCUP-M, a multiple usage prediction approach to resource management that takes the historic CPU and memory usage levels into consideration when allocating workloads. The work was an improvement on authors' initial VMCUP but with the introduction of memory utilization. Results of simulations carried out showed that the introduction of memory utilization prediction had minimal impact in QoS and resource utilization versus the initial VMCUP.

The works reviewed up to this point, have been purely based on deterministic approaches. Researchers have also applied stochastic approaches to address the aforementioned challenges. Though this study is based on the deterministic approach, for completeness, some works wherein stochastic approaches were applied are reviewed next.

Randles *et al.* (2009) proposed a load balancing technique that is inspired by nature. It is based on the behaviour of a colony of honeybees as they forage for food. Bees perform a waggle dance to update other bees on the status and location of a food source. The authors took PMs and User workloads to be analogous to the bees and food source respectively. Each PM performed a "waggle dance" by calculating profit based on workload serviced. This was then used to determine if more workload should be sent to that particular PM or to another. The approach concentrated on efficient workload scheduling and performed well under diverse user workloads, however it did not scale well in terms of throughput as the size of the DC grew.

In the work of Ferdaus *et al.* (2014), a modified ant colony optimization meta-heuristic was used to address resource utilization and energy conservation. Pheromone levels were associated with each VM-to-PM, and represented the attraction of VMs to a particular PM. The pheromone levels were computed based on the energy consumption and utilization level of an assignment of VMs to a PM.

Dhingra and Sanchita (2014) proposed a model for resource allocation and optimization using simulated annealing. It was a two stage model, wherein VMs were only allocated to PMs that offered the least increase in power consumption. At the second stage, VMs were migrated to other host when a pre-set upper and lower threshold value was met. Power consumptions of the entire DC and PMs was used as basis for VM migration. The decision on when to perform such VM migration was governed by an annealing process, whereby temperature (DC power consumption) was gradually reduced with each iteration until the acceptable power level was reached. An acceptance probability was also calculated at each iteration and this served as a further guide to accepting each solution. The work required an optimal power level be known before hand for any set of workload being serviced - this might be impossible to know a priori. It was also reported by the authors that the proposed approach could result in violation of a few SLA agreements, this would certainly not be taken likely by potential customers.

Al-maamari and Omara (2015) proposed the use of a modified particle swarm optimization to improve both the Makespan of user workloads and Cloud resource utilization levels. Mosa and Paton (2016) in their work presented a utility function based VM allocation approach to energy conservation, SLA adherence and profit maximization. The work identified optimal allocation of VMs to PMs as a NP-hard problem (Garey and Johnson, 1979) and thus used a meta-heuristic genetic algorithm to achieve this goal in the most profitable way to the CSP. The authors employed a utility factor which was based on expected income less estimated energy, violation and performance degradation costs. The approach recorded improvements in terms of QoS adherence and energy conservation but did not pay attention to resource utilization.

Lawanyashri *et al.* (2016) proposed the use of Fruitfly optimization algorithm to address load balancing, delay time and QoS in Cloud computing. Like most deterministic approach, this approach also monitored the state of VMs and iteratively performs workload migration from overloaded resources, albeit at the VM level rather than at the PM. Sleep-states were also used for energy conservation. The author used the Fruitfly algorithm to locate the destination for a

new workload or workloads being migrated. Experimental results showed improvement across all metrics however comparison were mostly with respect to other stochastic approaches.

Daharwal and Sharma (2017), surveyed numerous Cloud load balancing schemes using various metrics and concluded that Generic Algorithm (GA) performed best across all metrics except for fault tolerance. They then proposed a GA scheme that encoded VMs as genes, PMs as chromosomes and fitness functions were calculated based on energy consumption. Experimental reports showed that the GA based approach was able to conserve energy better and reduced the number of VM migrations. However, the results were based on dataset and benchmarks randomly generated by the authors.

This study is based on a deterministic approach and a summary of works that address the challenges of load balancing, QoS adherence and energy conservation using related approaches are summarized in Table 1.

Table 1: Comparison of Related Works

Authors	Title	Methodology	Strengths	Weaknesses
Beloglazov and Buyya (2012)	Optimal Online Deterministic Algorithms and Adaptive Heuristics for Energy and Performance Efficient Dynamic Consolidation of VMs in Cloud DCs.	Developed the PABFD for resource allocation. Power-Aware-VMC for energy conservation and QoS adherence.	Fast resource allocation. Proposed various PM utilization metrics. Benchmark for recent research works.	Focused on QoS and energy conservation only. Assumed all users workload requirements belong to a single class.
Hieu *et al.* (2015)	Virtual Machine Consolidation with Usage Prediction for Energy-Efficient Cloud Data Centres. (VMCUP)	PABFD for resource allocation. Power-Aware-VMC for energy conservation. Linear Regression (LR) for resource management.	Used PABFD as a benchmark and recorded better resource utilization, QoS and energy conservation.	Slow VM allocation and focused mainly on improving resource utilization. Assumed single class for all users' workload requirements.

Farahnakian *et al.* (2016)	Energy-aware VM Consolidation in Cloud Data Centres Using Utilization Prediction Model	PABFD for resource allocation. LR + K-Nearest Neighbour based UP for VMC and QoS adherence.	Improved resource utilization versus other VMC approaches compared.	Focused on energy and resource usage. Assumed single class for all users' workload requirements.
Hu *et al.* (2016)	Service-Oriented Resource Management of Cloud Platforms	Workload Characterization + use of dynamic VM sizing for resource management.	Grouped workloads into resource pools. Dynamically reallocated workloads to improve resource utilization during execution.	Requires prior knowledge of the content of user's workloads. Assumed workloads all followed a specific request pattern.
Bermejo *et al.* (2016)	Cloud Resource Management to Improve Energy Efficiency Based on Local Node Optimization	Autonomous PMs with Euclidean distance for local resources management. Global controller for VMC.	Reduced inter-nodal communication. Improvements in energy conservation.	Focused mainly on energy conservation. Assumed single class for all users' workload requirements.
Mosa and Paton (2016)	Optimizing Virtual Machine Placement for Energy and SLA in Clouds using Utility Functions	Genetic algorithm (utility functions) for VM placement and PM consolidation	Benchmarked against PABFD and recorded improvement in QoS and energy conservation.	Focus was on energy and QoS only. Assumed single class for all users' workload requirements.
Hieu *et al.* (2017)	VMC with Usage Prediction for Energy-Efficient Cloud Data Centres. (VMCUP-M)	Modified VMCUP to use both CPU and memory to predict the future PM utilization.	Slight improvement in QoS and resource utilization versus VMCUP.	Memory utilization had minimal influence on resource utilization and QoS

2.8 Cloud Simulation Frameworks

Numerous simulators exist but most are targeted towards Grid environments (Foster and Kesselman, 2003) and are not able to cater for all the five essential characteristics of Cloud Computing (Mell and Grance, 2011) especially rapid elasticity and pay per use. GridSim (Buyya and Murshed, 2002), GangSim(Dumitrescu and Foster, 2005), SimGrid (Legrand *et al.*, 2003) and GreenCloud (Kliazovich *et al.*, 2012), CloudAnalyst (Wickremasinghe *et al.*, 2010) and CloudSim(Calheiros *et al.*, 2011) are common examples of such simulators.

GridSim for instance is geared towards simulating virtualized organizations with resources connected to a grid network. It is therefore only able to model grid computing entities and corresponding users. GangSim is a tool for studying various scheduling techniques in grid environments. SimGrid is more generic and focused at simulating distributed applications on Grid platforms. GreenCloud is built on NS2 and focuses on energy efficient packet routing within a Cloud DC. CloudAnalyst is essentially CloudSim but with the introduction of a graphical user interface.

A major difference between Grid and Cloud environments is the use of virtualized infrastructure in the Cloud. These virtualized infrastructure are made available to users as platforms upon which workloads and applications can be executed or hosted environment. Apart from infrastructure, the Cloud also provides for programming (PAAS) and application services (SAAS) which are not present in the Grid. Therefore a unique simulation framework is needed for Cloud environments. The aforementioned simulators (except CloudAnalyst) though effective in their own respect are not suited for modelling the various entities and individual service models which the Cloud offers. This is the IAAS, PAAS and SAAS (Banerjee *et al.*, 2015), hence the choice of CloudSim for this work.

The CloudSim toolkit is thus used in this study to simulate, validate and benchmark the performance of MC-BAL against other approaches. It is chosen because of its ability to model virtualized environments, as well as its support for on-demand resource provisioning and dynamic workload allocation, which are key features of CC, lacking in other simulators.

2.9 Analysis of benchmark approaches

Most of these works reviewed above use a two-phased approach to resource management. These phases are described as follows:

In the first phase, the focus is on effectively allocating workloads to PMs with respect to certain criteria such as energy utilization and/or delay time.

In the second phase, the workloads allocated in phase one, are re-allocated with a view of achieving a balanced distribution of workloads across PMs in the DC. The aim of this is to improve utilization of resources across the entire data centre and avoid a situation whereby some PMs are running idle to the detriment of others that are over worked.

Of particular interest to this study are the pioneering work of Beloglazov and Buyya (2012), and a more recent work by Hieu *et al.* (2017), against which this work is benchmarked. These works are used as benchmarks for the following reasons:

1. PABFD is a pioneer work and is often used as a benchmark for recent works.
2. PABFD presented multiple VM selection schemes, hence making it a fair comparison for the class-based VM selection algorithm proposed in this study.
3. VMCUP-M is selected because MC-BAL proposed in this work uses the same usage prediction model for resource utilization, but with the introduction of the 2DHIS which sped up the workload allocation process.
4. Both PABFD and VMCUP-M use the same First Come First Serve (FCFS) workload allocation scheme used by MC-BAL.
5. MC-BAL used a similar energy conservation scheme as both works.

A comparative analysis of the approaches used in these work, their features and limitations are discussed in the following sub-sections.

2.9.1 Analysis of Power-Aware Best Fit Descending

2.9.1.1 The Power-Aware Best Fit Descending

Beloglazov and Buyya (2012) had proposed an approach that focuses on energy conservation while ensuring adherence to QoS requirements. It is a two-phased approach, with phases described as follows:

In phase 1, user workloads (VMs) are admitted and allocated on to PMs using a modified Best Fit Descending algorithm called Power-Aware Best Fit Descending (PABFD). It is a First-Come-First-Server workload allocation scheme with all PMs are sorted in descending order of their processing capabilities. PABFD, then performs a "power growth test" before allocating a VM to a PM. The power growth test, ensures that a VM is only allocated to a seemingly suitable PM, if and only if such allocation would not lead to the power consumption of the proposed PM becoming greater than a pre-set threshold value, otherwise a different PM is selected.

In Phase 2, load balancing of the VM placements done in phase 1 is carried out. Here, the PMs' CPU utilization level are compared against pre-set upper and lower threshold values to determine if such PMs are over/under worked. If the CPU utilization of a PM grows above the upper threshold, VMs are selected and migrated off that PM. Similarly, if its CPU utilization level is below the lower threshold, all VMs are migrated off it and the PM put to sleep to conserve energy.

Two algorithms were proposed by the authors; the first is for tasks allocation, while the second is for load balancing of the allocated tasks.

Algorithm 1: Allocation of Workloads

1. Get the set of requests (VMs) V = {v_1, v_2, v_3…. v_m}
2. Sort all VMs in descending order of CPU utilization
3. Get the set of servers H, H = {h_1, h_2, h_3….h_n}
4. For each v_j in V
    a. Set minPower to MAXPowerAllowed
    b. Set allocatedServer to Null
    c. For each h_i in H
        i. If h_i has enough resource to cater for v_j then
            1. Estimate h's newPower
            2. If h_i's newPower < minPower
                a. allocatedServer = h_i
                b. minPower = newPower
            3. Else select next h that is (h_{i+1})
        ii. Else select next h (that is h_{i+1})
    d. If allocatedServer is not empty then
    e. Allocate v_j to h_i

5. Return allocation

Algorithm 2: Load Balancing of Workloads

Select a VM to migrate

1. For each h_i in the list of servers H
    i. Get set of all VMs allocated to h_i (that is allocatedVMs)
    ii. Sort the VMs in descending order
    iii. hUtil = current utilization level of h_i
    iv. bestUtil = MAX
    v. While hUtil > UpperThreshold //that is, h_i is overloaded
        1. For each v_j in allocatedVMs
            1. if vUtil > (hUtil – upperThreshold) then
                i. tempUtil = vUtil − hUtil + upperThreshold
                ii. if tempUtil < bestUtil then
                1 bestUtil = temp
                2 Set vmWithHighestUtilImpact = v_j
            2. Else
                i. if bestUtil = MAX then
                ii. Set vmWithHighestUtilImpact = v_j
                iii. break
        2. hUtil = hUtil – vmWithHighestUtilImpact
        3. Add v_j to VMsToBeMigratedList
        4. Remove v_j from h_i's allocatedVMs
    vi. If hUtil < LowerThreshold then
        1. Add all allocatedVMs of h_j to VMsToBeMigratedList
        2. Remove all v_i from h_i's allocatedVMs
2. return VMsToBeMigratedList

Listing 1: PABFD Algorithms

2.9.1.2 Description of Algorithms

Algorithm 1 focuses on the allocation of workload (VMs) to suitable PMs in such a way that such allocation does not lead to a growth in power consumption of the PM above a pre-set upper threshold.

In Algorithm 2, the resource (CPU) utilization of each PM is monitored against static upper and lower thresholds to ensure that such PM is not over/under utilized. If there exists such a PM, the algorithm iteratively migrates VMs off it until its utilization level is within the thresholds.

2.9.1.3 Analysis of Algorithms

1. The allocation process in Algorithm 1 is an offline process that requires all workloads (VMs) be available before allocation begins. (Algorithm 1, step 1)
2. The requirements of all VMs must also be known beforehand. (Algorithm 1, step 2)
3. Status probes of all PMs must be done prior to VMs allocation. (Step 3)
4. A power growth test is also performed prior to allocation of VMs. (Algorithm 1, step 4ci1)
5. At the worst case, Algorithm 1 needs to run through all PMs in order to allocate all VMs.
6. Algorithm 2 also relies on PM and VM status probes. (Algorithm 2, step 1-4, 6 & 7)
7. The need for probes and growth test before allocation implies increased response time.
8. The migration process of algorithm 2 is based on current CPU utilization levels.

In determining PM power utilization level in Algorithm 1, step 4c1, Equation 4 is used.

$$P(u) = K * P_{max} + (1 + K) * P_{max} * u \quad 4$$

Where: K is fraction of power consumed when system is idle; K = 0.7. P_{max} = Max power consumed by a fully utilized server. u = CPU utilization.

2.9.2 Analysis of Virtual Machine Consolidation with Usage Prediction (VMCUP)

2.9.2.1 Virtual Machine Consolidation with Usage Prediction

Hieu *et al.* (2015) improved on the work done by Beloglazov and Buyya (2012) with the introduction of resource usage prediction. VMCUP is also a two-phased approach to Cloud resource management but with focus on efficient resource utilization, QoS adherence and energy conservation.

In phase one, the authors applied the same PABFD algorithm used in (Beloglazov and Buyya 2012) for initial allocation of tasks. While in phase two, load balancing was done using the current and future resource utilizations. The authors believe that this approach gives a more robust characterization of server status rather than just using a simple threshold based approach. To predict the short-term future resource utilization, a number of past utilization levels were passed into a linear regression model (Equation 5). The resultant values are then checked against a "hot threshold" value to determine whether or not a server is currently over / under utilized or will be in the shortest future.

$$U_{t+1(p)} = \beta_0 + \sum_{i=1}^{m} \beta_i * U_i(p) \quad 5$$

$$U_{t(p)} = \frac{u_t(p)+w(p)}{r\prime(p)} \quad 6$$

$$u_t(p) = \sum_{v \in vm} r(v) \quad 7$$

Where $U_{t(p)}$ = CPU Utilization of a PM (p) at a given time (t). p = Physical Machine (PM). m = total number of p in the DC. $u_t(p)$ = Total CPU utilization of all VMs running on p. w(p) = inherent CPU utilization level; that is, resources used to manage p itself. r' = Total capacity of PM (p). r(v) = resource required by a VM (v). β_0 and β_i are regression co-efficient

The VMCUP algorithms are detailed below:

VMCUP uses the same workload allocation algorithm as PABFD (algorithm 1) discussed in section 2.8.1.1

Algorithm 3: Load Balancing of Workloads

A. MigationProcesses()

//overloaded Servers

1. Get set of PMs H
2. Set m ← 1;
3. for each h_j in H do
    a. While **OverloadDetection** (h_j, m) = true do
        i. v = getVMwithLeastMigTime from h_j

ii. h_s = PABFD(H, v); //use PABDF to get PM with least power-growth
iii. If a suitable h_s is found in H, then place v on h_s, update U(h_s);
iv. Else if no suitable h_s is found,
    1. Wake an idle h_{idle} and allocate v to it.
    2. Place v on h_{idle}, update U(h_{idle});
    3. H += h_{idle} //update set of PMs (H) to include h_{idle}
v. Else break

//underloaded Servers

4. Set h = h_0; //select first PM in H
5. For each h_j in H do
    a. If U(h) > U(h_j) then h = h_j; //get h with smallest utilization level
6. If **UnderloadDetection** (h, m) = true then //migrate all VMs in h and switch h to idle
    a. Set status = true, suitablePMs = null;
    b. For each v_i in h do
        i. h_s = PABFD(H, v_i); //use PABFD to get PM with least power-growth
        ii. If h_s = null then status = false; break;
        iii. Else suitablePMs += h_s //add h_s to list of suitable PMs to migrate VMs to
    c. If status = true then
        i. For each v_i in h do //migrate VMs off underloaded PM (h) to h_s
            1. Remove server h_s from suitablePMs in FIFO order;
            2. Place v_i on h_s, update U(h_s);
        ii. Switch h to a low-power mode;
        iii. H -= h // Update H i.e. remove h from H

B. OverloadDetection(h,m**)**

1. FutureUtil(h) = **PredictionModule**(h,m);
2. If CurrentUtil(h) & FutureUtil(h) > thresh then return true;
3. Else return false;

C. UnderloadDetection(h,m)

1. FutureUtil(h) = **PredictionModule**(h,m);
2. If FutureUtil(h) ≤ CurrentUtil(h) then return true;
3. Else return false;

D. PredictionModule(h,m)

1. Set x, y and β = null
2. For t = 0 to n – m
    a. X_{t0} = 1; //Set initial that x_t to 1, that is at time t = 0
    b. For i = 0 to m do
        i. $X_{t(i+1)} = U_i(h)$;
        ii. X += $X_{t(i+1)}$ //update X to include $X_{t(i+1)}$
    c. $Y_t = U_i(h)$; Y += Y_t; //update Y to include Y_t
3. $\beta = (x^T * x)^{-1} * x^T * y$;
4. $U_{t+1}(h) = U(h) + \beta$ // next utilization level at time t = t + 1
5. Return $U_{t+1}(h)$

Listing 2: Load Balancing of Workloads using VMCUP

2.9.2.2 Description of Algorithms

In Algorithm 3, the current and short-term future CPU utilization levels of PMs are monitored against upper and lower thresholds to ensure that such PM is/will not be over/under utilized.

If an over/under-utilization is detected, the algorithm iteratively migrates VMs off the PM until its utilization returns to a level within the thresholds.

2.9.2.3 Analysis of Algorithms

1. Same as those of algorithm 1 and 2, discussed in section 2.8.1.3.
2. The introduction of a future resource utilization model helps better compliance with SLA/QoS by preventing a future server overload from occurring rather than correcting after it has occurred which is the case with other approaches.

2.9.3 Analysis of Virtual Machine Consolidation with Multiple Usage Prediction (VMCUP-M)

2.9.3.1 Virtual Machine Consolidation with Multiple Usage Prediction

In numerous works the utilization level of a single resource, often times the CPU alone is used as criteria for determining when a PM is over / under utilized and when VMC should be done.

However, in Hieu *et al.* (2017) an approach that takes multiple resources into consideration for VMC was presented. The approach uses the current and future utilization of the CPU, memory and bandwidth to determine the status of PMs and when VMC should be done. It was argued that the use of multiple resources led to better characterization of PMs thus reducing the number of VMC. The approach is similar to VMCUP but with the inclusion of a multi-iterative multiple resources prediction model. The analysis of the algorithms presented are discussed in the next subsections.

Algorithm 4: Allocation of Workloads using PABFD-MUP

1. P =0 and minPower = MAX
2. Get set of PMs H = {h_1, h_2, h_3 … h_m}
3. Get set of VMs V = {v_1, v_2, v_3 … v_n}
4. For each h in H
5. For each resource type r in H
    a. If v + $u_r(h_i)$ + $baseline_r(h_i)$ <= $cap_r(h_i)$
        i. oldPower = getPower(h_i)
        ii. Allocate v to hi and update utilization of r on h_i
        iii. newPower = getPower(h_i)
        iv. if (newPower – oldPower)< minPower and OHD-MUP = false
            1. minPower = newPower – oldPower
        v. else De-allocate v from h_i, and update utilization of r on h_i
6. Return h_i

Algorithm 5: Load Balancing of Workloads

A. MigationProcesses()

//overloaded Servers

1. Get set of PMs H
2. for each h_j in H do
    a. While **OHD-MUP** (h_j, D, m, K) = true do
        i. v = getVM_MRT(h_j)
        ii. h_s = PABFD-MUP(H, v, D, m, K);
        iii. If a suitable h_s is found in H, then place v on h_s, update $U_r(h_s)$;
        iv. Else if no suitable h_s is found,
            1. Wake an idle h_{idle} and allocate v to it.

2. Place v on h_{idle}, update $U(h_{idle})$;
3. H += h_{idle} //update set of active PMs (H) to include h_{idle}

v. Else break

//underloaded Servers

3. Set h = h_0; //select first PM in H
4. For each h_j in H do
    a. If $U_r(h) > U(h_j)$ then h = h_j; //get h with smallest utilization level
5. If **UHD-MUP** (h, D, m, K) = true then //migrate all VMs in h and switch h to idle
    a. Set status = true, suitablePMs = null;
    b. For each v_i in h do
        i. h_s = PABFD-MUP(H, v_i);
        ii. If h_s = null then status = false; break;
        iii. Else suitablePMs += h_s //add h_s to list of suitable H to migrate VMs to
    c. If status = true then
        i. For each v_i in h do //migrate VMs off underloaded PM (h) to h_s
            1. Remove h_s from suitablePMs in FIFO order;
            2. Place v_i on h_s, update $U_r(h_s)$;
        ii. Switch h to a low-power mode;
        iii. H -= h // Update H ie remove h from H

B. OHD-MUP(h, D, m, K**)**

1. For all resource r in D
2. For time k = current to short-term future (K)
    a. If $U_{r, t+k}(h)$ = **PredictionModule(**h, r, m + (k -1)**)**
    b. If $U_{r, t+k}(h)$ < thresh then return false;
3. Else return true

C. UHD-MUP(h,D,m,K**)**

1. For all resource r in D
2. For time k = current to short-term future (K)
    a. If $U_{r, t+k}(h)$ = **PredictionModule(**h, r, m + (k -1)**)**
    b. If $U_{r, t+k}(h) > U_{r, t}(h)$ then return false;

D. PredictionModule(h,r,m)

Same as algorithm 3, but repeated for multiple resources

Listing 3: VMCUP-M workload allocation and load balancing algorithms

2.9.3.2 Description of Algorithms

Similar to algorithm 1, in algorithm 4 workloads are also allocation onto suitable PMs in such a way that minimizes growth in power consumption. PABFD-MUP however also takes the multiple resources (CPU, memory and bandwidth) within PMs into consideration when allocating workloads. The effect of allocating a workload (demanding a certain resource from a potential PM) might have on the power level of such PM is also considered prior to final allocation. Algorithm 5, is an extension of algorithm 3 and considers multiple resource when determine the utilization level of PMs.

2.9.3.3 Analysis of Algorithms

1. In algorithm 4, a power-growth test (Step 5a,iv) needs to be conducted D number of times, where D is the number of resources within the PM being considered. These tests can potentially slow down the allocation phase.
2. The authors did not provide a generalized function that encompassed the power growths obtained from each of the individual resource. This can lead to a situation whereby an increase in power for one resource has an inverse effect when considering a different resource.
3. In both algorithms 4 and 5, apart from the CPU, other resources within a PM do not have varied power states and are either ON or OFF (Lefurgy *et al.*, 2007, Rice *et al.*, 2015). This implies that once ON these resources would consume the same amount of power irrespective of the amount of workload being serviced. Thus limiting their suitability as PM utilization thresholds.

2.10 Summary

In all the literature reviewed, a common assumption made was that all user workload requirements were similar and thus could be characterized and grouped together. This is not the case in reality, as users generally vary either as a result of socio-economic factors, purchasing power or simply requirements.

Furthermore, most of the works in literature only considered one or two of these challenges, ignoring the other(s). For those that considered all three, energy conservation was inferred to be a by-product of efficient resource utilization, or vice versa. This work therefore aims to simultaneously address these three challenges by using the class of users' workloads.

CHAPTER THREE
RESEARCH METHODOLOGY

3.1 Introduction

This chapter discusses the research approaches used to achieve the objectives as stated in section 1.3. In meeting these objectives, this study proposed MC-BAL and a summarized description of the MC-BAL model is done in section 3.2. In meeting the first objective of designing a scheme that ensures adherence to end-to-end, pre-set QoS levels while providing services to users, MC-BAL used a combination of class-based workload allocation and a modified half interval searching technique called 2DHIS for allocation of PMs to VMs. This combination is discussed in section 3.3. MC-BAL used class-based VMC, PM auto-scaling and usage prediction to meet the second objective of efficiently utilizing Cloud resources while providing services to users; this is discussed in section 3.4. The third objective of improving the overall energy consumption of Cloud DCs, was addressed by MC-BAL through monitoring CPU utilization and PM sleep-states. This objective is discussed in section 3.5. The policies implemented by MC-BAL at various phases with respect to the research objectives are presented in section 3.6. The entire system process flow is presented in section 3.7, while metrics for evaluating MC-BAL are discussed in section 3.8. Finally, the experimental framework used in this study is discussed in section 3.9.

3.2 Definition of the MC-BAL Model

This study proposes Multi-Class Load Balancing (MC-BAL) for resource management in CC. It focuses on utilizing Cloud resources (PMs) efficiently and in a manner that consumes the least amount of energy, yet adhering to users' QoS requirements.

3.2.1 The MC-BAL Model

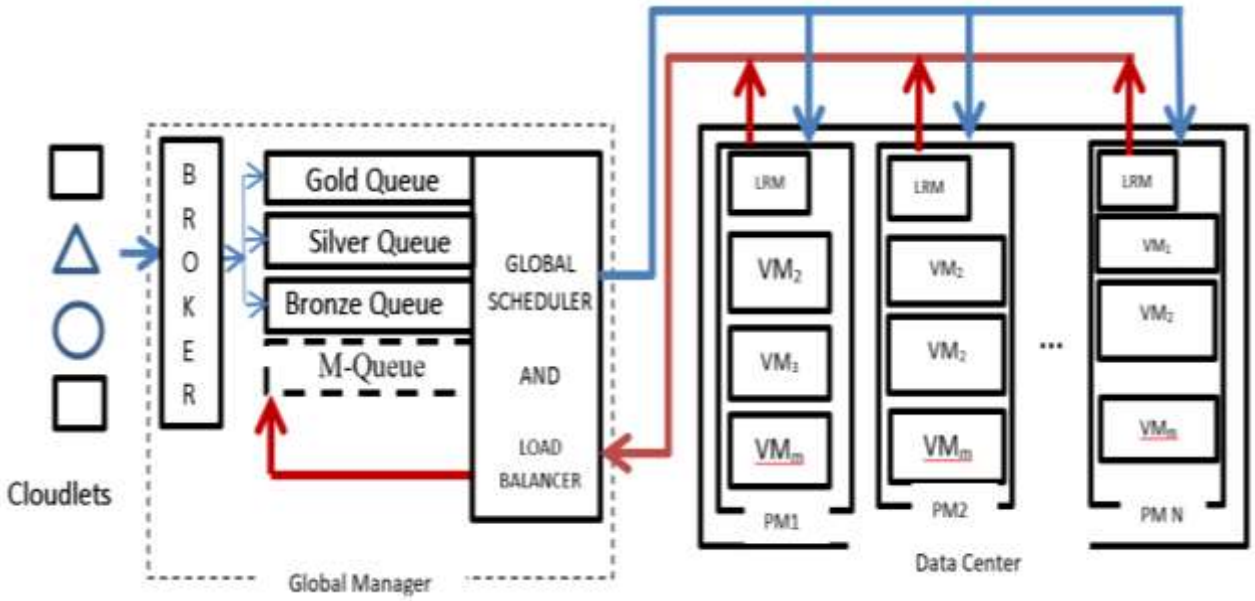


Figure 8: The Proposed MC-BAL Model (Ajayi, 2017)

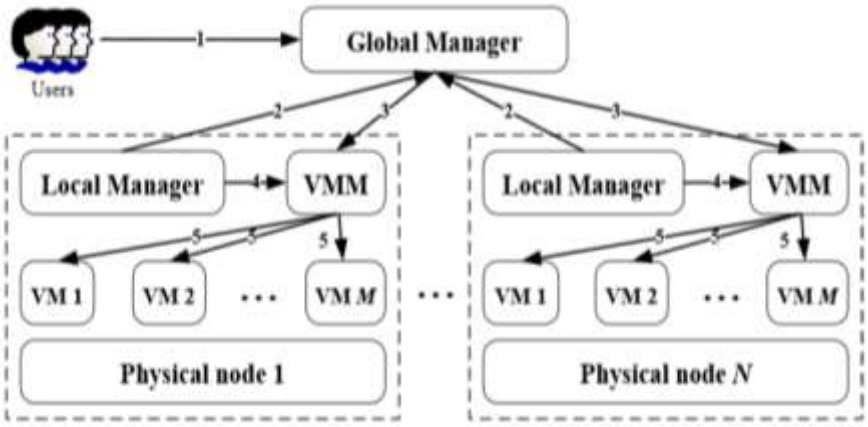


Figure 9: PABFD Model (Beloglazov and Buyya, 2012, Hieu *et al.*, 2017)

The MC-BAL model is shown in Figure 8, while the PABFD model is shown in Figure 9. Though the MC-BAL model is an adaptation of the PABFD model, there are significant differences between the two models. A description of these components and how they differ across both models is as follows:

1. Users / Cloudlets: In both models users submit workload to the system for processing. These workloads are assigned to distinct VMs, and viewed as VMs within the DC. MC-BAL however makes distinctions between the classes of users' requirements, which PABFD does not.

2. Global Manager (GM): This acts as a mediator between the users and the Cloud DC. In both models, the GM is made up of a broker and a scheduler but their functionalities differ.
    a. The broker: In PABFD, users submit requests to the broker, the broker then converts this requests to VMs and forwards the request to the scheduler. In MC-BAL, the broker does an additional job of classifying the user requests into appropriate queues of the scheduler based on user specified burst times.
    b. The Global Scheduler: In both models the global scheduler allocates submitted tasks (VMs) onto PMs via a scheduling policy.
        i. In PABFD, the Global Scheduler, allocates workloads to PMs, using linear search, while in MC-BAL, the allocation is done using 2DHIS (a modified binary search algorithm).
        ii. In PABFD, there is only one queue and a simple FCFS allocation policy is used. In MC-BAL however, there are four queues viz. Gold, Silver, Bronze and the M-Queue. The Gold, Silver and Bronze represent classes of user requirements with Gold given the highest priority and Bronze the least. Classification of workloads into appropriate queues by MC-BAL ensures that user requirements are taken into consideration right from the allocation phase and not just at the load balancing phase as is the case with PABFD.
3. Physical Nodes / PMs: In both models, these represent the physical server machines within a Cloud DC and serve as hosts for user workloads (VMs).
4. Local Resource Monitor / Local Manager: The Local Resource Manager (LRM) is present in both models and serve the same functions. The LRM is installed on all PMs and is concerned with monitoring the resource utilization levels of its PM as well as reporting this to the Global Scheduler for load balancing.
5. VMM: This is only available in the PABFD model and it collaborates with the Global Scheduler to perform VM Migration (load balancing). This function has been incorporated into the LRM in MC-BAL.
6. M-Queue: The Migration-Queue (M-Queue) is a virtual queue used by the Global Scheduler for load balancing. It is a queue onto which VMs to be migrated off a PM are enqueued. It has a higher priority than any of the other queues; this is done in order to prevent SLA violations (from queuing delay), as workloads enqueued on it had previously been allocated and were being serviced prior to being selected for migration.

3.3 Maintaining End-to-End Pre-Set QoS Levels

MC-BAL addressed adherence to pre-set QoS levels from two perspectives –guaranteed end-to-end QoS adherence and lowered resource allocation time (workload delay).

3.3.1 Guaranteed End-To-End QoS

MC-BAL is a class-based workload allocation and migration scheme, which ensures adherence to end-to-end QoS levels both at the allocation and at the load balancing phases. Given that there are three classes of workloads, all submitted workloads must belong to one of these three classes. At the allocation phase, MC-BAL ensured that workloads with higher priority were given preference, thus a Gold class workload would be admitted before a Silver, and then a Bronze. Therefore, right from the admission phase, QoS was being adhered to. MC-BAL relied on user specified burst time as the only criterion for classifying workloads. This was because all other criteria could be inferred from the burst time. The relationship between burst time, payment band, response time and users' classes is shown in Table 2.

Table 2: Classification of workload

CLASS	REQUIRED RESPONSE TIME	PAYMENT CLASS	BURST TIME
Gold	No delay allowed	Premium	20 % of all submitted jobs
Silver	Some delay allowed	Standard	40 % of all submitted jobs
Bronze	Best Effort	Basic	40 % of all submitted jobs

In practice, users would specify the requirements from their CSP. In this work, these requirements were represented as the burst times; and they specified how much of the PM's processor the users' workload required. Considering that this study was based on logs of workloads previously submitted and processed in various DCs, it was impossible to ask users to specify their respective burst times. To this end, twenty percent (20 %) of all workloads in the log were classified as Gold, 40 % as Silver and the last 40 % as Bronze class. This is as shown in Table 2.

During the load balancing phase workload classes were also considered; this prevented indiscriminate migration which was a major shortcoming of most other approaches in literature and a major cause of SLA violations. MC-BAL prevented workloads with higher priority from

being selected for migration, when workloads with lower priority were present within the PM. It achieved this by selecting workloads for migration based on priority levels; such that:

$$\text{Gold} > \text{Silver} > \text{Bronze} \quad (8)$$

This implies that the Gold class was given highest preference and guaranteed uninterrupted service (Ajayi *et al.*, 2017). The Silver class was considered next and lastly the Bronze class. Workloads selected for migration were enqueued on the virtual migration queue (M-Queue). This Class-based VM selection process is described and modelled below:

All workloads (VMs) allocated to PM must belong to one of the three classes (Gold, Silver or Bronze). Once a PM has been identified as overworked, the VM migration process was activated and a VM had to be selected from it for migration. This selection is modelled as follows:

Let B, S and G represent Bronze, Silver and Gold workload classes respectively. Let N, B_T, S_T and G_T respectively represent the total number of user workloads in the entire system, total number of B, total number of S and total number of G with N being the summation shown in Equation 9.

$$N = (B_T + S_T + G_T) \quad (9)$$

Let P_T represent the total number of VMs allocated to a given PM p, such that P_B, P_S and P_G are the numbers of B, S and G in p.

Let X be a VM selected for migration (without replacement), the probability of it being B is given by the hyper-geometric distribution (Weisstein, 2003) shown in Equation 10.

$$P(X = B) = \frac{\binom{B_T}{P_B} * \binom{N - B_T}{P_T - P_B}}{\binom{N}{P_T}} \quad (10)$$

The probability of a Silver being selected, that is P(X = S), is a conditional probability which could only occur if and only if all Bs had previously been selected. This meant that the probability of selecting S was dependent on the previous selection(s) being B or another S (if all Bs had previously been selected). This is modelled using the Bayesian model of two elements (Ghahramani, 2013) but with an added condition and described in Equation 11.

$$P(X = S) = \begin{cases} P(S\,|B) = \frac{P(B \cap S)}{P(B)}\ ,\ given\ that\ P(B) > 0 \\ P(S),\ given\ that\ P(B) = 0 \end{cases} \quad 11$$

The probability of the selection being a G can also be modelled using the Bayesian model but for three elements and described in Equation 12.

$$\mathrm{P(X = G)} = \begin{cases} \mathrm{P(G|B \cap S)} = \dfrac{\mathrm{P(B \cap S \cap G)}}{\mathrm{P(B) * P(S|B)}}, & given\ that\ P(B) and\ P(S) > 0 \\ P(G), & given\ that\ P(B) and\ P(S) = 0 \end{cases}$$

12

In Equation 11, the probability of any S being selected is dependent on all previous selections being Bs or another Ss (if no more Bs were present). While in Equation 12, the probability of the selected workload being a G is dependent on the probability of all previous selections being S or G (given that only Gs are left in the PM).

The Class-based VM selection algorithm is shown in listing 4, algorithm 6

Algorithm 6: The Class-Based VMC

1. Foreach PM (p) check status //*that is CPU utilization level*
2. If p is UNDERUTILIZED (Algorithm 8) //current_*utilization* > *threshold*
    a. Perform VMM (p, all VMs in p) //*migrate ALL vms*
    b. If step b is successfully completed put p to sleep
3. If p is OVERUTILIZED (Algorithm 9)
    a. For all VMs in p check VM_types
    b. If any VM_type = BRONZE
        Select it for migration //*bronze should be selected first*
        Else if any VM_type = SILVER
        Select it for migration //*select silver in the absence of bronze*
        Else if any VM_type = GOLD
        Select it for migration. //*select gold is silver & bronze are absent*
    c. Perform VMM(p, selected VM in p)
4. VMM (p, v)
    a. Enqueue all VMs in v on M-Queue
    b. Reallocate all vm other PMs (except p) using 2DHIS (Algorithm 7)

Listing 4: Class-Based VMC for MC-BAL

3.3.2 Lowered Resource Allocation Time

In order to ensure minimal compromises on QoS requirements as a result of resource allocation delay, MC-BAL used the Double-Depth Half-Interval Search (2DHIS) algorithm (Ajayi *et al.*, 2016) to find suitable PMs for user workloads. 2DHIS is a modified Red-Black Tree (RBT) (Hanke, 1999) with the introduction of two-dimensional nodes. A RBT was built from all the PMs in the DC based on their available CPU. Each node of the tree was made up of an array of PMs having equal CPU utilization levels, instead of a two-dimensional structure like a normal RBT, 2DHIS is a three-dimensional data structure, with thickness equal to the size of the PM array. The 2DHIS is shown in Figure 10 with A to G representing the available processing capabilities (CPU utilization levels) of PMs within the DC.

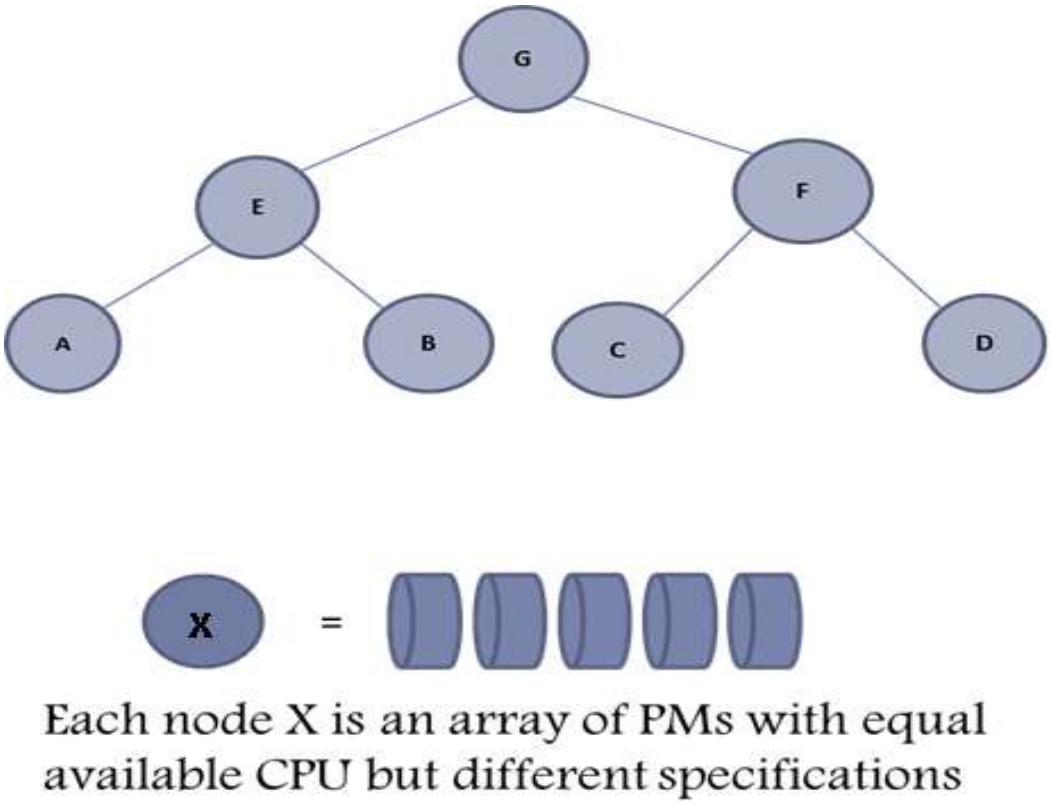


Figure 10: The 2DHIS Model

Being a Red Black Tree (RBT), 2DHIS has an average, best and worst case search time complexity of $O(\log_2 n)$ (Hanke, 1999). For large number of PMs (a common phenomenon in Cloud DCs), 2DHIS was much faster in finding suitable hosts than the average and worst cases of the linear array search used in other works in literature.

2DHIS ensured that only the PMs which best matched the stipulated requirements (burst time) of the workloads were selected. It achieved this by recursively searching the tree for a node such that the difference between the available CPU and workload requirement is almost zero as shown in Equation 13.

$$P_c - \sum_i^n W_i \rightarrow 0 \quad 13$$

Where P_c is PM's available CPU, n is number of workloads, W is workload requirement.

If such a node was found, 2DHIS then iterated through the node's PM array searching through the potential PMs for the most suitable. This process further ensured that only the PM which perfectly matched the workload requirement was selected. The 2DHIS algorithm is shown in listing 5.

Algorithm 7: The 2DHIS Algorithm

1. Arrange all PMs in ascending order of their available processing capacity (LcP)
2. Build a Red-BlackTree (BT_LcP) of available processing capacity from LcP
3. Accept VMs to be allocated (VM_Set) //*VMs = workloads*
4. Foreach vm in VM_Set
    a. Get vm's requirement (wR) //*CPU requirements of the workload/VM*
    b. SuitablePM = getHost(wR, BT_LcP)
5. getHost(wR, BT_LcP) //*recursive search for the best match PM*
    a. Search BT_LcP for a node n, such that n.AvailableMIPS - wR is almost zero
    b. If found,
        i. Search through n (potential ps) for a suitable p
        ii. Return p //step *i is required if other resource are to be considered*
    c. Else
        i. Remove p from BT_LcP and update BT_LcP
        ii. Return getHost(wR, BT_LcP)
6. If SuitablePM = null
    a. Get updated PM list
    b. Rebuild BT_LcP //*a rebuild is required because auto-scaling adds new PM often*
    c. Goto step 4

Listing 5: The 2DHIS algorithm

3.4 Ensuring Effective Utilization of Cloud Resources

In ensuring efficient utilization of Cloud Resources, MC-BAL used a combination of VMM, Usage Prediction and Auto-Scaling at both the allocation and load balancing phases. At the allocation phase, the 2DHIS algorithm was used to find the best PM-VM match, thus ensuring that from the allocation phase, resource wastage as a result of ill-proportioned distribution was mitigated.

3.4.1 Resource Auto-Scaling

MC-BAL applied auto-scaling at the allocation phase. Auto-scaling is a direct implementation of the rapid provisioning characteristics of CC as defined by the NIST (Mell and Grance, 2011). Conventionally, CSPs cannot know the exact number of PMs or resources to earmark for users' workloads a priori. Hence, they either guess, use historic data or rely on the user specified requirements, which in most cases lead to over provisioning. Most works in literature such as those of (Islam *et al.*, 2010, Wu *et al.*, 2011 and Batista *et al.* 2015) tackled auto-scaling from the perspective of the users' application type or requirement, and then scaled the Cloud resources in response to application demands. This study however is not concerned with the exact content or type of users' workloads, hence such approaches cannot be used. MC-BAL thus auto-scaled by monitoring resource levels against workload volume. It then gradually released more resources into the system if it detected an imbalance between the resource levels and workload volume. This process ultimately allowed MC-BAL to use only the barest minimum resources to cater for all users' workloads.

3.4.2 Determining Current PM Utilization Level

MC-BAL ensured that workloads were distributed evenly across PMs in order to avoid situations where certain PMs were underworked at the detriment of others. In order to determine when load balancing (workload migration) was needed, MC-BAL checked the current and short-term future CPU utilization levels (P_i and P_{i+1}) of the PM. To determine the current CPU utilization level, MC-BAL used any of four indicators. These four indicators are of two types - static and dynamic. For the static, upper and lower CPU utilization threshold values of 80 % and 20 % respectively were used. While for the dynamic threshold schemes, the Median Absolute Deviation (MAD) and Inter-Quartile Range (IQR) were used. The dynamic schemes provided adaptive upper CPU thresholds which varied constantly with the PM utilization. These four indicators are described as follows:

1. Static Upper Threshold:

   A static upper CPU utilization threshold value of 80 % was set; above which a PM was considered overloaded and VMs had to be selected from it for migration and consolidation on different PMs.

2. Dynamic Upper Threshold: Median Absolute Deviation (MAD)

   This is a statistical measure of the deviation of a PM's CPU's utilization level from the median of all other PMs' CPU's levels in the DC. It was adopted from Beloglazov and Buyya (2012) and shown in Equations 14 and 15. A sentinel value s was used to control the rate of VM consolidation with respect to QoS and energy conservation.

$$MAD = median_i\ (|X_i - \ median_j\ (X_j)|) \quad 14$$

$$Upper\ Threshold\ = 1 - \ s.MAD \quad 15$$

Where X_i is CPU utilization level, X_j is median utilization level, and s is consolidation factor.

3. Dynamic Threshold: Inter-Quartile Range (IQR)

   This is a statistical measure of the mid-spread and is defined as the difference between the third and the first quarter. It was adopted from Beloglazov and Buyya (2012) and shown in Equation 16

$$Upper\ Threshold\ = 1 - \ s.(Q_3 - Q_1) \quad 16$$

4. In determining if a PM was underutilized, MC-BAL used a static lower threshold value of 20 % below which the PM was marked as underutilized and all VMs on it were migrated off and consolidated on different PMs.

The VM selection algorithms for over-utilization and under-utilized PMs are shown in listing 6 and 7 respectively.

Algorithm 8: Over-utilized PM Detection Algorithm

1. Set upper threshold = U_{TH} (Static or dynamic) *//U_{TH} = upper utilization threshold*
2. Foreach p in set of PMs
   a. $U_{t+1}(p) \leftarrow UP(p, m)$
   b. If $U_t(p) > U_{TH}$ and $U_{t+1}(p) > U_{TH}$

Return true //*returns true if the PM is underutilized and would remain so in the future*

c. Else Return false

Listing 6: PM over-utilization detection algorithm used in MC-BAL

Algorithm 9: Under-utilized PM Detection Algorithm

1. Set lower threshold = L_{TH} //L_{TH} = lower CPU utilization threshold
2. Foreach p in set of PMs
   a. $U_{t+1}(p) \leftarrow UP(p, m)$ //*m = an array of previous CPU utilization levels*
   b. If $U_{t+1}(p) \leq L_{TH}$ *//returns true PM would still be underutilized in the future*
      Return true
   c. Else *//returns false if the PM would be over-utilized in the future*
      Return false

Listing 7: PM under-utilization detection algorithm used in MC-BAL

3.4.3 Predicting Future PM Utilization Level

In order to determine the short-term future CPU utilization of a PM, MC-BAL used the CPU utilization prediction model proposed by Hieu *et al.* (2015). The model was able to predict the short-term future utilization of PMs using their historic CPU utilization levels. This model has been discussed in section 2.8.2.

Using a combination of the current and future CPU utilization levels, MC-BAL was able to manage the resource utilization. If a PM was determined to be over-utilized, both in the current and short-term future, VMs were selected from it for migration and consolidated onto different less worked PMs using the class-based migration policy described in section 3.3.1.

3.5 Improving Energy Conservation in Cloud Data Centres

In a bid to conserve the overall energy consumed, MC-BAL like other similar works, actively monitored the status of all PMs. Once a PM's CPU utilization fell below the lower threshold value, all VMs allocated to the PM were migrated off it and the PM was switched to sleep-state. Unlike in other works however, MC-BAL introduced an additional step to the energy conservation process. In this step, though a PM might not have been considered under-utilized,

if all workloads on such PM belong to the Bronze class, MC-BAL attempted to migrate these workloads onto other PMs. If successful, the PM was switched to sleep-state.

This additional process further improved MC-BAL's energy conservation. By switching PMs to sleep-state, an energy saving of up to 90 % could be obtained; as Pennsylvania (2013) has shown that a PM in sleep-state consumes less than 10 % of its maximum usable power.

3.6 Objectives and Corresponding Policies of MC-BAL

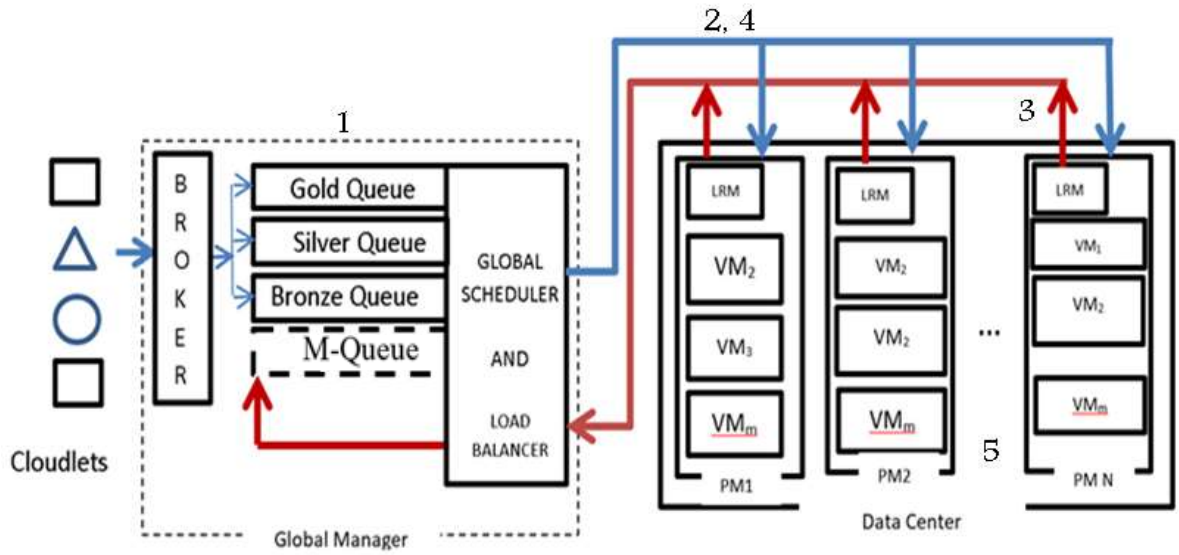


Figure 11: MC-BAL Policies

Table 3: Summary of Objectives and corresponding policies

OBJECTIVES	**POLICIES**
Objective 1	1: Classification of workloads 2: Allocation of VMs to PMs using 2DHIS
Objective 2	3: Class-based VMC for selecting VMs from over-utilized PMs 4: PM auto-scaling is used to manage number of active PMs.
Objective 3	5: CPU utilization levels and PM sleep-states

Figure 11 and Table 3, show where the various MC-BAL policies described in sections 3.3 to 3.5 were used with respect to achieving the objectives of this study. Figures 12 and 13 show the MC-BAL system flowchart.

3.7 System Process Flow

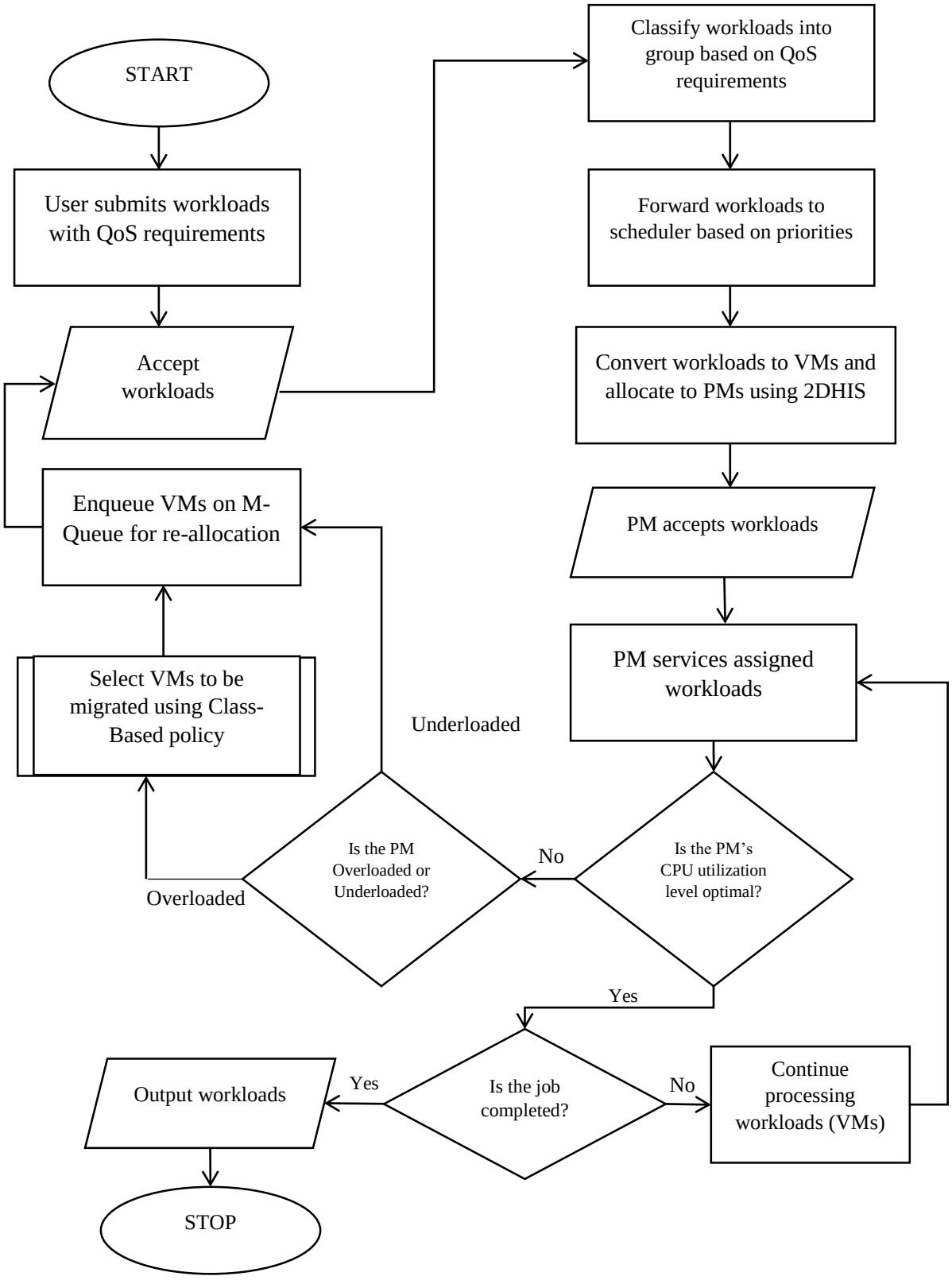


Figure 12: MC-BAL System Flow chart

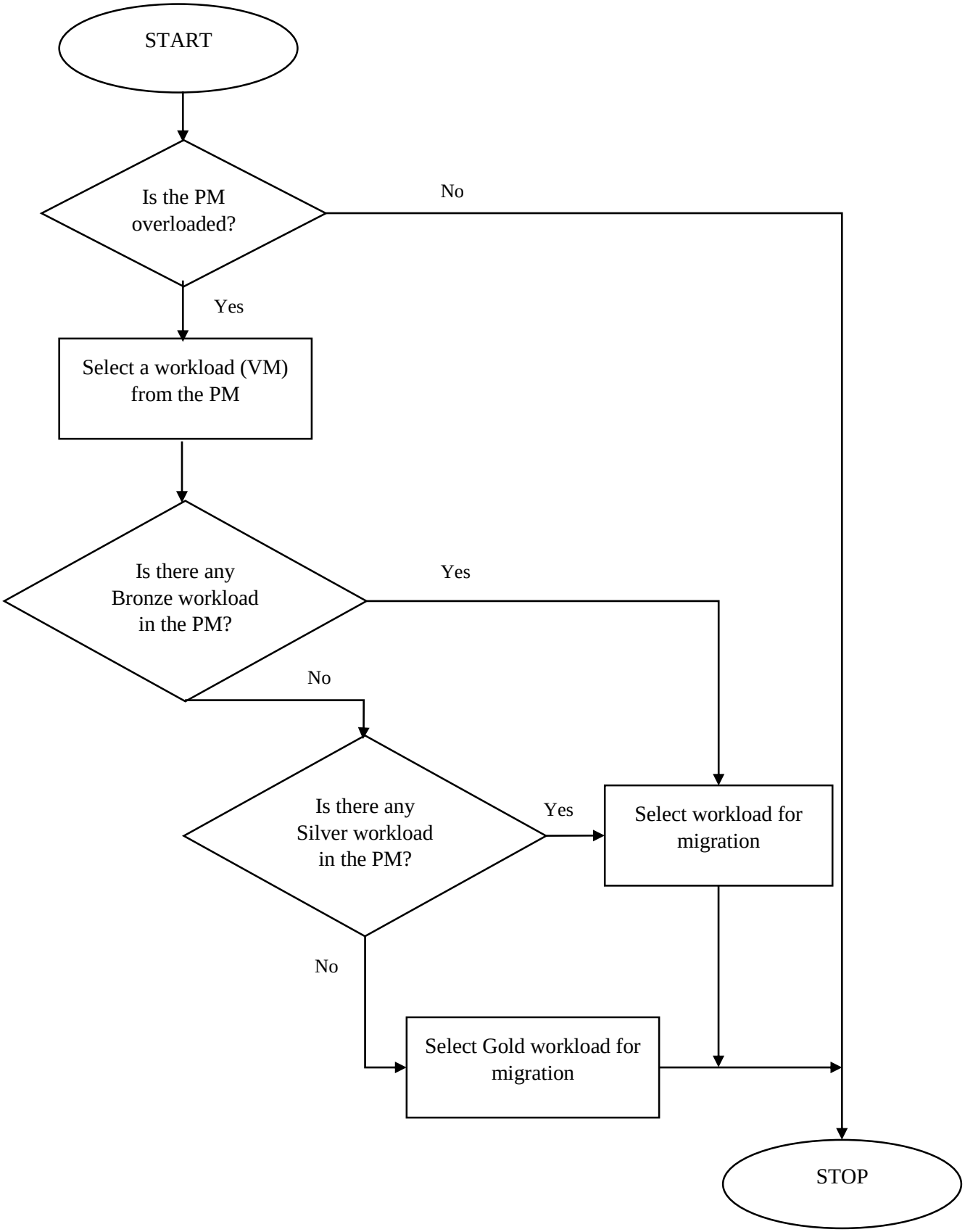


Figure 13: Class-Based VM Selection from Overloaded Physical Machines

3.8 The Metrics for Evaluation

3.8.1 Evaluation Metrics

In order to benchmark MC-BAL and validate its performance against other state of the art approaches, evaluation metrics were required. To measure compliance with user stipulated QoS requirements, percentage of SLA violation and average workload delay were used. Lower SLA violation percentage and delay values were desired.

In determining effectiveness of MC-BAL in terms of resource utilization, capacity utilization was used. While for energy conservation, energy consumption levels and number of Power State Changes (PSC) per PM were measured. Lower values were also desirable for these metrics. All five metrics are discussed in the following sub sections.

3.8.2 SLA Violation

This metric is adopted from (Beloglazov, 2013) and is the percentage of time during which a PM hosting VMs experiences utilization greater than the upper threshold, hence unable to provide service at agreed levels to the user. It is defined in Equation 17.

$$SLAV = \frac{1}{n} * \sum_{j=1}^{n} \frac{T_{over_j}}{T_{total}} \quad 17$$

Where n is number of PMs. T_{over} is the period during which the PM is operating above its upper threshold. T_{total} is the total time during which the PM is actively serving VMs.

3.8.3 Average Workload Delay

The average time spent by workloads waiting to be allocated to PMs. It is defined in Equation 18.

$$\delta_{avg} = \frac{1}{n} * \sum_{i=0}^{n} \left[T_d^k(i) - T_a^k(i) \right] \quad 18$$

Where n is number of jobs on the queue. T_d^k (i) and T_a^k (i) are respectively job departure and arrival times on any given queue k.

3.8.4 Capacity Utilization

A percentage ratio of the number of PMs in use in a data centre to the total number of PMs in the data centre. It is adopted from the works of Kang and Kim (2015) and Johansen (1968) and given by:

$$C_u = \frac{T - T_u}{T} * 100\ \% \quad 19$$

Where C_u is capacity utilization. T is total number of PMs. T_u is total number of unused PMs

3.8.5 Energy Consumption

The total energy consumed by all the active PMs in the DC. This is adopted from the work of Beloglazov (2013) and given in Equation 20.

$$P_{tot} = \sum_{j=1}^{n} k * P_{max} + (1 - k) * P_{max} * u_j \quad 20$$

Where P_{max} = 250W. P_{tot} = Total energy of the DC. k = 0.7, fraction of power used by an idle PM. n = number of active PMs in the DC, u_j is current utilization level of a PM j.

3.8.6 Power State Changes per PM

This is a ratio of PMs switched to sleep-state to total number of PMs in the DC and is defined in Equation 21.

$$PSC = \sum T_s : \sum T \quad 21$$

Where PSC= Power State Change ratio. T_s= PMs in sleep-state. T= Total of PMs in the DC.

3.9 Experimental Framework

3.9.1 Experimental Framework

Evaluating the performance of MC-BAL under varying system conditions and user requirements is almost impossible to achieve in a live environment, hence a simulation toolkit was used. In this study, the CloudSim Cloud simulator was used. It was chosen because of its ability to model virtualized environments, as well as its support for on-demand resource provisioning and dynamic workload allocation, which are key features of CC, lacking in other simulators. A review of related simulators was done in section 2.8.

3.9.2 The CloudSim Toolkit

The CloudSim toolkit is a Java-based Cloud simulator developed by Calheiros *et al.* (2011) as a robust tool for performing repeatable experiments to test proposed algorithms and research works relating to CC. It is built to model the IAAS deployment model of CC. The CloudSim architectural framework is as depicted in Figure 14.

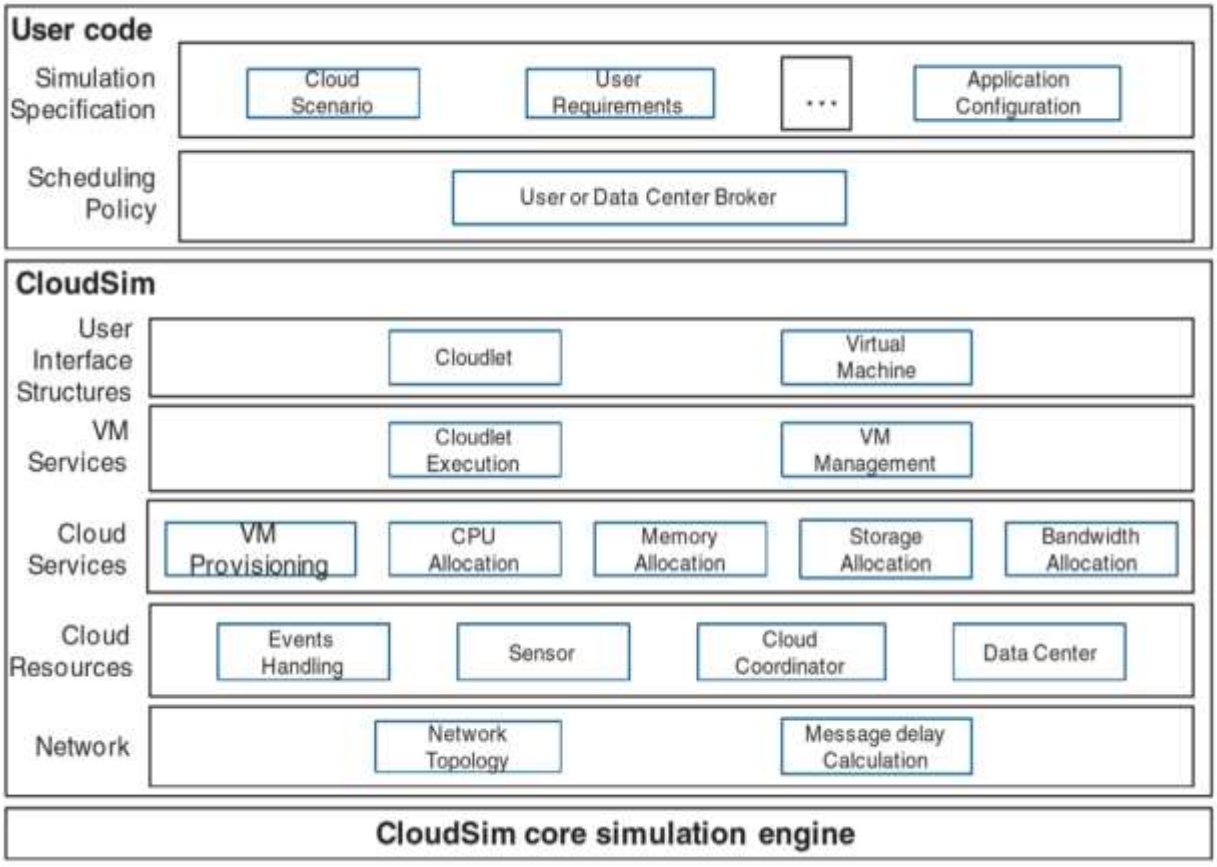


Figure 14: CloudSim architecture (Calheiros *et al*., 2011)

The components of the CloudSim framework are discussed below:

The top layer is the User Code layer and is where user interaction with the system occurs. At this layer, users submit workloads to the broker, which in turn allocates these workloads to host PMs based on specified scheduling policy. The number of PMs required and their respective configurations as desired by the user are also configured at this layer.

The next layer is the CloudSim layer and it is where the actual Cloud is modelled. At its lowest level is the network, which is an abstraction of actual network components. This layer is abstracted as CloudSim does not actually include network entities but rather simulates delays experienced while sending messages between entities.

Above this level is the Cloud Resources and it is made up of the DC. PMs are stored within the DC and have the following specifications: processing capacity in Millions of Instructions Per Second (MIPS), number of CPU cores (processing elements), memory, storage capacities and allocation policies. The sensor is responsible for monitoring and dynamically collecting relevant information about performance of various entities. All collected information are forwarded to the Cloud Coordinator for the purpose of improving resource utilization and load balancing.

Above this level is the Cloud Services layer, which handles allocation of workload requirements onto the Cloud Resources. It includes the VM provisioning, CPU, Memory, Storage and allocation policies. The process of allocating VMs to corresponding PMs is done by the VM provisioning module. Though CloudSim supports two types of VM provisioning which are allocation of VMs to PMs and allocation of applications to VMs; in this work however, only the former is considered. This allocation of VMs to PMs is done using the FCFS policy by default. On top of this level is the VM services, which handles managing, monitoring and performance isolation between VMs. Above this is the User Interface level and it handles accepted user workloads called cloudlets. Below the CloudSim layer is the actual CloudSim simulation engine (Calheiros *et al.*, 2009).

3.9.3 Experiment Setup and Data

In this study, a DC similar to that used in (Beloglazov and Buyya, 2012 and Hieu *et al.*, 2017) was used and consisted of at most 1200 heterogeneous PMs. These PMs were of two categories, the first ran on Intel Xeon 3040 dual core processors clocked at 1.86 GHz, while the second

ran on Intel Xeon 3075 dual core processors clocked at 2.6GHz. It is assumed that PMs in the second category can run more workloads per unit time than those in the first category. This is shown in Table 4. The power consumption models of these PMs are based on SpecPower (2010) real servers benchmarked data. Three different datasets were used in this study - PlanetLab day 3 dataset with 1,078 jobs (Park and Pai, 2006); Google Test Cluster cell with 168 jobs; and Google Clusters data with 1,600 jobs (Wilkes and Reiss, 2011). Though these datasets were reportedly logged for CPU intensive workloads; however, for compatibility with VMCUP-M, a fixed minimal memory value of 0.001 was used. These datasets are made up of independent workloads, which run as a whole on their allocated PM. This implies that a workload cannot simultaneously run on multiple PMs.

Table 4: Specifications of the PMs used for simulation

Category	**Make**	**CPU**	**Cores**	**Memory**
1	HP ProLiant ML110 G4	1,860 MHz	Intel Xeon 3040, 2 cores	4GB
2	HP ProLiant ML110 G5	2,600 MHz	Intel Xeon 3075, 2 cores	4GB

Table 5: Summary of Datasets

Source	**Period**	**Number of VM logged**
PlanetLab	Day 3	1078
Google Test Cluster Data	7 hours	168
Google Dataset	30 days	1600

3.9.4 Implementation and Coding

Implementation was done using CloudSim and Eclipse IDE. Code listings and snapshots of some results are shown in Appendix I and II respectively.

CHAPTER FOUR
DISCUSSION OF RESULTS

4.1 Introduction

For comparison purposes with the other approaches which categorized all user workloads into a single class, this study focuses primarily on the Gold class of workloads. However, for completeness purposes, results obtained for QoS adherence of the Silver and Bronze classes are also reported, albeit in a concise manner.

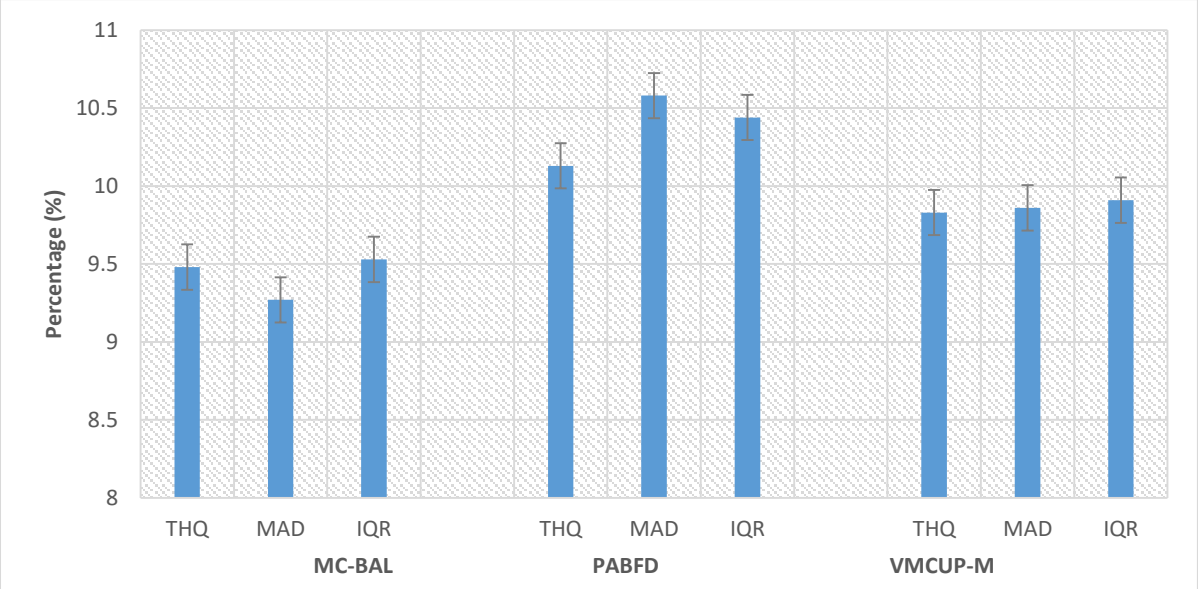
4.2 Tests using PlanetLab Datasets

The results obtained using PlanetLab dataset are detailed in the following subsections:

4.2.1 Adherence to End-to-end Pre-set QoS Constraints (PlanetLab Dataset)

4.2.1.1 Average SLA Violation (PlanetLab Dataset)

In measuring the degree of adherence to QoS constraint, the SLA violation metric discussed in section 3.8.2 was used. MC-BAL was tested against PABFD and VMCUP-M and the result obtained is represented in Figure 14.



THQ: Static Threshold. **MAD:** Median Absolute Deviation. **IQR**: Inter Quartile Range. **MC-BAL**: Multi-Class Load Balancing. **PABFD**: Power-Aware Best Fit Descending.**VMCUP-M**: Virtual Machine Consolidation with Multiple Usage Prediction

Figure 15: Average number of SLA Violation for all PlanetLab Dataset

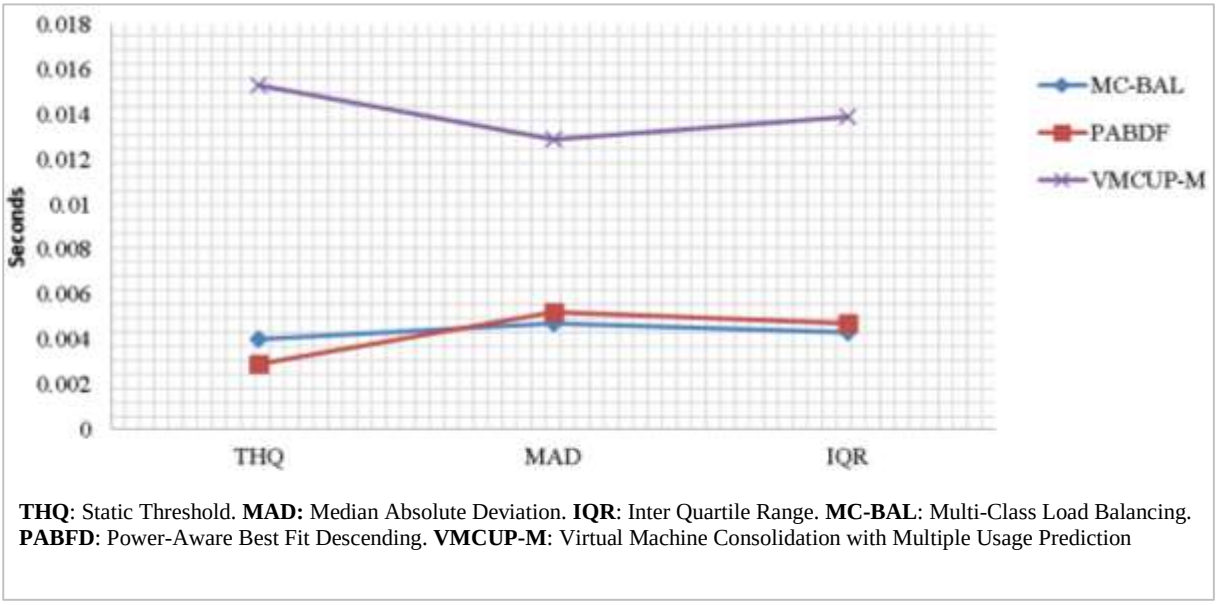
Table 6: Average SLA Violation of each workload class in MC-BAL (PlanetLab Dataset)

METRIC	THQ	MAD	IQR
Average SLA violation for Gold Class (%)	9.35	9.2	9.36
Average SLA violation for Silver Class (%)	10.01	9.24	9.55
Average SLA violation for Bronze Class (%)	11.33	10.54	11.33

In Figure 15, MC-BAL resulted in the lowest average SLA violation of all three approaches and across all three threshold schemes tested. Using the static threshold (THQ), MC-BAL (9.48 %) was better than both PABFD (10.13 %) and VMCUP-M (9.83 %). The same trend was observed for the dynamic thresholds; with Median Absolute Deviation (MAD), MC-BAL (9.27 %) was better than PABFD (10.58 %) and VMCUP-M (9.86 %); while for Inter-Quartile Range (IQR), MC-BAL (9.53 %) resulted in a lower SLA violation than both PABFD (10.44 %) and VMCUP-M (9.91 %).

In Table 6, a comparison of the SLA violation experienced by workloads in each of the three MC-BAL queues is shown for the PlanetLab dataset. As expected the Gold class workloads experienced the least SLA violation, across all threshold schemes. This was followed by the Silver and then the Bronze class workloads. Across all classes, the dynamic threshold schemes (MAD and IQR) resulted in better SLA adherence versus the static scheme (THQ).

4.2.1.2 Average Workload Delay (PlanetLab Dataset)



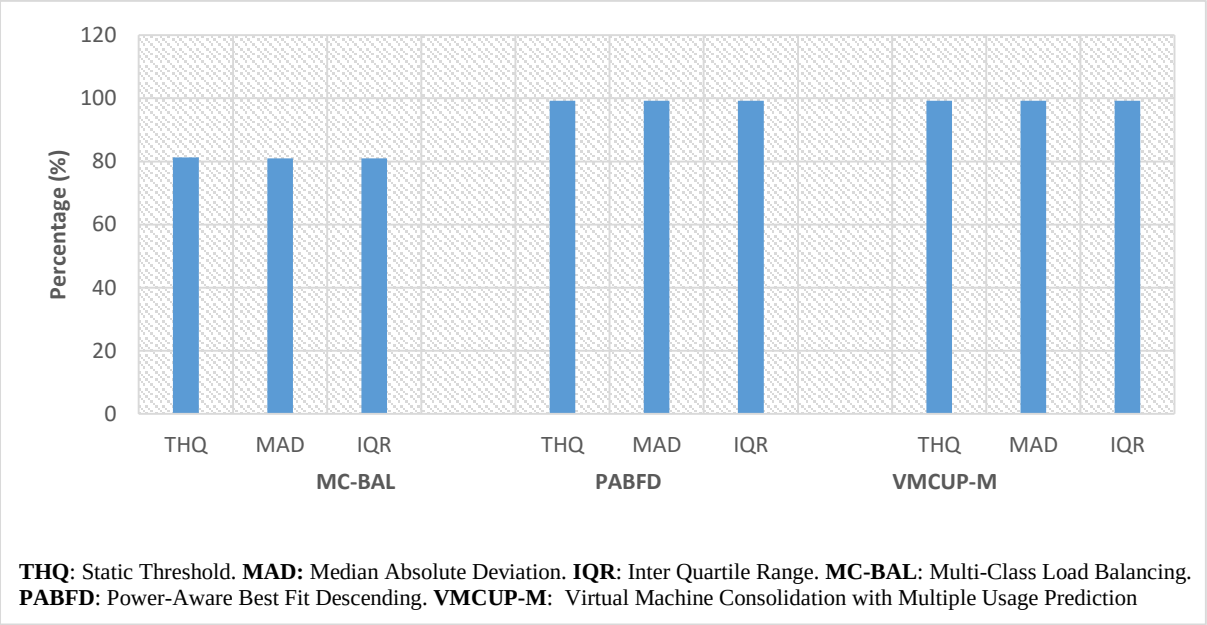
THQ: Static Threshold. **MAD:** Median Absolute Deviation. **IQR**: Inter Quartile Range. **MC-BAL**: Multi-Class Load Balancing. **PABFD**: Power-Aware Best Fit Descending. **VMCUP-M**: Virtual Machine Consolidation with Multiple Usage Prediction

Figure 16: Comparison of Average Workload Delay using PlanetLab Dataset

For static threshold (THQ), Figure 16 shows that PABFD with an average delay of 0.0029s performed better than both MC-BAL (0.0040s) and VMCUP-M (0.0153s). With respect to the dynamic thresholds, PABFD recorded 0.0052s for MAD and 0.0047s for IQR respectively as against MC-BAL's 0.0047s for MAD and 0.0043s for IQR and VMCUP-M's 0.0129s for MAD and 0.0139s IQR.

PABFD was reportedly faster than both other approaches using static threshold, because it only checks the status of potential PMs for power growth prior to allocation. VMCUP-M and MC-BAL on the other hand performed an additional linear regression based CPU utilization test prior to workload allocations. This process slowed them down with respect to PABFD. However, the 2DHIS used by MC-BAL to search for suitable PMs sped it up and accounted for the significant reductions in delay time when compared to VMCUP-M.

4.2.2 Efficient Resource Utilization (PlanetLab Dataset)



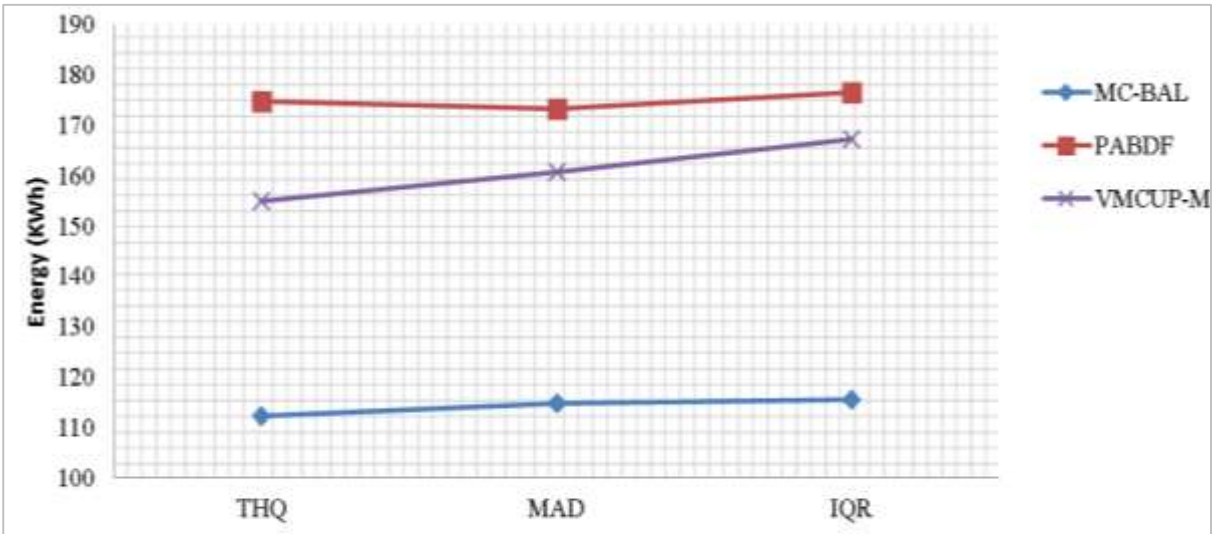
THQ: Static Threshold. **MAD:** Median Absolute Deviation. **IQR**: Inter Quartile Range. **MC-BAL**: Multi-Class Load Balancing. **PABFD**: Power-Aware Best Fit Descending. **VMCUP-M**: Virtual Machine Consolidation with Multiple Usage Prediction

Figure 17: Comparison of Resource Utilization Levels using PlanetLab Dataset

Figure 17 shows that both PABDF and VMCUP-M used 99% of all the PMs in the DC to provide service to all submitted workloads, while MC-BAL used 81 % to achieve same. This implies that MC-BAL managed PM utilization better than both PABFD and VMCUP-M.

4.2.3 Conservation of Energy (PlanetLab Dataset)

4.2.3.1 Total Energy Consumption (PlanetLab Dataset)

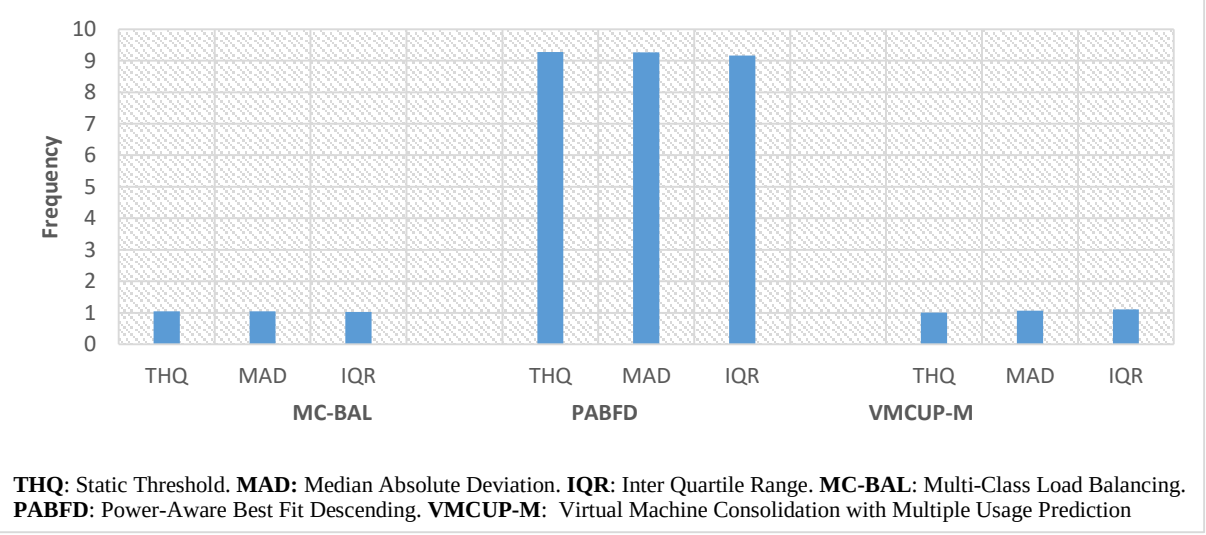


THQ: Static Threshold. **MAD:** Median Absolute Deviation. **IQR**: Inter Quartile Range. **MC-BAL**: Multi-Class Load Balancing. **PABFD**: Power-Aware Best Fit Descending. **VMCUP-M**: Virtual Machine Consolidation with Multiple Usage Prediction

Figure 18: Comparison of Total Energy Consumption for PlanetLab Dataset

In Figure 18, MC-BAL was benchmarked with the other two approaches and performed better than both with respect to conservation of energy. With the static threshold, MC-BAL recorded 112.2 KWh as against 174.9 KWh and 155 KWh for PABDF and VMCUP-M. With respect to MAD, MC-BAL at 114.8 KWh used less energy as compared to PABFD (173.1 KWh) and VMCUP-M (160.6 KWh). Finally, when IQR was used, MC-BAL (115.6 KWh) was better than the other approaches at 176.4 KWh and 167.3 KWh, respectively.

4.2.3.2 Host Power State Change PSC (PlanetLab Dataset)



THQ: Static Threshold. **MAD:** Median Absolute Deviation. **IQR**: Inter Quartile Range. **MC-BAL**: Multi-Class Load Balancing. **PABFD**: Power-Aware Best Fit Descending. **VMCUP-M**: Virtual Machine Consolidation with Multiple Usage Prediction

Figure 19: Comparison of Power State Changes using PlanetLab Dataset

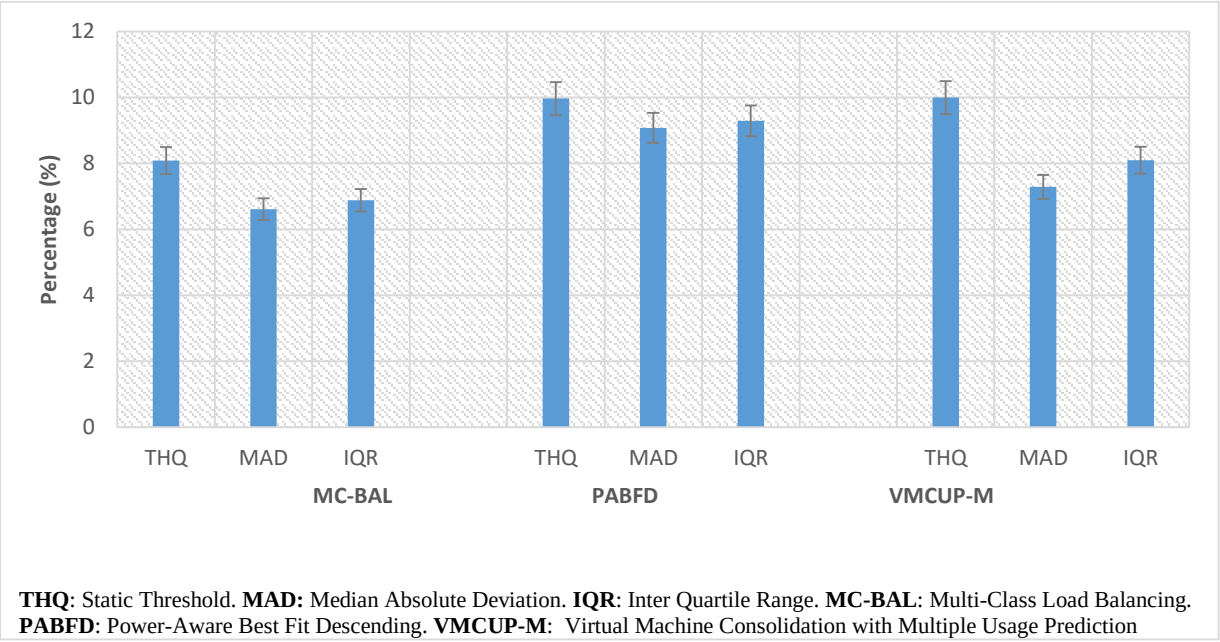
Host PSC is the switching on and off (vice versa) of a PM with respect to load balancing. Results depicted in Figure 19, showed that PMs experienced equal number of average PSC with both MC-BAL and VMCUP-M. PABFD, however, gave the worst result of the three techniques, with a significantly higher number of PSCs per PMs.

4.3 Tests using Google Test Cluster Dataset (GTC)

The GTC dataset is a log of workloads submitted over a period of seven hours to a single cell cluster in one of Google's DC. User workloads are independent and run as a single entity on a PM. Results obtained using GTC dataset are discussed in the following subsections:

4.3.1 Adherence to End-to-end Pre-set QoS Constraints (GTC)

4.3.1.1 Average SLA Violation (GTC Dataset)



THQ: Static Threshold. **MAD:** Median Absolute Deviation. **IQR**: Inter Quartile Range. **MC-BAL**: Multi-Class Load Balancing. **PABFD**: Power-Aware Best Fit Descending. **VMCUP-M**: Virtual Machine Consolidation with Multiple Usage Prediction

Figure 20: Average number of SLA Violation for GTC dataset

Table 7: Average SLA Violation of each workload class in MC-BAL (GTC Dataset)

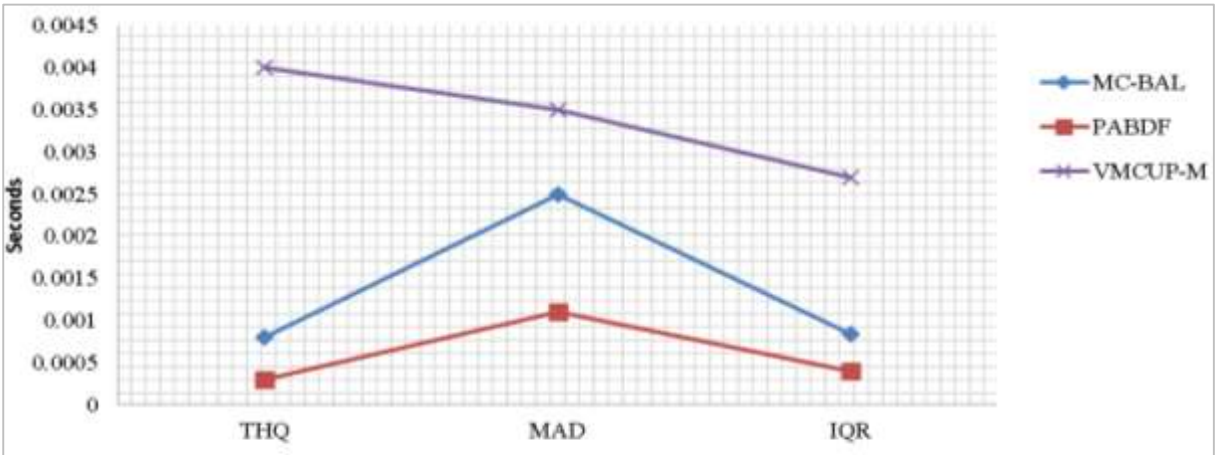
METRIC	THQ	MAD	IQR
Average SLA violation for Gold Class (%)	8.09	6.61	6.89
Average SLA violation for Silver Class (%)	12.35	6.99	7.12
Average SLA violation for Bronze Class (%)	10.89	8.59	9.21

Figure 20 shows that for static threshold (THQ), workloads experienced the least SLA violation with MC-BAL at 8.09 % as against 9.97 % and 10 % with PABFD and VMCUP-M respectively. When dynamic thresholds were used; MC-BAL was better than the other

approaches with 6.61 % for MAD and 6.89 % for IQR, as against PABFD's 9.08 % (MAD) and 9.29 % (IQR) and VMCUP-M's 7.29 % (MAD) and 8.1 % (IQR). MC-BAL therefore resulted in the lowest average SLA violation of all three approaches and across all three threshold schemes tested.

Table 7 shows a comparison of SLA violations experienced by workloads enqueued onto the three classes when using the GTC dataset. Similar to the result obtained with the PlanetLab dataset, of the three classes, the Gold class workloads experience the least SLA violation across all threshold schemes. Similarly, the dynamic threshold schemes particularly MAD resulted in the least SLA violation versus the other threshold schemes.

4.3.1.2 Average Workload Delay (GTC Dataset)

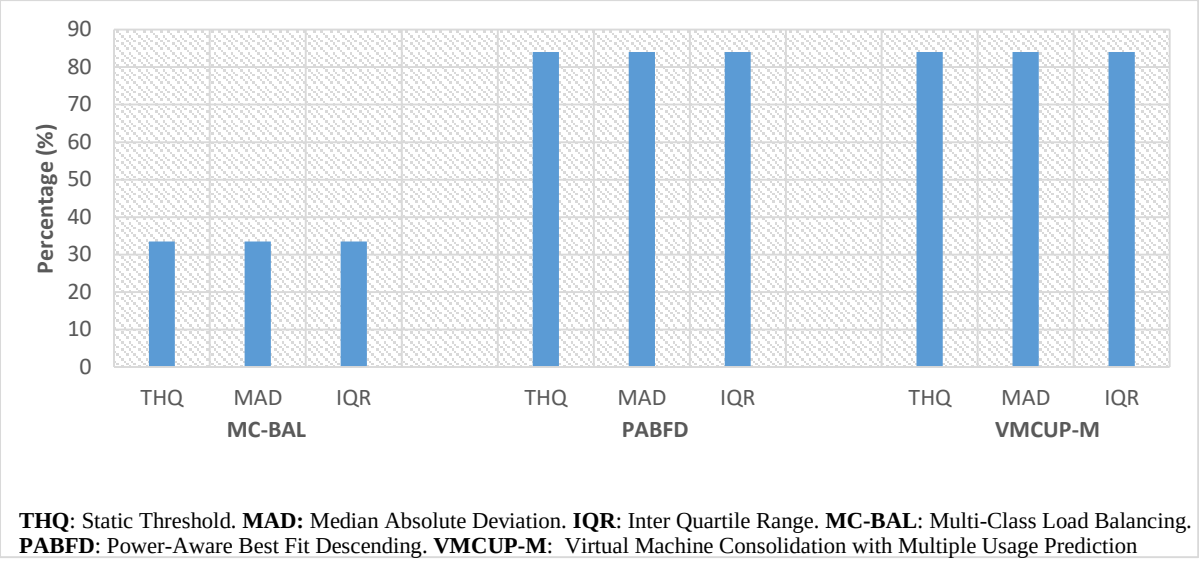


THQ: Static Threshold. **MAD:** Median Absolute Deviation. **IQR**: Inter Quartile Range. **MC-BAL**: Multi-Class Load Balancing. **PABFD**: Power-Aware Best Fit Descending. **VMCUP-M**: Virtual Machine Consolidation with Multiple Usage Prediction

Figure 21: Comparison of Average Workload Delay using GTC dataset

For static threshold (THQ), Figure 21 shows that PABFD with an average delay of 0.0003s was faster than both MC-BAL (0.0008s) and VMCUP-M (0.004s). The same trend was repeated for both MAD and IQR, with PABFD recording 0.0011s for MAD and 0.0004s for IQR respectively as against MC-BAL's 0.0025s for MAD and 0.00084s for IQR. VMCUP-M on the other hand was the slowest at 0.0035s for MAD and 0.0027s for IQR respectively. Once again, the 2DHIS used by MC-BAL accounts for the decrease in workload allocation delay versus VMCUP-M.

4.3.2 Efficient Resource Utilization (GTC Dataset)



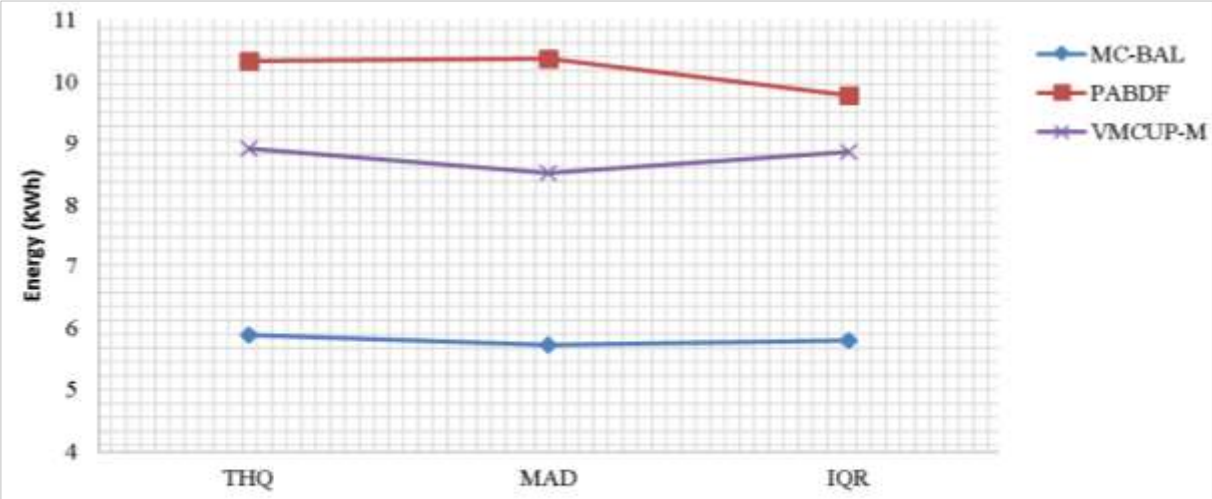
THQ: Static Threshold. **MAD:** Median Absolute Deviation. **IQR**: Inter Quartile Range. **MC-BAL**: Multi-Class Load Balancing. **PABFD**: Power-Aware Best Fit Descending. **VMCUP-M**: Virtual Machine Consolidation with Multiple Usage Prediction

Figure 22: Comparison of Resource Utilization Levels using GTC dataset

Figure 22 shows that MC-BAL uses only 33 % of all PMs in the DC to provide services to all user workloads, as against 84 % used by both PABDF and VMCUP-M. This represents about 40 % improvement in resource utilization for MC-BAL as compared to the two other approaches.

4.3.3 Conservation of Energy (GTC Dataset)

4.3.3.1 Total Energy Consumption (GTC Dataset)

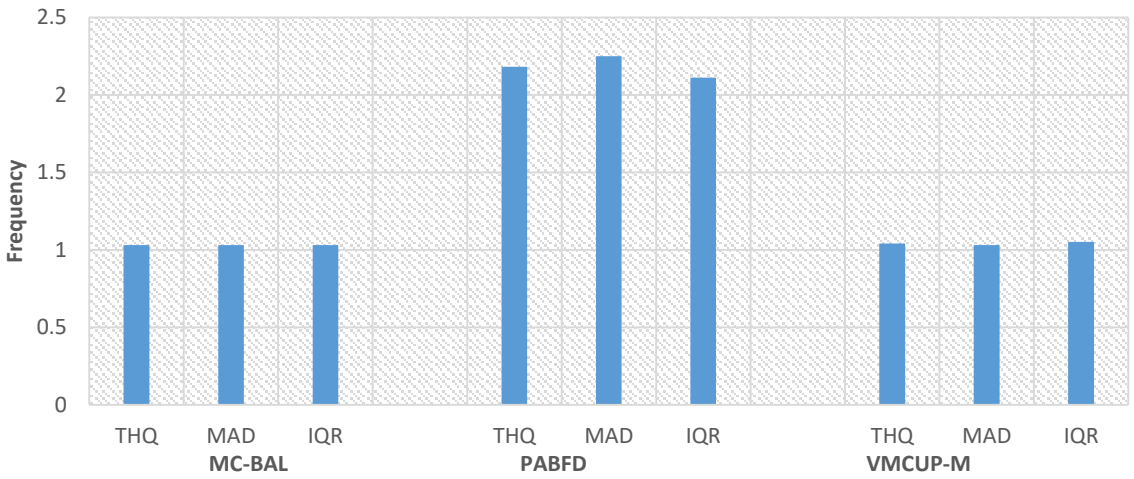


THQ: Static Threshold. **MAD:** Median Absolute Deviation. **IQR**: Inter Quartile Range. **MC-BAL**: Multi-Class Load Balancing. **PABFD**: Power-Aware Best Fit Descending. **VMCUP-M**: Virtual Machine Consolidation with Multiple Usage Prediction

Figure 23: Comparison of Total Energy Consumption for GTC dataset

In Figure 23, MC-BAL is compared with PABFD and VMCUP-M; and as observed with the PlanetLab dataset, MC-BAL conserved energy better than the other approaches. With the static threshold, MC-BAL recorded 5.89 KWh as against 10.33 KWh and 8.91 KWh for PABDF and VMCUP-M respectively. Using MAD as threshold, MC-BAL utilized 5.73 KWh of energy which is less than that used by PABFD (10.37 KWh) and VMCUP-M (8.52 KWh). Finally, when IQR was used as threshold, MC-BAL consumed only 5.8 KWh of energy which is better than PABFD and VMCUP-M which both consume 9.77 KWh and 8.86 KWh respectively.

4.3.3.2 Host Power State Change PSC (GTC Dataset)



THQ: Static Threshold. **MAD:** Median Absolute Deviation. **IQR**: Inter Quartile Range. **MC-BAL**: Multi-Class Load Balancing. **PABFD**: Power-Aware Best Fit Descending. **VMCUP-M**: Virtual Machine Consolidation with Multiple Usage Prediction

Figure 24: Comparison of Power State Changes using GTC dataset

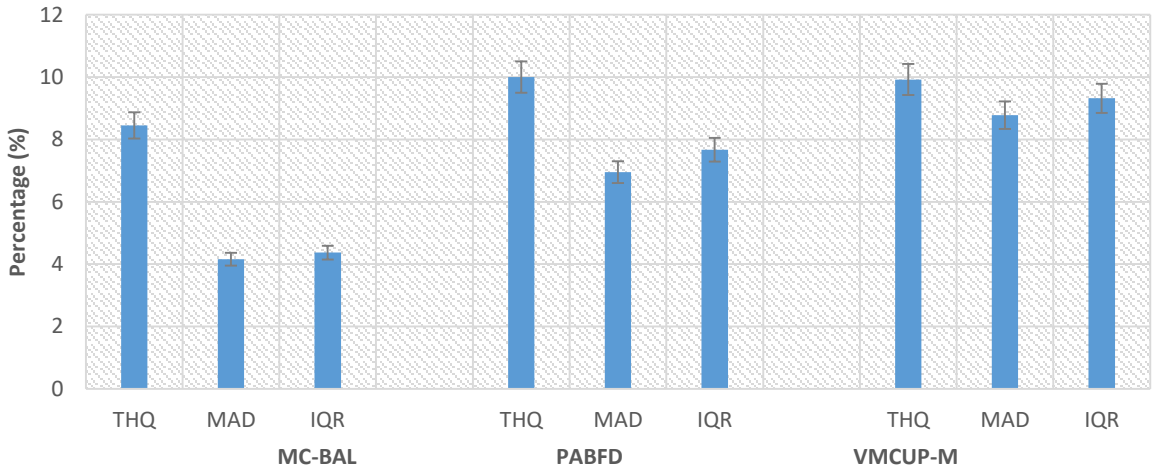
In terms of average PM PSCs, Figure 24 shows that MC-BAL matched VMCUP-M with an average number of 1.03 PSCs across all thresholds (THQ, MAD and IQR) and both were better than PABFD with 2.18 (THQ), 2.3 (MAD) and 2.1 (IQR).

4.4 Tests using Google Cluster Dataset (GCD)

The GCD is a log of workloads submitted to a cluster within a Google DC, consisting of 12,500 PMs and logged over a period of 30 days. The results obtained using this dataset are detailed below:

4.4.1 Adherence to End-to-end Pre-set QoS Constraints (GCD)

4.4.1.1 Average SLA Violation (GCD Dataset)



THQ: Static Threshold. **MAD:** Median Absolute Deviation. **IQR**: Inter Quartile Range. **MC-BAL**: Multi-Class Load Balancing. **PABFD**: Power-Aware Best Fit Descending. **VMCUP-M**: Virtual Machine Consolidation with Multiple Usage Prediction

Figure 25: Average number of SLA Violation for GCD dataset

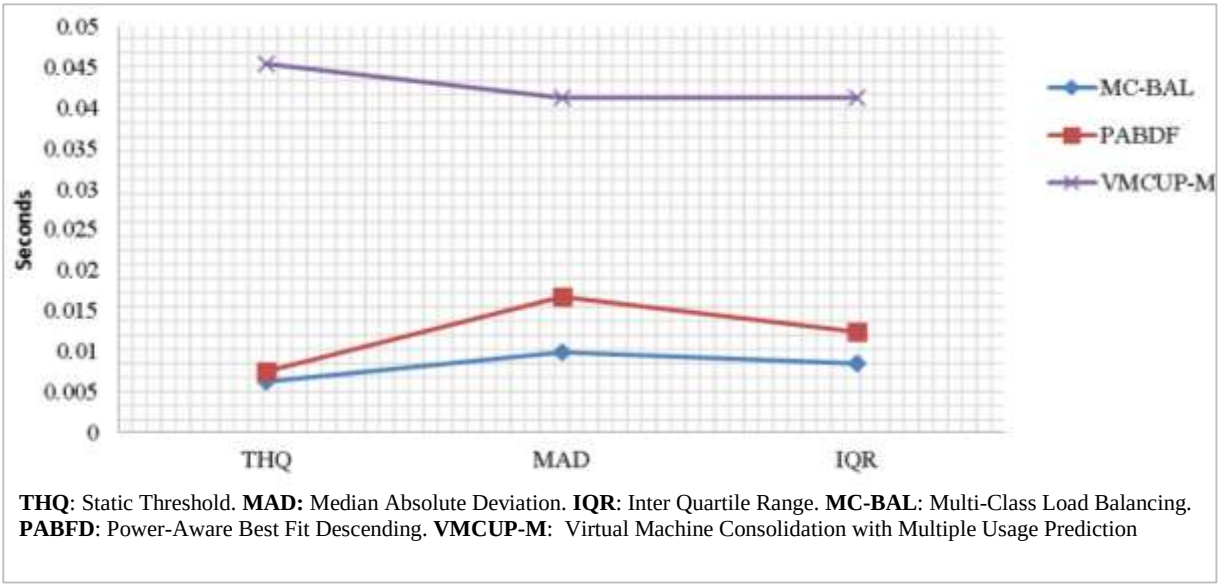
Table 8: Average SLA Violation of each workload class in MC-BAL (GCD Dataset)

METRIC	THQ	MAD	IQR
Average SLA violation for Gold Class (%)	8.45	4.16	4.37
Average SLA violation for Silver Class (%)	10.06	4.84	6.22
Average SLA violation for Bronze Class (%)	10.83	14.98	17.60

Figure 25 shows that workloads experienced the least SLA violation when MC-BAL was used as compared to the two other approaches. Using the static threshold (THQ), workloads experienced 8.45 % violation with MC-BAL, 10 % with PABFD and 9.92 % with VMCUP-M. Using MAD dynamic threshold, MC-BAL resulted in 4.16 % violation as against the 6.95 % of PABFD and 8.78 % of VMCUP-M. Using IQR, workloads experienced only 4.37 % SLA violation with MC-BAL as against the 7.67 % and 9.32 % of PABFD and VMCUP-M respectively.

In Table 8, a comparison of the average SLA violations for each MC-BAL workload class is shown. As with the other datasets, of the three classes, the Gold class experienced the least SLA violation. Similarly, the MAD dynamic threshold scheme gave the best SLA adherence of all three PM utilization threshold schemes.

4.4.1.2 Average Workload Delay (GCD Dataset)



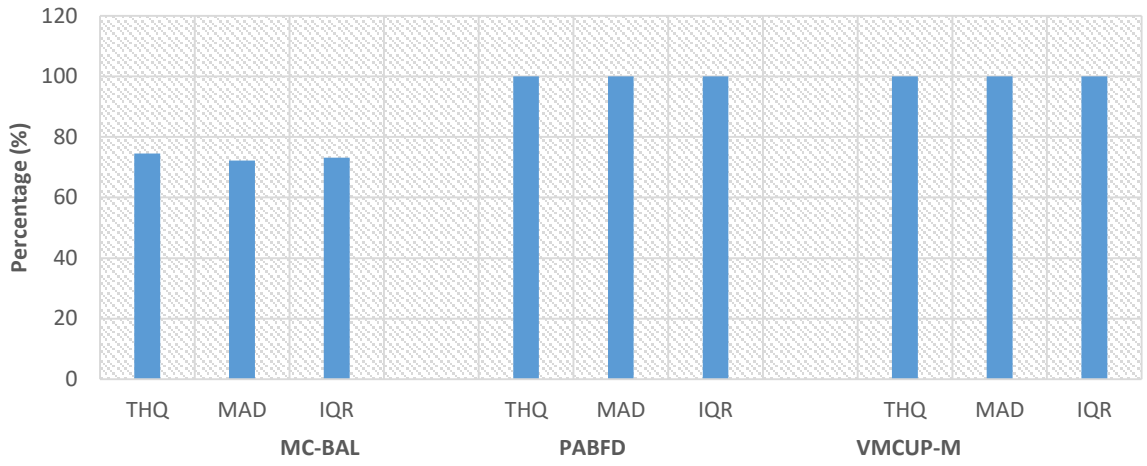
THQ: Static Threshold. **MAD:** Median Absolute Deviation. **IQR**: Inter Quartile Range. **MC-BAL**: Multi-Class Load Balancing. **PABFD**: Power-Aware Best Fit Descending. **VMCUP-M**: Virtual Machine Consolidation with Multiple Usage Prediction

Figure 26: Comparison of Average Workload Delay using GCD dataset

Results of average workload delay using the GCD dataset are depicted in Figure 26. For the static threshold (THQ), MC-BAL at 0.0063s was faster than both PABFD (0.0075s) and VMCUP-M (0.0454s). The same trend was repeated for MAD and IQR thresholds, with MC-BAL having 0.0099s for MAD and 0.0085s for IQR as against PABDF's 0.0167s for MAD and 0.0124s for IQR. VMCUP-M however resulted in the greatest workload allocation delays with 0.0413s for both MAD and IQR.

The advantage of 2DHIS is vividly shown here, as it resulted in workloads experiencing the least delay with MC-BAL versus the other two approaches. This was because a larger number of PMs (1200) was needed for the GCD dataset as against the 200 and 569 PMs used for GTC and PlanetLab datasets respectively. With the larger number of PMs, the speed of 2DHIS at $O(\log_2 n)$ versus that of linear search (O(n)) used by both PABFD and VMCUP-M became apparent.

4.4.2 Efficient Resource Utilization (GCD Dataset)



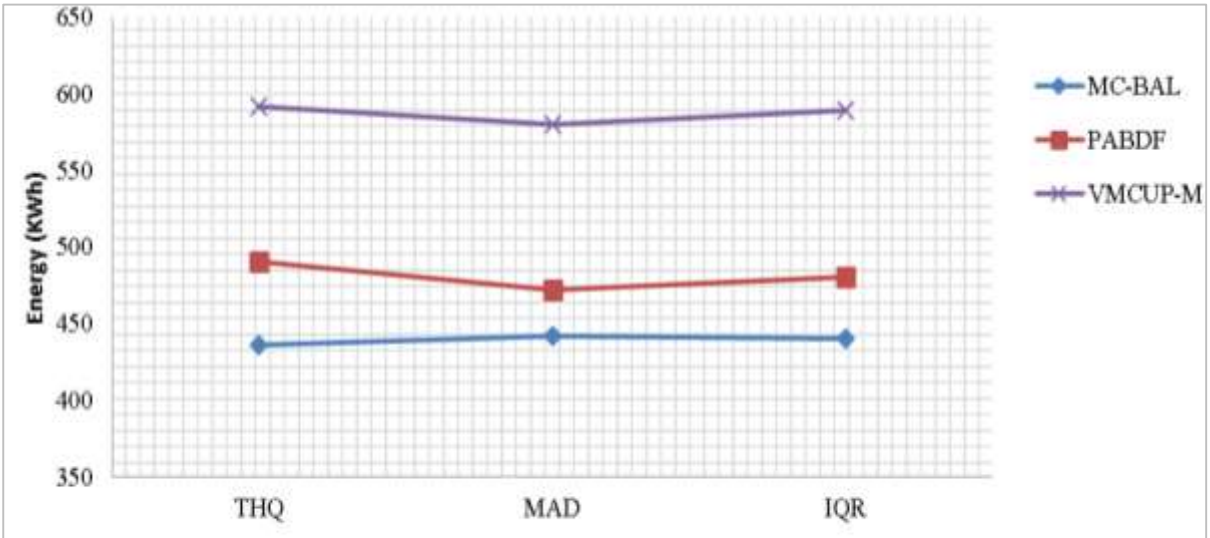
THQ: Static Threshold. **MAD:** Median Absolute Deviation. **IQR**: Inter Quartile Range. **MC-BAL**: Multi-Class Load Balancing. **PABFD**: Power-Aware Best Fit Descending. **VMCUP-M**: Virtual Machine Consolidation with Multiple Usage Prediction

Figure 27: Comparison of Resource Utilization Levels using GCD dataset

Figure 27 shows that MC-BAL used an average of 75 % of all PMs in the DC as against the 100 % used by both PABDF and VMCUP-M. This implied an improvement of 25 % in resource utilization for MC-BAL versus the two other approaches.

4.4.3 Conservation of Energy (GCD Dataset)

4.4.3.1 Total Energy Consumption (GCD Dataset)

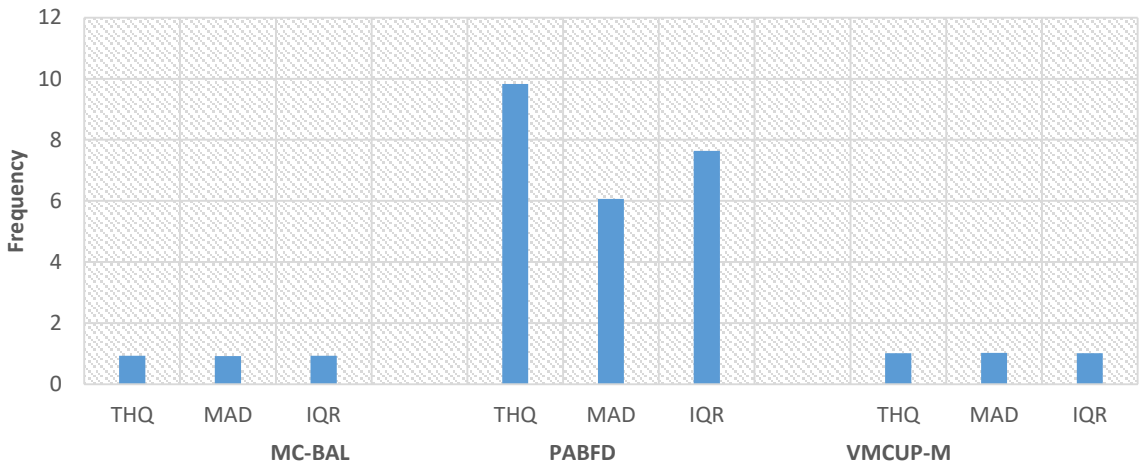


THQ: Static Threshold. **MAD:** Median Absolute Deviation. **IQR**: Inter Quartile Range. **MC-BAL**: Multi-Class Load Balancing. **PABFD**: Power-Aware Best Fit Descending. **VMCUP-M**: Virtual Machine Consolidation with Multiple Usage Prediction

Figure 28: Comparison of Total Energy Consumption for GCD dataset

In Figure 28, MC-BAL is compared with PABFD and VMCUP-M, and as observed with the two other datasets, MC-BAL conserved energy better than the other two approaches. When the static threshold (THQ) was used MC-BAL consumes 436.3 KWh of energy as against 490.4 KWh and 591.3 KWh consumed by PABDF and VMCUP-M respectively. Using the MAD dynamic threshold, MC-BAL consumed 441.8 KWh which was less than those consumed by PABFD (471.6 KWh) and VMCUP-M (579.7 KWh). Finally, when IQR was used as threshold, MC-BAL was better than the others with 440.3 KWh of energy versus the 480.4 KWh and 589.3 KWh of PABFD and VMCUP-M respectively.

4.4.3.2 Host Power State Changes (GCD Dataset)



THQ: Static Threshold. **MAD:** Median Absolute Deviation. **IQR**: Inter Quartile Range. **MC-BAL**: Multi-Class Load Balancing. **PABFD**: Power-Aware Best Fit Descending. **VMCUP-M**: Virtual Machine Consolidation with Multiple Usage Prediction

Figure 29: Comparison of Power State Changes using GCD dataset

In Figure 29, the number of PSCs for all three approaches were compared. Though PABFD showed an improvement in the number of state changes when the MAD dynamic threshold was used (6.06), it was clearly outclassed by both VMCUP-M and MC-BAL. MC-BAL slightly edges out VMCUP-M with an average value of 0.93 across all thresholds versus VMCUP-M's average of 1.01.

4.5 Summary of Results

The results obtained for simulations carried out using the three datasets are discussed in the subsequent subsections and in tables 9 to tables 11.

4.5.1 Using PlanetLab Dataset

Using PlanetLab dataset, the performance of MC-BAL with respect to PABFD and VMCUP-M is summarized as follows:

1. With respect to adherence to QoS, MC-BAL was better than PABFD by an average of 9 % for both static and dynamic thresholds. It was also better than VMCUP-M by an average 4.5 % for both the static and dynamic thresholds.
2. In terms of average delay, PABFD was faster than VMCUP-M by 81 % and 64 % for static and dynamic thresholds respectively. Compared to PABFD, MC-BAL was slower by 26 % with the static threshold but faster by an average of 5 % with the dynamic thresholds. This implied that workloads experienced the least allocation delays when MC-BAL was combined with the dynamic thresholds schemes.
3. Resource utilization was improved by approximately 19 % when MC-BAL was used compared to when the two other approaches were used.
4. In terms of improving energy consumption, MC-BAL conserved an average of 27 % and 30 % more energy than VMCUP-M and PABFD respectively.
5. For number of PSCs, MC-BAL performed better than PABFD with an 89 % improvement for both static and dynamic thresholds; and was at par with VMCUP-M.

Table 9: Summary of results obtained using PlanetLab datasets

PLANETLAB DATASET									
	MC-BAL			PABFD			VMCUP-M		
	THQ	MAD	IQR	THQ	MAD	IQR	THQ	MAD	IQR
Number of PMs	569	569	569	569	569	569	569	569	569
Number of VMs	1078	1078	1078	1078	1078	1078	1078	1078	1078
Energy Consumption (KWh)	112.16	114.75	115.59	174.85	173.14	176.36	155.02	160.62	167.29
Avg. SLA Violation (%)	9.48	9.27	9.53	10.13	10.58	10.44	9.83	9.86	9.91

No. of PM Shutdowns	539	541	610.5	521	522	515	517	515	511
Unused PMs	191	193	204	2	2	2	2	2	2
Avg. No of Migrations / VM	5.7	5.07	6.38	23.27	30.47	22.6	1.78	1.85	1.82
No. of PM PSCs	1.02	1.02	1.03	9.34	9.99	9.66	1.01	1.01	0.99
PM Selection Time (s)	0.0040	0.0047	0.0045	0.0029	0.0052	0.0045	0.0153	0.0129	0.0139
Resource Utilization (%)	81	81	81	99	99	99	99	99	99

4.5.2 Using Google Test Cluster Dataset

Results obtained from simulations carried out using dataset from GTC are summarized as follows:

1. In terms of adherence to QoS, MC-BAL was 19 % better than both PABFD and VMCUP-M when the static threshold was used. For the dynamic thresholds, when MAD was used, MC-BAL was 27 % and 9 % better than PABFD and VMCUP-M respectively. When IQR was used, MC-BAL was 25 % and 15 % better than PABFD and VMCUP-M respectively.
2. For average workload delay, MC-BAL was slower than PABFD and this is as a result of the small number of PMs (200) used by the GTC dataset. Compared with VMCUP-M however, MC-BAL was faster for both static and dynamic thresholds.
3. For resource utilization, compared to PABFD and VMCUP-M, MC-BAL used 40 % less resources to accomplish the same amount of work.
4. In terms of improving energy consumption, when MC-BAL was used, 42 % less energy was consumed in the DC versus PABFD across both thresholds schemes. Similarly, versus VMCUP-M, MC-BAL consumed an average of 32 % less energy across both thresholds schemes.

5. For number of PSCs, MC-BAL switched PMs states half the number of times PABFD did, thus representing a 50 % improvement across all thresholds schemes. With respect to VMCUP-M, MC-BAL was at par.

Table 10: Summary of results obtained using GTC datasets

GOOGLE TEST CLUSTER DATASET									
	MC-BAL			PABFD			VMCUP-M		
	THQ	MAD	IQR	THQ	MAD	IQR	THQ	MAD	IQR
Number of PMs	200	200	200	200	200	200	200	200	200
Number of VMs	168	168	168	168	168	168	168	168	168
Energy Consumption (KWh)	5.89	5.73	5.8	10.33	10.37	9.77	8.91	8.52	8.86
Avg. SLA Violation (%)	8.09	6.61	6.885	9.97	9.08	9.29	10	7.29	8.1
No. of PM Shutdowns	191	193	192	181	169	168	175	181	180
Unused PMs	133	133	133	32	32	32	32	32	32
Avg. No of Migrations / VM	3.08	2.32	3.12	10.45	13.19	11.60	2.98	3.39	3.58
No. of PM PSCs	1.03	1.03	1.03	2.18	2.25	2.11	1.03	1.03	1.04
PM Selection Time (s)	0.0008	0.0025	0.0008	0.0004	0.0011	0.0004	0.0040	0.0035	0.0027
Resource Utilization (%)	33	33	33	84	84	84	84	84	84

4.5.3 Using Google Cluster Dataset

Results obtained from tests carried out using dataset from GCD are summarized as follows:

1. In terms of adherence to QoS, workloads experienced 15 % less SLA violation with MC-BAL versus PABFD and VMCUP-M when the static threshold was used. For the dynamic thresholds, with MAD, MC-BAL was 40 % and 52 % better than PABFD and VMCUP-M respectively. When IQR was used, MC-BAL was 43 % and 53 % better than PABFD and VMCUP-M respectively.
2. For average workload delay, MC-BAL was 19 % and 36 % faster than PABFD for the static and dynamic threshold schemes respectively. Compared with VMCUP-M, MC-BAL was significantly faster at 86 % and 77 % for both static and dynamic thresholds respectively. MC-BAL using its 2DHIS, was able to take advantage of the large number of PMs (1200) in the DC, hence the bolstered speed.
3. For resource utilization, MC-BAL used at least 25 % less resources than both PABFD and VMCUP-M to accomplish the same amount of work.
4. In terms of improving energy consumption, when MC-BAL was used, an energy saving of 11 % and 7 % were recorded for the static and dynamic thresholds respectively versus PABFD. Compared with VMCUP-M, MC-BAL results in an average of 25 % energy saving across all threshold schemes.
5. For number of PSCs, MC-BAL was on the average 87 % better than PABFD across all thresholds schemes. Compared with VMCUP-M, MC-BAL was better by an average of 7 % across all threshold schemes.

Table 11: Summary of results obtained using GCD datasets

GOOGLE CLUSTER DATASET									
	MC-BAL			PABFD			VMCUP-M		
	THQ	MAD	IQR	THQ	MAD	IQR	THQ	MAD	IQR
Number of PMs	1200	1200	1200	1200	1200	1200	1200	1200	1200
Number of VMs	1600	1600	1600	1600	1600	1600	1600	1600	1600
Energy Consumption (KWh)	436.34	441.79	440.32	490.39	471.63	480.36	591.27	579.7	589.28

Avg. SLA Violation (%)	8.45	4.16	4.37	10	6.95	7.67	9.92	8.78	9.32
No. of PM Shutdowns	845	860	914	1062	1086	1079	951	997	989
Unused PMs	306	334	322	0	0	0	0	0	0
Avg. No of Migrations / VM	4.26	3.15	3.64	41.37	38.49	48.14	2.41	2.34	2.35
No. of PM PSCs	0.93	0.92	0.93	9.83	6.06	7.64	1.01	1.02	1.01
PM Selection Time (s)	0.0063	0.0099	0.0085	0.0078	0.0167	0.0124	0.0454	0.0413	0.0414
Resource Utilization (%)	75	75	75	100	100	100	100	100	100

CHAPTER FIVE

SUMMARY, CONCLUSION AND CONTRIBUTIONS TO KNOWLEDGE

5.1 Summary of Findings

Based on results presented and discussed in chapter four, Table 12 summarizes the findings of this study.

Table 12: Summary of Findings

Objectives		**Findings**
1	To design a load balancing scheme that ensures adherence to end-to-end, pre-set Quality of Service (QoS) while providing services to Cloud users.	Multi-Class Balancing Scheme (MC-BAL) was developed to ensure QoS adherence. From findings, it is the only approach that guarantees adherence to QoS both at the allocation and load balancing phases using workload classes; and does so with the least average SLA violation (13 %) when compared to other state of the art schemes.
2	To develop a scheme that efficiently utilizes Cloud resources while providing services to Cloud users.	The findings show that MC-BAL and its combined use of resource usage prediction, class-based virtual machine consolidation and auto-scaling manages resources better. It is able to use at least 19 % less data centre resources to achieve the same amount of work as the other comparative approaches.
3	To improve the overall energy consumption of Cloud Data centres.	The findings, show that when compared to the other approaches, MC-BAL conserves 24 % more energy. This translates to lower carbon emission and cleaner (green) environment; which is one of the major concerns in the world today.

5.2 Conclusions

In this study, two Cloud stakeholders were identified – the Cloud users and the Cloud Service Providers (CSPs); and these stakeholders have varied requirements. To the users, adherence to pre-agreed service level agreements is vital; while to the CSPs, efficient resource utilization and energy conservation are key. These requirements are often contrasting as an attempt to improve one, results in a reduction or violation of the other(s). The aim of this study therefore, was to develop a resource management scheme for Cloud data centres that simultaneously satisfied these stakeholders' requirements. This was achieved using Multi-Class Load Balancing (MC-BAL), a class-based Cloud workload allocation and load balancing scheme.

MC-BAL addressed the user requirement of quality service provisioning, by combining a class-based workload management scheme with a half interval searching technique for allocation of workloads to PMs. This combination improved user satisfaction as MC-BAL was able to guarantee end-to-end adherence to pre-agreed QoS requirements. On the other hand, MC-BAL was also able to satisfy the CSPs' requirements by combining a class-based virtual machine consolidation scheme with resource usage prediction and auto-scaling. This combination enabled MC-BAL ensure efficient utilization of Cloud resources, while providing services to users. Using MC-BAL, CSPs can thus consolidate their resources better, creating room to take on new customers and increase profitability. MC-BAL, using PM sleep states was able to reduce the energy consumption of Cloud data centres. This makes MC-BAL environmentally friendly, as reductions in energy consumption translates to lower carbon emission, consequently addressing one of the global concerns in the world today.

5.3 Significant Contributions to Knowledge

The study made the following contributions to knowledge:

1. The study developed a resource management model for Cloud computing that guarantees end-to-end pre-set quality of service adherence while providing service to users.

2. A class-based resource management scheme for Cloud computing was developed to address three major challenges of adherence to quality of service requirements, efficient resource utilization and energy conservation using the class of users' workloads.

3. A workload allocation scheme, called Double-Depth Half Interval Search (2DHIS) was developed. The scheme, which is an adaptation of the binary search, is faster than the linear search approach used by other works. This makes 2DHIS particularly advantageous in data centres with large number of physical machines.

5.4 Further Work

A Class based workload allocation and load balancing scheme for Cloud Computing was presented in this work. However, classification of user workloads was done using only the user specified burst time. It is has been argued that users often over-estimate their workload requirements and this might lead to over provisioning and resource wastage. In the future an alternative workload classification scheme can be considered.

This work paid more emphasis on the Gold (premium) users and thus favoured them to the detriment of the other classes of users. Future research work could be in the direction of improving MC-BAL in view of improving fairness to the other workload classes.

Furthermore, in this study, CPU utilization was the sole criterion used in determining the work load status of PMs. Other resources such as Memory, Bandwidth and Storage could be considered in future works.

REFERENCES

Ajayi O., and Oladeji F. (2015). An Overview of Resource Management Challenges in Cloud Computing. *Book of Proc. 10th Unilag Research Conf. and Fair*, 2(3), (pp. 554-560).

Ajayi O., Oladeji F. and Uwadia C. (2015). Analysis of Two-Phased Approaches to Load Balancing in Cloud Computing. *Journal of Computer Science and its Applications*, 22(2), pp. 123-131

Ajayi O., Oladeji F. and Uwadia C. (2016). Multi-Class Load Balancing for QoS and Energy Conservation in Cloud Computing. *West African Journal of Industrial and Academic Research*, 17, pp. 28-36.

Ajayi O., Oladeji F. and Uwadia C. (2017). An Energy-Aware Class-Based Load Balancing Model for Resource Management in Cloud Computing. *FUW Trends in Science and Technology Journal*, 2(1), pp. 295-300

Al-maamari A. and Omara F. (2015). Task Scheduling Using PSO Algorithm in Cloud Computing Environments, *Intl. J. of Grid Distribution Computing*. 8(5), pp. 245-256.

Amalarethinam D. and MalaiSelvi F. (2012). A Minimum Makespan Grid Workflow Scheduling Algorithm. In *Computer Communication and Informatics*. *Intl. Conf. on* (pp. 1-6). IEEE.

Amarante S., Cardoso A., Roberto F. and Celestino J. (2013). Using the Multiple Knapsack Problem to Model the Problem of Virtual Machine Allocation in Cloud Computing. In *16th Computational Science and Engineering*. *Intl. Conf. on* (pp. 476 – 483). IEEE.

Anagnostopoulou V., Biswas S., Saadeldeen H., Savage A., Bianchini R., Yang T., Franklin D. and Chong F. (2012). Barely alive memory servers: Keeping data active in a low-power state. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 8(4), pp. 31:1–31:20.

Ani V., Nzeako A. and Obianuko J. (2012). Energy Optimization at Datacenters in Two Different Locations of Nigeria. *Intl. Journal of Energy Engineering*, 2(4), pp. 151-164.

Baig R., Khan W., Haq I. and Khan I. (2017). Agent-based SLA negotiation protocol for cloud computing. *Intl. Conf. of Cloud Computing Research and Innovation, CloudAsia2017*, p.5.

Banerjee S., Adhikari M., Kar S. and Biswas U. (2015). Development and Analysis of a New Cloudlet allocation Strategy for QoS Improvement in Cloud. *Arabian Journal for Science and Engineering*, 40(5), pp. 1409-1425.

Batista B., Estrella J., Ferreira C., Filho D., Nakamura V., Reiff-Marganiec S., Santana M. and Santanat R. (2015). Performance Evaluation of Resource Management in Cloud Computing Environments. *PLoS one*, 10(11), p.8

Beloglazov A. (2013). Energy Efficient Management of Virtual Machines in Data Centers for Cloud Computing. Ph.D. Thesis, University of Melbourne, Australia.

Beloglazov A. and Buyya R. (2012). Optimal Online Deterministic Algorithms and Adaptive Heuristics for Energy and Performance Efficient Dynamic Consolidation of Virtual Machines in Cloud Data Centers. *Concurrency and Computation: Practice and Experience*, 24(13), pp. 1397-1420.

Beloglazov A., Buyya R., Lee Y. and Zomaya A. (2011). A Taxonomy and Survey of Energy-Efficient Data Centers and Cloud Computing Systems. *Advances in Computers*, 82(2), pp. 47-111.

Bermejo B., Guerrero C., Lera I. and Juiz, C. (2016). Cloud Resource Management to Improve Energy Efficiency Based on Local Node Optimization. *Procedia Computer Science*, 83, pp. 878-885.

Bigelow J. (2012). Managing Physical and Virtual Machine Sprawl, When Good Data Centers Go Bad. *Techtarget Inc, Whitepaper*, p.6. Retrieved from http://docs.media.bitpipe.com

Buyya R. and Murshed M. (2002). Gridsim: A Toolkit For The Modeling And Simulation Of Distributed Resource Management And Scheduling For Grid Computing. *Concurrency and computation: practice and experience*, *14*(13-15), pp.1175-1220.

Buyya R., Garg S. and Calheiros N. (2011). SLA-Oriented Resource Provisioning for Cloud Computing: Challenges, Architecture and Solutions. *In Cloud and Service Computing (CSC).2011 Intl. Conf. on* (pp. 1-10). IEEE

Buyya R., Ranjan R. and Calheiros, R. (2009). Modeling and Simulation of Scalable Cloud Computing Environments and the Cloudsim Toolkit: Challenges and Opportunities. In *High Performance Computing and Simulation 2009.* HPCS'09. *Intl. Conf. on* (pp. 1-11). IEEE

Calheiros R., Ranjan R., Rose C. and Buyya R. (2009). CloudSim: A Novel Framework for Modeling and Simulation of Cloud Computing Infrastructures and Services. To be published. p. 9. Retrieved from https://arxiv.org. Accessed 1st April, 2015.

Calheiros R., Ranjan R., Beloglazov A., Rose C. and Buyya R. (2011). CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms. *Software: Practice and Experience*, 41(1), pp.23–50.

Candler B. (2014). Virtual Machine Migration, Network Startup Resource Center. [Online] Available at www.nsrc.org/workshops. Accessed 5th December, 2016.

Cloud Security Alliannce CSA (2011). Security Guidance for Critical Areas of Focus in Cloud Computing v4.0, p.152. Retrieved from https://downloads.cloudsecurityalliance.org /assets/research/security-guidance/security-guidance-v4-FINAL.pdf. Accessed 1/4/15

Choudhary M. (2015). SLA Aware Load Balancing Algorithm Using Join-Idle-Queue for Virtual Machine in Cloud Computing. *Intl. Journal of Innovative Research in Computer and Communication Engineering*. 3(9), pp. 9005-9011

Cisco Inc. (2016) Cisco Global Cloud Index: Forecast and Methodology, 2015-2020. White paper. p. 29. Retrieved from: www.cisco.com/c/dam/en/us/.../global-cloud-index.../white-paper-c11-738085.pdf. Accessed 11th May, 2017.

Daharwal P. and Sharma V. (2017). Energy Efficient Cloud Computing Vm Placement Based On Genetic Algorithm. *Intl. Journal of Computer Trends and Technology*. 44(1), pp. 15-23.

Dhinesh B. and Krishna P. (2014). Honey bee behavior inspired load balancing of tasks in Cloud computing environment. Applied Soft Computing, 13(5), pp. 2292-2303

Dhingra A. and Paul S. (2014). Green cloud: smart resource allocation and optimization using simulated annealing technique. *Indian Journal of Computer Science and Engineering*, 5(2), pp. 40–48

Dumitrescu C. and Foster I. (2005), GangSim: A Simulator for Grid Scheduling Studies. In *Cluster Computing and the Grid, 2005. CCGrid 2005. Intl. Symp.on,* (2, pp. 1151-1158). IEEE.

Effatparvar M. and Garshasbi M. (2014). A genetic algorithm for static load balancing in parallel heterogeneous systems. *Intl. Conf. on Innovation. Management and Technology Res*., 129, pp. 358-364.

Fan X., Weber W. and Barroso L. (2007). Power Provisioning for a Warehouse-sized computer. In *ACM SIGARCH Computer Architecture News*, 35(2), pp. 13-23. ACM.

Fang Y., Wang F. and Ge J. (2010). A Task Scheduling Algorithm Based on Load Balancing in Cloud Computing, *Web Information Systems and Mining Lecture, Notes in Computer Science*, 6318, pp. 271-277.

Farahnakian F., Pahikkala T., Liljeberg P., Plosila J., Hieu N., and Tenhunen H. (2016). Energy-Aware VM Consolidation in Cloud Data Centers Using Utilization Prediction Model. *IEEE Transactions on Cloud Computing*. p. 13. Retrieved from: ieeexplore.ieee.org/document/7593250/ Accessed 19th March, 2017

Ferdaus M., Murshed M., Calheiros R. and Buyya R. (2014). Virtual Machine Consolidation in Cloud Data Centers Using ACO Metaheuristic. In *Euro-Par* (pp. 306-317).

Furht B. and Escalante A. (2010). Handbook of Cloud Computing. (3), New York: Springer

Foster I. and Kesselman C. eds., (2003). *The Grid 2: Blueprint for a new computing infrastructure*. Elsevier.

Galvin P. (2009). VMware vSphere Vs. Microsoft Hyper-V: A Technical Analysis. *Corporate Technologies, CTI Strategy White Paper*. p. 32. Retrieved from http://www.cptech.com/

Garey M. and Johnson D. (1979).*Computers and intractability: A guide to the Theory of NP-Completeness*. W.H Freeman & Co. New York, NY

Gartner (2016), Worldwide Public Cloud Services Market to Grow 17 Percent in 2016, Press Release, Retrieved from http://www.gartner.com/newsroom/id/3443517

Han H., Deyu Z., Zheng W. and Bin F. (2013). A Qos Guided task Scheduling Model in cloud computing environment. *In 4th Intl. Conf. on Emerging Intelligent Data and Web Technologies*, pp. 72-75.

Hanke S. (1999). The Performance of Concurrent Red-Black Tree Algorithms. In *Intl. Workshop on Algorithm Engineering* (pp. 286-300). Springer Berlin Heidelberg.

Hashizume K., Rosado D., Fernández-Medina E., and Fernandez E. (2013). An Analysis of Security Issues For Cloud Computing. *Journal of Internet Services and Applications*, 4(1), p.5. Retrieved from link.springer.com/article/10.1186/1869-0238-4-5. Accessed 1st April, 2015.

Hieu N., Francesco M. and Yla-Jaaski A. (2015). Virtual Machine Consolidation with Usage Prediction for Energy-Efficient Cloud Data Centers. In *Intl. Conf. on Cloud Computing (CLOUD), 2015 IEEE 8th Intl. Conf. on* (pp. 750-757). IEEE

Hieu N., Francesco M. and Yla-Jaaski A. (2017). Virtual Machine Consolidation with Multiple Usage Prediction for Energy-Efficient Cloud Data Centers. *IEEE Transactions on Services Computing*. p. 14. Retrieved from http://ieeexplore.ieee.org/document/7807360/ Accessed 21st April, 2017.

Holzle U. and Barroso L. (2007). The Case for Energy-Proportional Computing. *Computer*, 40(12), pp. 33-37.

Hu X., Zhang R. and Wang Q. (2016). Service-Oriented Resource Management in Cloud Platforms. In *Services Computing (SCC), 2016 IEEE Intl. Conf. on* (pp. 435-442). IEEE.

Islam S., Keung J., Lee K. and Liu A. (2010). An empirical study into adaptive resource provisioning in the cloud. In *Utility and Cloud Computing (UCC 2010). IEEE Intl Conf.*

on (p.8) IEEE. Retrieved from http://www.nicta.com.au/pub?doc=4349. Accessed 11th July, 2015.

Johansen L. (1968). Production Functions and the Concept of Capacity. *Recherches Recentes sur la Fonction de Production, Collection. Economie Mathematique et Econometrie*, 2, pp. 49-72

Kahrs S. (2001). Red-Black Trees with Types. *Journal of Functional Programming*, 11(4), pp. 425-432.

Kang S. and Kim M. (2015). Short-Run Cost Minimization and Capacity Utilization of Regional Public Hospitals in South Korea. Modern Economics, 6, pp. 21-29.

Kliazovich D., Bouvry P. and Khan S. (2012). GreenCloud: A Packet-level Simulator of Energy-Aware Cloud Computing Data Centers, *The Journal of Supercomputing*, 62(3), pp. 1263-1283

Lamke K. (2016). Liquid to Server Cooling: The Next Big Thing in Data Center Cooling Technology? Schneider Electric. Retrieved from http://blog.schneider-electric.com/datacenter/power-and-cooling/2016/06/02/liquid-server-cooling-next-big-thing-data-center/. Accessed 23rd June, 2017.

Lawanyashri M., Subha S. and Balusamy B. (2016). Energy-Aware Fruitfly Optimisation Algorithm for Load Balancing in Cloud Computing Environments, Intl. Journal of Intelligent Engineering and Systems, 10(1) pp. 75-85.

Lefurgy C., Wang X. and Ware M. (2007). Server–level power control. In *Proceedings of the 4th Intl. Conf. on Autonomic Computing (ICAC '07)*, p. 4

Legrand A., Marchal L., and Casanova H. (2003). Scheduling Distributed Applications: The SimGrid Simulation Framework. In *Cluster Computing and the Grid, 2003, CCGrid 2003. 3rd IEEE/ACM Intl. Symp.on* (pp. 138-145). IEEE.

Lu Y., Xie G., Kliot G., Geller A., Larus J. and Greenberg A. (2011). Join-Idle-Queue: A Novel Load Balancing Algorithm for Dynamically Scalable Web Services, *Performance Evaluation*, 68(11), pp.1056-1071.

Mahajan K., Makroo A., and Dahiya D. (2013). Round Robin with Server Affinity: A VM Load Balancing Algorithm for Cloud Based Infrastructure, *Journal of Info. Process System*, 9(3), pp. 379-394

Mell P. and Grance T. (2011). The NIST Definition of Cloud Computing. *NIST Special Publication 800-145 Whitepaper* (p.7). Retrieved from acm.org/citation.cfm?id=2206223 Accessed 21st March, 2015.

Meisner D., Gold B. and Wenisch T. (2009). PowerNap: eliminating server idle power. In *ACM SIGPLAN Notices.* 44(3), pp. 205-216. ACM.

Membrey P., Plugge E. and Hows D. (2012). Practical Load Balancing Ride the Performance Tiger, Apress.

Microsoft (2009). Microsoft® Hyper-V™ Server 2008 R2. Retrieved from https://www.microsoft.com/en-us/download/details.aspx?id=3512" Accessed 28th June, 2017.

Mosa A. and Paton N. (2016). Optimizing Virtual Machine Placement for Energy and SLA in Clouds Using Utilization. *Journal of Cloud Computing: Advances, Systems and Applications*. 5(1), p.17. Retrieved from dl.acm.org/citation.cfm?id=3013388 Accessed 6th December, 2016.

Nurmi D., Wolski R., Grzegorczyk C., Obertelli G., Soman S., Youseff L. and Zagorodnov, D. (2009). The Eucalyptus open-source cloud computing system. In *Proceedings of 9th IEEE/ACM Intl. Symposium on Cluster Computing and the Grid (CCGrid 2009)*, Shanghai, China

OpenStack (2010). Open Source Cloud Computing Software. Retrieved from https://www.openstack.org/software/ Accessed 9th April, 2015

Pallipadi V. and Starikovskiy A. (2006). The on demand governor, *Proceedings of the Linux Symposium*, vol. 2, 2006, pp. 215–230

Park K. and Pai V. (2006). CoMon: a mostly-scalable monitoring system for PlanetLab. *ACM SIGOPS Operating Systems Review*, 40(1), pp. 65–74.

Patel P., Ranabahu A. and Sheth A. (2009). Service Level Agreement in Cloud Computing. p. 10. Retrieved from http://corescholar.libraries.wright.edu/knoesis/78. Accessed 30th June, 2017.

Pennsylvania (2013). Computer Power Usage Tables. University of Pennsylvania. Retrieved from https://www.isc.upenn.edu/computer-power-usage-tables. Accessed 30th April, 2016.

Randles M., Lamb D. and Taleb-Bendiab A. (2009). Experiments with Honeybee Foraging Inspired Load Balancing. *In Developments in eSystems Engineering (DESE)*, 2009 2nd International Conference on (pp. 240-247). IEEE.

Reiss C. and Wilkes J. (2011). Google Cluster Usage Traces: format + schema' of Google Workloads. *Google Inc., Whitepaper*, pp. 1-14

Rice D., Glick J., Cercy D., Sandifer C. and Cramblitt B. (2015). Standard Performance Evaluation Corporation (SPEC). Retrieved from https: //www.spec.org/power_ssj2008/results/. Accessed 30th April, 2016

Sempolinski P., and Thain D. (2010). A Comparison and Critique of Eucalyptus, OpenNebula and Nimbus. In *Cloud Computing Technology and Science (CloudCom), 2010 2nd Intl. Conf. on* (pp. 417-426). IEEE.

Setty S. (2011). VMware vSphere vMotion Architecture, Performance and Best Practices in VMware vSphere5. VMware, Inc., *White paper*, August 2011. http://www.vmware.com/files/pdf/vmotion-perf-vSphere5.pdf. Accessed 10th April, 2015

Shahzad F. (2014). State-of-the-art Survey on Cloud Computing Security Challenges, Approaches and Solution, *Procedia Computer Science* 37 pp. 357–362.

Shehabi A., Smith S., Sartor D., Brown R., Herrlin M., Koomey J., Masanet E., Horner N., Azevedo I. and Lintner W. (2016). United States data center energy usage report. Retrieved from https://eta.lbl.gov/sites/default/files/publications/lbnl-1005775_v2.pdf. Accessed 23 June, 2017.

Sidhu A. and Kinger S. (2013). Analysis of Load Balancing Techniques in Cloud Computing. In *International Journal of Computers and Technology*, 4(2), pp. 737-741

Standard Performance Evaluation Corporation (SPEC) (2010). SPECpower ssj2008 Benchmark. Retrieved from https://www.spec.org. Accessed 30th April, 2015.

Standard Performance Evaluation Corporation (SPEC) (2016). SPEC Cloud_IaaS 2016. Retrieved from https://www.spec.org/benchmarks.html#cloud Accessed 23 June, 2017.

Tao Y. and Lee H. (2017). MTCS for Healthcare. In *Intl. Conf. of Cloud Computing Research and Innovation ICCCRI*, CloudAsia2017, p. 7.

Ullrich M., Lassig J. and Gaedke M. (2016). Towards Efficient Resource Management in Cloud Computing: A Survey. In *Future Internet of Things and Cloud (FiCloud), 2016 IEEE 4th Intl. Conf. on* (pp. 170-177). IEEE.

Vardhan V., Sachs D., Yuan W., Harris A., Adve S., Jones D., Kravets R. and Nahrstedt K. (2005). Integrating fine-grained application adaptation with global adaptation for saving energy. In *Proceedings of the International Workshop on Power-Aware Real-Time Computing, Jersey City, NJ*, 2005, pp. 1–14.

VCL. (2010). Virtual Computing Lab. Retrieved from https://cwiki.apache.org/VCL/ Accessed 10th April, 2015

Voorsluys W., Broberg J. and Buyya R. (2011). Cloud Computing: Principles and Paradigms. John Wiley & Sons, Inc.

Wickremasinghe B., Calheiros R. and Buyya R. (2010). Cloudanalyst: A cloudsim-based visual modeller for analysing cloud computing environments and applications. In *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on* pp. 446-452. IEEE.

Wu L., Garg S. and Buyya R. (2011). SLA-based Resource Allocation for Software as a Service Provider (SaaS) in Cloud Computing Environments. In *Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM Intl. Symposium on* (pp. 195-204). IEEE.

Xu L. and Li J. (2016). Building Efficient Resource Management Systems in the Cloud: Opportunities and Challenges. *Intl. Journal of Grid and Distributed Computing*, 9(3), pp. 157-172

Xu Y., Musgrave Z., Nobel B. and Bailey, M. (2014). Workload-Aware Provisioning in Public Clouds. IEEE Internet Computing, 18(4), pp. 15-21, IEEE.

Zhang L., Li Z. and Wu C. (2014). Dynamic Resource Provisioning in Cloud Computing: A Randomized Auction Approach. In *INFOCOM, 2014 Proceedings IEEE*, pp. 433-441. IEEE, 2014.

APPENDIX I

CODE LISTING

MC-BAL

```
package org.cloudbus.cloudsim.examples.power.planetlab;

import java.io.FileWriter;

import java.io.IOException;

import org.cloudbus.cloudsim.examples.power.planetlab.NonPowerAware;

import org.cloudbus.cloudsim.examples.power.planetlab.PlanetLabRunner;

import org.cloudbus.cloudsim.examples.power.planetlab.MqBalRunner;

//Actual entry point of the application

public class MQBAL_Entry {

        public static void main(String[] args) throws IOException {

                boolean enableOutput = true;

                boolean outputToFile = false;

String inputFolder =
NonPowerAware.class.getClassLoader().getResource("examples/workload/planetlab").getPat
h();

String outputFolder = "output";

      //DATASETS

                String workload = "20110325"; // PlanetLab workload        -1078

                //String workload = "GCDWorkload"; //Google Cluster Data 1 - 168

                //String workload = "GCD";   //same specs as PL (1301)

                //String workload = "gcd/GCD_VMs";          //hieu cleaned(1600)

                String vmSelectionPolicy = "classMQ";

                //thr_MQ, 0.8; mad, 2.5; iqr, 1.5

                String vmAllocationPolicy = "thr_MQ"; //"mad";//"iqr";

                String parameter = "0.8";

                System.out.println(workload + ": Running " + vmAllocationPolicy + "-" +
vmSelectionPolicy);

      System.out.println("Running MQ_BAL");

      new MqBalRunner(enableOutput, outputToFile, inputFolder, outputFolder, workload,
vmAllocationPolicy, vmSelectionPolicy, parameter);          }         }
```

MC-BAL RUNNER CLASS

```
public class MqBalRunner extends RunnerAbstract_MQ {


public MqBalRunner( boolean enableOutput, boolean outputToFile, String inputFolder,
String outputFolder,String workload, String vmAllocationPolicy,    String vmSelectionPolicy,
String parameter) {

super(enableOutput, outputToFile,inputFolder,  outputFolder, workload, vmAllocationPolicy,
vmSelectionPolicy, parameter);                              }


@Override

protected void init(String inputFolder) {

try {

        CloudSim.init(1, Calendar.getInstance(), false);

        //1. create new instance of broker (gBroker)

        gBroker = Helper.createBroker();

        int brokerId = gBroker.getId();

        //create and populate VM list

gCloudletList = MqBalHelper.createCloudletListMqBal(brokerId, inputFolder);//sBrokerId,
bBrokerId, inputFolderName)

//allCloudlets r now a list of lists

//create vm for all cloudlets

gVmList = Helper.createVmList(brokerId, gCloudletList.size());

//create the hosts to run the VMs

hostList = Helper.createHostList(NUMBER_OF_HOSTS);

}

catch (Exception e) {

e.printStackTrace();

System.exit(0);

}

}                   }
```

MC-BAL RUNNER ABSTRACT CLASS - EXRTACT

```
package org.cloudbus.cloudsim.examples.power;

import java.io.File;

import java.io.FileNotFoundException;

import java.io.FileOutputStream;

import java.io.IOException;

import java.util.List;

import org.cloudbus.cloudsim.Cloudlet;

import org.cloudbus.cloudsim.DatacenterBroker;

import org.cloudbus.cloudsim.VmAllocationPolicy;

import org.cloudbus.cloudsim.core.CloudSim;

import org.cloudbus.cloudsim.power.PowerDatacenter;

import org.cloudbus.cloudsim.power.PowerHost;

import org.cloudbus.cloudsim.power.PowerVmAllocationPolicyMigrationAbstract;

import
org.cloudbus.cloudsim.power.PowerVmAllocationPolicyMigrationInterQuartileRange;

import org.cloudbus.cloudsim.power.PowerVmAllocationPolicyMigrationLocalRegression;

import
org.cloudbus.cloudsim.power.PowerVmAllocationPolicyMigrationLocalRegressionRobust;

import
org.cloudbus.cloudsim.power.PowerVmAllocationPolicyMigrationMedianAbsoluteDeviation
;

import org.cloudbus.cloudsim.power.PowerVmAllocationPolicyMigrationStaticThreshold;

import org.cloudbus.cloudsim.power.PowerVmAllocationPolicySimple;

import org.cloudbus.cloudsim.power.PowerVmSelectionPolicy;

import org.cloudbus.cloudsim.power.PowerVmSelectionPolicyMaximumCorrelation;

import org.cloudbus.cloudsim.power.PowerVmSelectionPolicyMinimumMigrationTime;

import org.cloudbus.cloudsim.power.PowerVmSelectionPolicyMinimumUtilization;

import org.cloudbus.cloudsim.power.PowerVmSelectionPolicyRandomSelection;


public abstract class RunnerAbstract {

        private static boolean enableOutput;                    /** The enable output. */
```

```
protected static DatacenterBroker broker;          /** The broker. */
protected static List<Cloudlet> cloudletList;      /** The cloudlet list. */
protected static List<Vm> vmList;                  /** The vm list. */
protected static List<PowerHost> hostList;         /** The host list. */

public RunnerAbstract(boolean enableOutput, boolean outputToFile, String inputFolder,
String outputFolder, String workload, String vmAllocationPolicy, String vmSelectionPolicy,
String parameter) {
    try {
        initLogOutput(enableOutput, outputToFile, outputFolder, workload,
        vmAllocationPolicy, vmSelectionPolicy, parameter);
    } catch (Exception e) {
        e.printStackTrace();
        System.exit(0);
    }
    init(inputFolder + "/" + workload);
    start(getExperimentName(workload, vmAllocationPolicy, vmSelectionPolicy,
    parameter),
    outputFolder,
    getVmAllocationPolicy(vmAllocationPolicy, vmSelectionPolicy, parameter));
    }
    /**
     * Starts the simulation.
     */
protected void start(String experimentName, String outputFolder, VmAllocationPolicy
vmAllocationPolicy) {
        try {
        PowerDatacenter datacenter = (PowerDatacenter) Helper.createDatacenter(
        "Datacenter", PowerDatacenter.class, hostList, vmAllocationPolicy);
        datacenter.setDisableMigrations(false);
        broker.submitVmList(vmList);
        broker.submitCloudletList(cloudletList);
        CloudSim.terminateSimulation(Constants.SIMULATION_LIMIT);
```

```
double lastClock = CloudSim.startSimulation();
List<Cloudlet> newList = broker.getCloudletReceivedList();
CloudSim.stopSimulation();

Helper.printResults(datacenter, vmList, lastClock, experimentName,
Constants.OUTPUT_CSV, outputFolder);

Helper.getThroughput(vmList);        //added 4 thruput - supoAjayi, 09.02.2017
} catch (Exception e) {
Log.printLine("The simulation has been terminated due to an unexpected error");
                System.exit(0);                                    }                }
protected VmAllocationPolicy getVmAllocationPolicy(String vmAllocationPolicyName,
String vmSelectionPolicyName, String parameterName) {
        VmAllocationPolicy vmAllocationPolicy = null;
        PowerVmSelectionPolicy vmSelectionPolicy = null;
        if (!vmSelectionPolicyName.isEmpty()) {
            vmSelectionPolicy = getVmSelectionPolicy(vmSelectionPolicyName); }
           double parameter = 0;
        if (!parameterName.isEmpty()) {
                parameter = Double.valueOf(parameterName);
        }
        if (vmAllocationPolicyName.equals("iqr")) {
PowerVmAllocationPolicyMigrationAbstract fallbackVmSelectionPolicy = new
PowerVmAllocationPolicyMigrationStaticThreshold(hostList, vmSelectionPolicy, 0.7);
vmAllocationPolicy = new PowerVmAllocationPolicyMigrationInterQuartileRange(
hostList, vmSelectionPolicy, parameter, fallbackVmSelectionPolicy);                }
else if (vmAllocationPolicyName.equals("mad")) {
PowerVmAllocationPolicyMigrationAbstract fallbackVmSelectionPolicy = new
PowerVmAllocationPolicyMigrationStaticThreshold(hostList, vmSelectionPolicy, 0.7);
vmAllocationPolicy = new PowerVmAllocationPolicyMigrationMedianAbsoluteDeviation(
hostList, vmSelectionPolicy, parameter, fallbackVmSelectionPolicy);                }
```

```
else if (vmAllocationPolicyName.equals("lr")) {
PowerVmAllocationPolicyMigrationAbstract fallbackVmSelectionPolicy = new
PowerVmAllocationPolicyMigrationStaticThreshold(hostList, vmSelectionPolicy, 0.7);
vmAllocationPolicy = new PowerVmAllocationPolicyMigrationLocalRegression(
hostList, vmSelectionPolicy, parameter, Constants.SCHEDULING_INTERVAL,
fallbackVmSelectionPolicy);            }
else if (vmAllocationPolicyName.equals("lrr"))
{PowerVmAllocationPolicyMigrationAbstract fallbackVmSelectionPolicy = new
PowerVmAllocationPolicyMigrationStaticThreshold(hostList, vmSelectionPolicy,0.7);
                        vmAllocationPolicy = new
PowerVmAllocationPolicyMigrationLocalRegressionRobust(hostList, vmSelectionPolicy,
parameter, Constants.SCHEDULING_INTERVAL, fallbackVmSelectionPolicy);            }
else if (vmAllocationPolicyName.equals("thr")) {
vmAllocationPolicy = new PowerVmAllocationPolicyMigrationStaticThreshold(
hostList, vmSelectionPolicy, parameter);}
else if (vmAllocationPolicyName.equals("dvfs")) {
        vmAllocationPolicy = new PowerVmAllocationPolicySimple(hostList);}
else {                  System.out.println("Unknown VM allocation policy: " +
vmAllocationPolicyName);
                        System.exit(0);
                }
                return vmAllocationPolicy;
        }

protected PowerVmSelectionPolicy getVmSelectionPolicy(String vmSelectionPolicyName) {
                PowerVmSelectionPolicy vmSelectionPolicy = null;
                if (vmSelectionPolicyName.equals("mc")) {
                        vmSelectionPolicy = new
PowerVmSelectionPolicyMaximumCorrelation(
                                        new
PowerVmSelectionPolicyMinimumMigrationTime());
                } else if (vmSelectionPolicyName.equals("mmt")) {
                        vmSelectionPolicy = new
PowerVmSelectionPolicyMinimumMigrationTime();
```

```
                } else if (vmSelectionPolicyName.equals("mu")) {
                        vmSelectionPolicy = new
PowerVmSelectionPolicyMinimumUtilization();
                } else if (vmSelectionPolicyName.equals("rs")) {
                        vmSelectionPolicy = new
PowerVmSelectionPolicyRandomSelection();
                } else {
                        System.out.println("Unknown VM selection policy: " +
vmSelectionPolicyName);
                        System.exit(0);
                }
                return vmSelectionPolicy;
        }


}
```

CLASS-BASED VM SELECTION POLICY

```
package org.cloudbus.cloudsim.power;
import java.util.List;
import org.cloudbus.cloudsim.Vm;
import org.cloudbus.cloudsim.core.CloudSim;
import org.cloudbus.cloudsim.power.PowerHost;
import org.cloudbus.cloudsim.power.PowerVm;
import org.cloudbus.cloudsim.power.PowerVmSelectionPolicy;
import org.cloudbus.cloudsim.examples.power.Constants_MQ;

/**
 * The ClassBased VM selection policy.
 *
 * This selects the next VM to migrate based on workload class
 * Available classes are
 * Gold (highest priority), Silver and Bronze (Least Priority)
 *
 * Selected VM & PM from which there r selected r placed on the M-Queue
 * @author olasupoAjayi
 * @since 28.01.2016
 */

public class VmSelectionPolicyClassBased extends PowerVmSelectionPolicy {
	public VmSelectionPolicyClassBased (int wkldLen)
	{
		this.gVmId = (int)(wkldLen * 0.2);
		this.sVmId = (int)(wkldLen * 0.6);
		this.bVmId = (int)(wkldLen);
	}
```

```
        int gVmId;
        int sVmId;
        int bVmId;
@Override
public Vm getVmToMigrate(PowerHost host) {
        Vm vmToMigrate = null;
                //get VMs allocated to PM (host)
        List<PowerVm> migratableVms = getMigratableVms(host);
                if (!migratableVms.isEmpty()) {
                        vmToMigrate = checkBronze(migratableVms);
                if (vmToMigrate == null) {    //no bronze, check 4 silver
                        vmToMigrate = checkSilver(migratableVms);                       }
                if (vmToMigrate == null) {//no silver, check 4 gold
                        vmToMigrate = checkGold(migratableVms);                         }
                }
                return vmToMigrate;
        }

private Vm checkBronze(List<PowerVm> ListofMigratableVms)   {
                Vm v = null;
//run thru list, test and pick bronze first
        for (Vm vm : ListofMigratableVms) {
                if (vm.isInMigration())
                                continue;
                if (vm.getId() > sVmId)         //test and pick bronze first
                                v = vm;
                }
                return v;
        }
```

```
private Vm checkSilver(List<PowerVm> ListofMigratableVms)     {
                Vm v = null;
                //if not bronze found, run thru list, test and pick silver next
        for (Vm vm : ListofMigratableVms) {
                if (vm.isInMigration())
                        continue;
                if (vm.getId() > gVmId & vm.getId() <= sVmId)      //test and pick silver next
                        v = vm;
                }
                return v;
        }

private Vm checkGold(List<PowerVm> ListofMigratableVms)      {
                Vm v = null;
                ////finally pick Gold in d absence of others
        for (Vm vm : ListofMigratableVms) {
                if (vm.isInMigration())
                        continue;
                if  (vm.getId() <= gVmId)
                                v = vm;
                }
                return v;
                //perform MIPS test as tie breaker
        }
}
```

DOUBLE-DEPTH HALF INTERVAL SCHEDULING

```
package org.cloudbus.cloudsim.power;
import java.util.Collection;
import java.util.Iterator;
import java.util.List;
import java.util.NavigableSet;
import java.util.Set;
import org.cloudbus.cloudsim.Host;
import org.cloudbus.cloudsim.Vm;
import com.google.common.collect.Multimap;
import com.google.common.collect.Ordering;
import com.google.common.collect.TreeMultimap;
/**
 * This class binarySearch replaces the conventional LinearSearch
 * with Binary Search (Red-black). In a bid to reduce VM allocation
 * Time in DCs with large number of PM. IE from O(n) to O(log(n))
 * If an exact matching host is not found, the next closest match (greater than it)
 * is returned
  * @author ooAjayi
 * @since 24.05.2016
 * */

class  binarySearchBF           {
                    double reqdMIPS;
                    PowerHost selectedHost = null;
                    List <PowerHost> hosts = null;
                    private  Multimap<Double, PowerHost> BTree;

binarySearchBF(List <PowerHost> p )         {
                            this.hosts = p;
```

```
                BTree = createHostListofAvailableMIPS();
}
//called occasionally to update the BTree
public void updateBT(List <PowerHost> p)        {
                this.hosts = p;
                BTree = createHostListofAvailableMIPS();
}

        /**
         *
         * @return a Multimap of AvailableMIPS and PowerHost
         * Its a balanced red-black Tree hence with log (n) speed
         * Uses Guava's Multimap and
         * Returns the best fit PM for the required MIPS
         * @author olasupoAjayi
         * @since 15.12.2016
         */
Multimap<Double, PowerHost> createHostListofAvailableMIPS()        {
    List<PowerHost> tempList = hosts;
    Multimap<Double, PowerHost> listOfAvailableMIPS =
TreeMultimap.create(Ordering.natural(), Ordering.arbitrary());

    for (PowerHost p : tempList)        {
    //build tree
        listOfAvailableMIPS.put(p.getAvailableMips(), p);
                }
        return listOfAvailableMIPS;
        }
public Multimap<Double, PowerHost> getBTree()   {
                return BTree;
    }
```

```
/**
* This method searches for the best matching PM for a required vm's MIPS
* getHost also checks if the selected PM has a utilization of zero. getHostOnly does not do this
* @param vm
* @return PowerHost
*/
PowerHost getHost (Vm vm)                    {
        this.reqdMIPS = vm.getMips();
        Multimap<Double, PowerHost> sortedPHs = getBTree();
        PowerHost selectedHost = null;
                        //gets mips that are >= reqdMIPS ie best matches
        NavigableSet<Double> ns =  (NavigableSet<Double>) sortedPHs.keySet();
        Double x = ns.ceiling(reqdMIPS);
        //get all PMs that can provide reqdMIPS
        Collection<PowerHost> selectedHost1 = sortedPHs.get(x);
        Iterator<PowerHost> i = selectedHost1.iterator();
        while (i.hasNext())                                {
                        PowerHost s = i.next();
                        if (!s.isSuitableForVm(vm))             {
                                        continue;               }
                        else                                    {
                        if (getUtilizationOfCpuMips(s) == 0)
                        return  selectedHost = s;
                        else
                                continue;                       }
                        }
                        //if best match isnt found, return FIRST match
                if (selectedHost == null)
                        selectedHost = adjuster(vm, selectedHost1, sortedPHs, null);
```

```
                return selectedHost;
        }

        /**
         * This method recursively searches for a suitable PM
         * It is called by all getHost methods as a last resort to finding
         * a suitable PM. It returns the next first fit
         * @param vm
         * @param possibleHosts
         * @param tree
         * @return
         */
PowerHost adjuster (Vm vm, Collection<PowerHost> possibleHosts, Multimap<Double,
PowerHost> tree, Set<? extends Host> excludedHosts)     {     //remove
unsuitable hosts ie previously tested collections
    Iterator keyItr = tree.keySet().iterator();
    keyItr.next();
    keyItr.remove();
    //get next matching PMs
    NavigableSet<Double> ns =  (NavigableSet<Double>) tree.keySet();
    Double x = ns.ceiling(vm.getMips());
    PowerHost sHost = null;
    Collection<PowerHost> nextKeySet = tree.get(x);
    Iterator<PowerHost> i = nextKeySet.iterator();
    while (i.hasNext())     {
        PowerHost s = i.next();
        if (excludedHosts.contains(s))     {     continue;                    }
    if (s.isSuitableForVm(vm))    {
            if (getUtilizationOfCpuMips(s) != 0) {
                    continue;          }
                sHost = s;
```

```
                        }
                    }
        if (sHost == null)
            sHost = adjuster(vm, nextKeySet, tree, excludedHosts);
            return sHost;
        }

            /**
             * Used in the initialization / allocation stage by some classes
             * PowerVmAllocation
             * @param vm
             * @return
             */
PowerHost getHostOnly (Vm vm)           {
        this.reqdMIPS = vm.getMips();
        Multimap<Double, PowerHost> sortedPHs = getBTree();
        //get next matching PMs
        NavigableSet<Double> ns =  (NavigableSet<Double>) sortedPHs.keySet();
        Double x = ns.ceiling(reqdMIPS);
        PowerHost sHost = null;
        Collection<PowerHost> nextKeySet = sortedPHs.get(x);
        Iterator<PowerHost> i = nextKeySet.iterator();
        while (i.hasNext())                 {
                    PowerHost s = i.next();
        if (s.isSuitableForVm(vm))          {           return sHost = s;           }
        else                    continue;                                       }
        if (sHost == null)              {
        try     {
            sHost = adjusterOnly(vm, nextKeySet, sortedPHs);
            }
```

```
        catch (Exception e)             {
                sHost = null;
                }
                }
                return sHost;
        }


PowerHost adjusterOnly (Vm vm, Collection<PowerHost> possibleHosts,
Multimap<Double, PowerHost> tree)
{
        Iterator keyItr = tree.keySet().iterator();
        keyItr.next();
        keyItr.remove();


        //get next matching PMs
        NavigableSet<Double> ns =  (NavigableSet<Double>) tree.keySet();
        Double x = ns.ceiling(vm.getMips());
        PowerHost sHost = null;
        Collection<PowerHost> nextKeySet = tree.get(x);
        Iterator<PowerHost> i = nextKeySet.iterator();
                while (i.hasNext())             {
                        PowerHost s = i.next();
                        if (!s.isSuitableForVm(vm))
                                        continue;
                                else
                                        return  sHost = s;
}
        if (sHost == null)
                sHost = adjusterOnly(vm, nextKeySet, tree);
                return sHost;
        }
```

```
PowerHost getHost (Vm vm, Set<? extends Host> excludedHosts)          {
        this.reqdMIPS = vm.getMips();
        Multimap<Double, PowerHost> sortedPHs = getBTree();
        PowerHost sHost = null;
        //get next matching PMs
        NavigableSet<Double> ns = (NavigableSet<Double>) sortedPHs.keySet();
            Double x = ns.ceiling(reqdMIPS);
            Collection<PowerHost> nextKeySet = sortedPHs.get(x);
            Iterator<PowerHost> vItr = nextKeySet.iterator();
                while (vItr.hasNext()) {
                PowerHost s = vItr.next();
                if (excludedHosts.contains(s))                {
                    continue;
                }
                if (s.isSuitableForVm(vm))
                {
                if (getUtilizationOfCpuMips(s) != 0) {
                            continue;                         }
                    sHost = s;                                }

            }

            if (sHost == null)
            {
        try     {
        sHost = adjuster(vm, nextKeySet, sortedPHs, excludedHosts);
        }
        catch (Exception e)    {              sHost = null;                    }
            }
            return sHost;
```

```
                }


        /**
         * Gets utilization of CPU in MIPS 4d current potentially allocated VMs.
          * @param host the host
         * @return the utilization of the CPU in MIPS
         */
        double getUtilizationOfCpuMips(PowerHost host) {
                double hostUtilizationMips = 0;
                for (Vm vm2 : host.getVmList()) {
                        if (host.getVmsMigratingIn().contains(vm2)) {
hostUtilizationMips += host.getTotalAllocatedMipsForVm(vm2) * 0.9 / 0.1;
                        }
hostUtilizationMips += host.getTotalAllocatedMipsForVm(vm2);
                }
                return hostUtilizationMips;
        }


}
```

VM ALLOCATION POLICY - EXRTACT

```
public abstract class PowerVmAllocationPolicyAbstract_MC extends VmAllocationPolicy
{

private final Map<String, Host> vmTable = new HashMap<String, Host>();

private binarySearchBF bsbfx;

private int oldHostListSize = 0;

public PowerVmAllocationPolicyAbstract_MQ(List<? extends Host> list) {         super(list);

bsbfx = new binarySearchBF(this.<PowerHost> getHostList()); oldHostListSize =
getHostList().size(); }

@Override

public boolean allocateHostForVm(Vm vm) {

return allocateHostForVm(vm, findHostForVm2(vm,1));

//type 0 = firstFit or original / MC-BAL/PABFD, 1 = binary (MQ-BAL (BSBF), 2 = BestFit,
3 = WorstFit, 4 = RandomFit  }

@Override

public boolean allocateHostForVm(Vm vm, Host host) {

if (host == null)          {          return false;                }

if (host.vmCreate(vm)) {        getVmTable().put(vm.getUid(), host);          return true;      }

return false;                }

public PowerHost findHostForVm2(Vm vm, int type) {       PowerHost ph = null;

switch (type)   {

case 0:

FirstFit ff = new FirstFit(this.<PowerHost> getHostList());  ph = ff.getHost(vm);   break;

case 1: if (getHostList().size() > oldHostListSize)     { bsbfx.updateBT(this.<PowerHost>
getHostList()); ph = bsbfx.getHostOnly(vm);                                  }

else      ph = bsbfx.getHostOnly(vm);  oldHostListSize = getHostList().size(); break;  }

public PowerHost findHostForVm(Vm vm) {

for (PowerHost host : this.<PowerHost> getHostList()) {     if (host.isSuitableForVm(vm)) {
          return host;                          }          }          return null;      }

public PowerHost findHostForVm_BSBF(Vm vm) {  return bsbf.getHostOnly(vm);  } }
```

MEDIAN ABSOLUTE DEVIATION - EXRTACT

```
package org.cloudbus.cloudsim.power;
import java.util.List;
import org.cloudbus.cloudsim.Host;
import org.cloudbus.cloudsim.Log;
import org.cloudbus.cloudsim.Vm;
import org.cloudbus.cloudsim.examples.power.planetlab.UP;
import org.cloudbus.cloudsim.util.MathUtil;
public class PowerVmAllocationPolicyMigrationMedianAbsoluteDeviation_MQ extends
PowerVmAllocationPolicyMigrationAbstract_MQ {
private double safetyParameter = 0;
private PowerVmAllocationPolicyMigrationAbstract fallbackVmAllocationPolicy;
public PowerVmAllocationPolicyMigrationMedianAbsoluteDeviation_MQ(
List<? extends Host> hostList, PowerVmSelectionPolicy vmSelectionPolicy,
double safetyParameter, PowerVmAllocationPolicyMigrationAbstract
fallbackVmAllocationPolicy, double utilizationThreshold) {
        super(hostList, vmSelectionPolicy);
                setSafetyParameter(safetyParameter);
                setFallbackVmAllocationPolicy(fallbackVmAllocationPolicy);
        }
public PowerVmAllocationPolicyMigrationMedianAbsoluteDeviation_MQ(
List<? extends Host> hostList, PowerVmSelectionPolicy vmSelectionPolicy,
double safetyParameter, PowerVmAllocationPolicyMigrationAbstract
fallbackVmAllocationPolicy) {
        super(hostList, vmSelectionPolicy);
                setSafetyParameter(safetyParameter);
                setFallbackVmAllocationPolicy(fallbackVmAllocationPolicy);
        }

@Override
protected boolean isHostOverUtilized_UP(PowerHost host) {
PowerHostUtilizationHistory _host = (PowerHostUtilizationHistory) host;
        double upperThreshold = 0;
        try {
        upperThreshold = 1 - getSafetyParameter() * getHostUtilizationMad(_host);
                } catch (IllegalArgumentException e) {
                        return getFallbackVmAllocationPolicy().isHostOverUtilized(host);
                }
                addHistoryEntry(host, upperThreshold);
                double totalRequestedMips = 0;
                for (Vm vm : host.getVmList()) {
                        totalRequestedMips += vm.getCurrentRequestedTotalMips();
                }
        double utilization = totalRequestedMips / host.getTotalMips();
        double[] data = _host.getUtilizationHistory();
       double predictedValue = 0;
          if(data.length >= 12)
          {
```

```
                UP up = new UP();
                predictedValue = up.UsagePrediction(data, 1);

                if(utilization > upperThreshold && predictedValue > upperThreshold)
                {
                   return true;
                }
                else
                   return false;
             }
             else
                return utilization > upperThreshold;
        }

protected double getHostUtilizationMad(PowerHostUtilizationHistory host) throws
IllegalArgumentException {
        double[] data = host.getUtilizationHistory();
        if (MathUtil.countNonZeroBeginning(data) >= 12) {
                        return MathUtil.mad(data);
                }
                throw new IllegalArgumentException();
        }

public void setFallbackVmAllocationPolicy(
        PowerVmAllocationPolicyMigrationAbstract fallbackVmAllocationPolicy) {
                this.fallbackVmAllocationPolicy = fallbackVmAllocationPolicy;
        }

public PowerVmAllocationPolicyMigrationAbstract getFallbackVmAllocationPolicy() {
        return fallbackVmAllocationPolicy;
        }

}
```

STATIC THRESHOLD WITHOUT USAGE PREDICTION

```
public class PowerVmAllocationPolicyMigrationStaticThreshold extends

PowerVmAllocationPolicyMigrationAbstract {
        private double utilizationThreshold = 0.8;

public PowerVmAllocationPolicyMigrationStaticThreshold(List<? extends Host> hostList,
PowerVmSelectionPolicy vmSelectionPolicy, double utilizationThreshold)
{
super(hostList, vmSelectionPolicy); setUtilizationThreshold(utilizationThreshold);
}

@Override
protected boolean isHostOverUtilized(PowerHost host) {
        addHistoryEntry(host, getUtilizationThreshold());
        double totalRequestedMips = 0;
        for (Vm vm : host.getVmList()) {
                totalRequestedMips += vm.getCurrentRequestedTotalMips();
        }
        double utilization = totalRequestedMips / host.getTotalMips();
        return utilization > getUtilizationThreshold();
}

protected void setUtilizationThreshold(double utilizationThreshold) {
        this.utilizationThreshold = utilizationThreshold;
}

protected double getUtilizationThreshold() {
        return utilizationThreshold; }
}
```

MINIMUM MIGRATION TIME VM SELECTION POLICY

```
public class PowerVmSelectionPolicyMinimumMigrationTime extends
PowerVmSelectionPolicy {

@Override
public Vm getVmToMigrate(PowerHost host) {
        List<PowerVm> migratableVms = getMigratableVms(host);
if (migratableVms.isEmpty()) {
        return null;
}

        Vm vmToMigrate = null;
        double minMetric = Double.MAX_VALUE;
        for (Vm vm : migratableVms) {
                if (vm.isInMigration()) {
                        continue;
        }
```

```
        double metric = vm.getRam();
        if (metric < minMetric) {
                minMetric = metric;
                vmToMigrate = vm;
        }
        }
        return vmToMigrate;
        }
}
```

PABFD - EXRTACT

```
public class VMCUP_PlanetLab {

public static void main(String[] args) throws IOException {

        boolean enableOutput = true;

        boolean outputToFile = false;

String inputFolder =
NonPowerAware.class.getClassLoader().getResource("examples/workload/planetlab").getPat
h();

        String outputFolder = "output";

        String workload = "20110325"; // PlanetLab workload

        //String workload = "gcd/GCD_VMs"; //new gcd

String vmSelectionPolicy = "mmt"; // Minimum Migration Time (MMT) VM selection
policy

        //1. THR threshold

        String vmAllocationPolicy = "thr"; // Static Threshold (THR) VM allocation policy

        String parameter = "0.8"; // the static utilization threshold

        new PlanetLabRunner(enableOutput, outputToFile, inputFolder, outputFolder,
        workload, vmAllocationPolicy, vmSelectionPolicy, parameter);


//2. IQR threshold

vmAllocationPolicy = "iqr"; // Inter Quartile Range (IQR) VM allocation policy

parameter = "1.5"; // the safety parameter of the IQR policy

new PlanetLabRunner(enableOutput, outputToFile, inputFolder, outputFolder, workload,
vmAllocationPolicy, vmSelectionPolicy, parameter);
```

```
//3. MAD threshold
vmAllocationPolicy = "mad"; // Median Absolute Deviation (MAD) VM allocation policy
parameter = "2.5"; // the safety parameter of the MAD policy
new PlanetLabRunner(enableOutput, outputToFile, inputFolder, outputFolder, workload,
vmAllocationPolicy, vmSelectionPolicy, parameter);
}
```

VMCUP-M - EXRTACT

```
public class VMCUP_M {
public static void main(String[] args) throws IOException {
boolean enableOutput = true; boolean outputToFile = false;
String inputFolder =
NonPowerAware.class.getClassLoader().getResource("examples/workload/planetlab").getPat
h();
String outputFolder = "output2";
String workload = "20110325";
//1. STATIC THR
String vmAllocationPolicy = "thr"; // Static Threshold (THR) VM allocation policy
String vmSelectionPolicy = "mmt"; // Minimum Migration Time (MMT) VM selection
policy
String parameter = "0.8"; // the static utilization threshold
new PlanetLabRunner_MUP(enableOutput, outputToFile, inputFolder, outputFolder,
workload, vmAllocationPolicy, vmSelectionPolicy, parameter);

//2. MAD threshold
vmAllocationPolicy = "mad"; // Median Absolute Deviation (MAD) VM allocation policy
parameter = "2.5"; // the safety parameter of the MAD policy
new PlanetLabRunner_MUP(enableOutput, outputToFile, inputFolder, outputFolder,
workload, vmAllocationPolicy, vmSelectionPolicy, parameter);

//3. IQR threshold
vmAllocationPolicy = "iqr"; // Inter Quartile Range (IQR) VM allocation policy
parameter = "1.5"; // the safety parameter of the IQR policy
```

```
new PlanetLabRunner_MUP(enableOutput, outputToFile, inputFolder, outputFolder,
workload, vmAllocationPolicy, vmSelectionPolicy, parameter);


}

}
```

USAGE PREDICTION

```
public class UP {

public double UsagePrediction(double[] testSetInput, int numberOfInput)     {

double predictedValue = 0;     double[] testSet = new double[testSetInput.length];

for (int i = 0; i < testSetInput.length; i++) { testSet[i] = testSetInput[testSetInput.length - i -
1];       }


double[][] x = new double[testSet.length-numberOfInput][numberOfInput + 1];

double[] y = new double[testSet.length-numberOfInput];

for(int i=0; i<testSet.length-numberOfInput; i++)            {             x[i][0] = 1.0;

 int j=0;

 for(j=0; j<numberOfInput; j++)       {

x[i][j+1] = testSet[i+j];          y[i] = testSet[i+j];       }


Matrix X = new Matrix(x);

Matrix Y = new Matrix(y, testSet.length-numberOfInput);

Matrix beta;

try            {

QRDecomposition qr = new QRDecomposition(X); beta = qr.solve(Y);

        predictedValue = beta.get(0, 0);

        for(int k=0; k<numberOfInput; k++)

        predictedValue = predictedValue + beta.get(k+1, 0) * testSet[testSet.length -
        numberOfInput + k];

        return predictedValue;

}

catch(Exception e) {
```

```
        return testSet[0];

        }

        }

}
```

HELPER CLASS FOR MC-BAL, PABFD AND VMCUP-M - EXRTACT

```
public class Helper {

//create VMs
public static List<Vm> createVmList(int brokerId, int vmsNumber) {
        List<Vm> vms = new ArrayList<Vm>();
for (int i = 0; i < vmsNumber; i++) {
        int vmType = i / (int) Math.ceil((double) vmsNumber / Constants.VM_TYPES);

        vms.add(new PowerVm(i, brokerId, Constants.VM_MIPS[vmType],
        Constants.VM_PES[vmType], Constants.VM_RAM[vmType], Constants.VM_BW,
        Constants.VM_SIZE, 1, "Xen", new
        CloudletSchedulerDynamicWorkload(Constants.VM_MIPS[vmType],
        Constants.VM_PES[vmType]), Constants.SCHEDULING_INTERVAL));
                }
                return vms;
}

//create Hosts
public static List<PowerHost> createHostList(int hostsNumber) {
        List<PowerHost> hostList = new ArrayList<PowerHost>();
        for (int i = 0; i < hostsNumber; i++) {
                int hostType = i % Constants.HOST_TYPES;

                List<Pe> peList = new ArrayList<Pe>();
        for (int j = 0; j < Constants.HOST_PES[hostType]; j++)
        {
                 peList.add(new Pe(j, new
                PeProvisionerSimple(Constants.HOST_MIPS[hostType])));
        }

        hostList.add(new PowerHostUtilizationHistory(i, new
        RamProvisionerSimple(Constants.HOST_RAM[hostType]), new
        BwProvisionerSimple(Constants.HOST_BW), Constants.HOST_STORAGE, peList,
        new VmSchedulerTimeSharedOverSubscription(peList),
        Constants.HOST_POWER[hostType]));
        }
                return hostList;
}

//create Broker
```

```
public static DatacenterBroker createBroker() {
	DatacenterBroker broker = null;
	try
	{
		broker = new PowerDatacenterBroker("Broker");
	}
	catch (Exception e)
	{
		e.printStackTrace();
		System.exit(0);
	}
		return broker;
}

//create DC
public static Datacenter createDatacenter(String name, Class<? extends Datacenter>
datacenterClass, List<PowerHost> hostList, VmAllocationPolicy vmAllocationPolicy)
throws Exception {
	String arch = "x86";
	String os = "Linux";
	String vmm = "Xen";
	double time_zone = 1.0;
	double cost = 3.0;
	double costPerMem = 0.0001;
	double costPerStorage = 0.0;
	double costPerBw = 0.0;
	DatacenterCharacteristics characteristics = new DatacenterCharacteristics(
arch, os, vmm, hostList, time_zone, cost, costPerMem, costPerStorage, costPerBw);

Datacenter datacenter = null;
try
{
	datacenter = datacenterClass.getConstructor(String.class,
DatacenterCharacteristics.class, VmAllocationPolicy.class, List.class,
Double.TYPE).newInstance(name, characteristics, vmAllocationPolicy, new
LinkedList<Storage>(), Constants.SCHEDULING_INTERVAL);
}
catch (Exception e) {
	e.printStackTrace();
	System.exit(0);
}
	return datacenter;
}

public static List<Cloudlet> createCloudletListFromSWF(int brokerId, String
inputFolderName) throws NumberFormatException, IOException
{
	List<Cloudlet> allTypes = new ArrayList<Cloudlet>();
	long fileSize = 300;		//y 300??
	long outputSize = 300;
```

```
        UtilizationModel utilizationModelNull = new UtilizationModelNull();

        WorkloadModel r;
        try {
                r = new WorkloadFileReader("src" + File.separator + "examples" +
File.separator +"workload"+ File.separator +"planetlab"+ File.separator + "UniLu-Gaia-
2014-1.swf.gz", 1);
                r.generateWorkload();
        for(Cloudlet c : allTypes)
        {
                c.setUserId(brokerId);
                c.setVmId(allTypes.indexOf(c));
                c.setCloudletLength((int)(Constants_MQ.CLOUDLET_LENGTH));
                c.setNumberOfPes(Constants_MQ.CLOUDLET_PES);
        }
        } catch (FileNotFoundException e) {          e.printStackTrace();                    }
```

CONSTANTS

```
package org.cloudbus.cloudsim.examples.power;


import org.cloudbus.cloudsim.power.models.PowerModel;

import
org.cloudbus.cloudsim.power.models.PowerModelSpecPowerHpProLiantMl110G4Xeon304
0;

import
org.cloudbus.cloudsim.power.models.PowerModelSpecPowerHpProLiantMl110G5Xeon307
5;


/**

 * If you are using any algorithms, policies or workload included in the power package,
please cite

 * the following paper:

 *

 * Anton Beloglazov, and Rajkumar Buyya, "Optimal Online Deterministic Algorithms and
Adaptive

 * Heuristics for Energy and Performance Efficient Dynamic Consolidation of Virtual
Machines in

 * Cloud Data Centers", Concurrency and Computation: Practice and Experience (CCPE),
Volume 24,
```

```
 * Issue 13, Pages: 1397-1420, John Wiley & Sons, Ltd, New York, USA, 2012
 *
 * @author Anton Beloglazov
 * @since Jan 6, 2012
 */

//Constants class for MQ-BAL - modification of the original constants class by Anton

public class Constants_MQ {

        //no of Hosts in the DC value set to 800
        public static int maxNUMBER_OF_HOSTS = 1200;
        public static int NUMBER_OF_HOSTS = 500; //800;

        public final static boolean ENABLE_OUTPUT = true;
        public final static boolean OUTPUT_CSV    = false;

      //For PlatnetLab VMs (for 24 hours), so the SIMULATION_LIMIT is set to 24*60*60
      //Experiment time = 24 hours
        public final static double SCHEDULING_INTERVAL = 300;
        public final static double SIMULATION_LIMIT = 24 * 60 * 60;

      //For Google Cluster Data (for 6 hours), so the SIMULATION_LIMIT is set to 6*60*60
     //Experiment time = 6 hours
         public final static double SCHEDULING_INTERVAL = 300;
        public final static double SIMULATION_LIMIT = 6 * 60 * 60;

//For Google Cluster Data 2 (3 days), so the SIMULATION_LIMIT is set to 3*24*60*60
//    //Experiment time = 3 * 24 hours
        public final static double SCHEDULING_INTERVAL = 300;
```

```
	public final static double SIMULATION_LIMIT = 24 * 60 * 60;
public final static int CLOUDLET_LENGTH	= 2500 * (int) SIMULATION_LIMIT;
	public final static int CLOUDLET_PES	= 1;

	/*
	 * VM instance types:
	 *   High-Memory Extra Large Instance: 3.25 EC2 Compute Units, 8.55 GB // too much MIPS
	 *   High-CPU Medium Instance: 2.5 EC2 Compute Units, 0.85 GB
	 *   Extra Large Instance: 2 EC2 Compute Units, 3.75 GB
	 *   Small Instance: 1 EC2 Compute Unit, 1.7 GB
	 *   Micro Instance: 0.5 EC2 Compute Unit, 0.633 GB
	 *   We decrease the memory size two times to enable oversubscription
	 *
	 */
	public final static int VM_TYPES	= 4;
	public final static int[] VM_MIPS	= { 2500, 2000, 1000, 500 };
	public final static int[] VM_PES	= { 1, 1, 1, 1 };
	public final static int[] VM_RAM	= { 870,  1740, 1740, 613 };
	public final static int VM_BW		= 100000; // 100 Mbit/s
	public final static int VM_SIZE		= 2500; // 2.5 GB

	/*
	 * Host types:
	 *   HP ProLiant ML110 G4 (1 x [Xeon 3040 1860 MHz, 2 cores], 4GB)
	 *   HP ProLiant ML110 G5 (1 x [Xeon 3075 2660 MHz, 2 cores], 4GB)
	 *   We increase the memory size to enable over-subscription (x4)
	 */
	public final static int HOST_TYPES	= 2;
	public final static int[] HOST_MIPS	= { 1860, 2660 };
	public final static int[] HOST_PES	= { 2, 2 };
```

```
        public final static int[] HOST_RAM   = { 4096, 4096 };
        public final static int HOST_BW                 = 1000000; // 1 Gbit/s
        public final static int HOST_STORAGE = 1000000; // 1 GB

        public final static PowerModel[] HOST_POWER = {
                new PowerModelSpecPowerHpProLiantMl110G4Xeon3040(),
                new PowerModelSpecPowerHpProLiantMl110G5Xeon3075()
        };

}
```

PM UTILIZATION MODEL

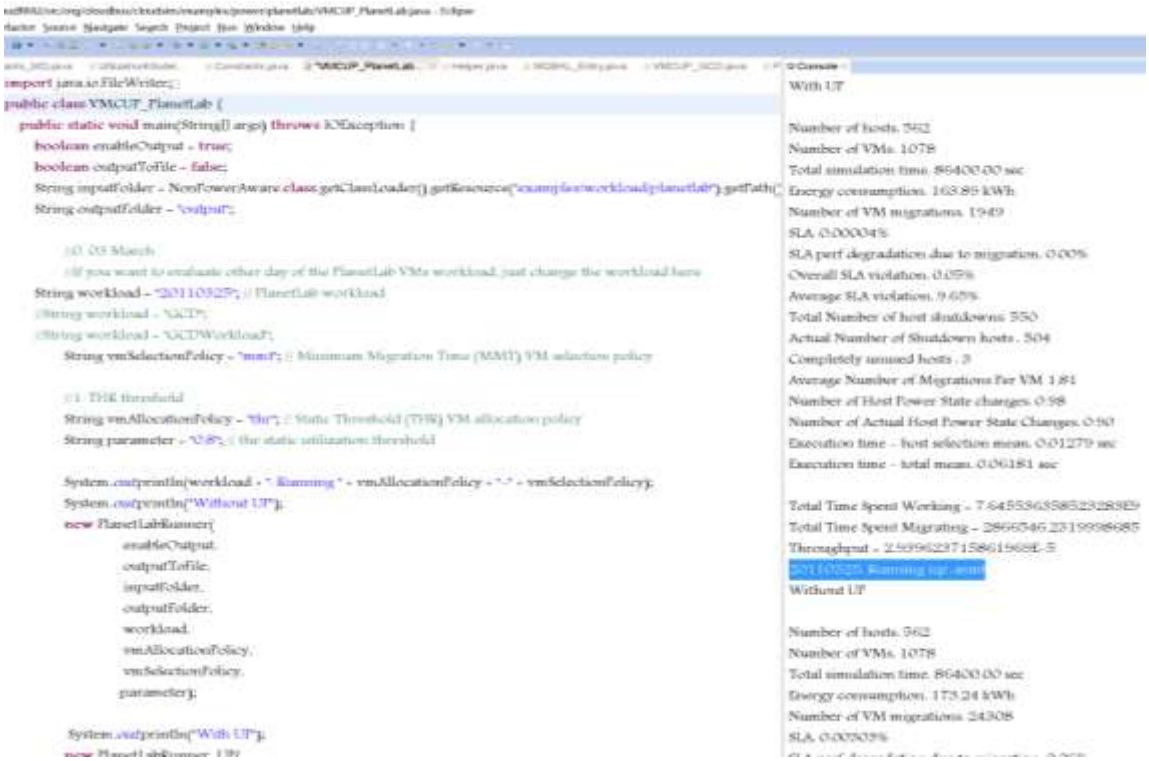
```
package org.cloudbus.cloudsim;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

/**
 * The Class UtilizationModelPlanetLab.
 */
public class UtilizationModelPlanetLabInMemory implements UtilizationModel {
        /** The scheduling interval. */
        private double schedulingInterval;
        /** The data (5 min * 288 = 24 hours). */
        private final double[] data;

   //For PlanetLab VMs workload (24 hours). The resource utilization of VM is measured
//every 5 minutes. Therefore, the total value in each PlanetLab VM file is 288 values.
 //So, the data variable is set to 289
public UtilizationModelPlanetLabInMemory(String inputPath, double schedulingInterval)
```

```
                throws NumberFormatException,
                IOException {
        data = new double[289];
        setSchedulingInterval(schedulingInterval);
        BufferedReader input = new BufferedReader(new FileReader(inputPath));
        int n = data.length;
        for (int i = 0; i < n - 1; i++) {
                data[i] = Double.valueOf(input.readLine())/100.0;
        }
        data[n - 1] = data[n - 2];
        input.close();
    }
//For Google Cluster Data VMs workload (6 hours). The resource utilization of VM is measured every 5 minutes.
//Therefore, the total value in each PlanetLab VM file is 72 values.
//So, the data variable is set to 72

public UtilizationModelPlanetLabInMemory(String inputPath, double schedulingInterval)
throws NumberFormatException, IOException {
    data = new double[73];
    setSchedulingInterval(schedulingInterval);
    BufferedReader input = new BufferedReader(new FileReader(inputPath));
    int n = data.length;
    String x =  input.readLine();
    for (int i = 0; i < n - 1; i++) {
        if (x == null)
                data[i] = 0.0;
                data[i] = Double.valueOf(x)/1000.0;
                data[i] = Integer.valueOf(input.readLine()) / 100.0;
    }
    data[n - 1] = data[n - 2];
```

```
        input.close();
}

        /**
         * FOR NEW GCD = 1600 VMs added 18.04.2017  to compare with Hieu2017
         */
         //For Google Cluster Data jobs workload with 2 resources [0] is CPU and [1] is MEM
   public UtilizationModelPlanetLabInMemory(String inputPath, double schedulingInterval)
        throws NumberFormatException, IOException {
                data = new double[289];
        setSchedulingInterval(schedulingInterval);
        BufferedReader input = new BufferedReader(new FileReader(inputPath));
        int n = data.length;
        for (int i = 0; i < n - 1; i++) {
           String line = input.readLine();
           String[] arrLine = line.split(" ");
           //data[i] = Integer.valueOf(arrLine[0]) / 100.0;
           data[i] = Double.valueOf(arrLine[0]) / 100.0;
        }
        data[n - 1] = data[n - 2];
        input.close();
}

        @Override
        public double getUtilization(double time) {
                if (time % getSchedulingInterval() == 0) {
                        return data[(int) time / (int) getSchedulingInterval()];
                }
                int time1 = (int) Math.floor(time / getSchedulingInterval());
                int time2 = (int) Math.ceil(time / getSchedulingInterval());
```

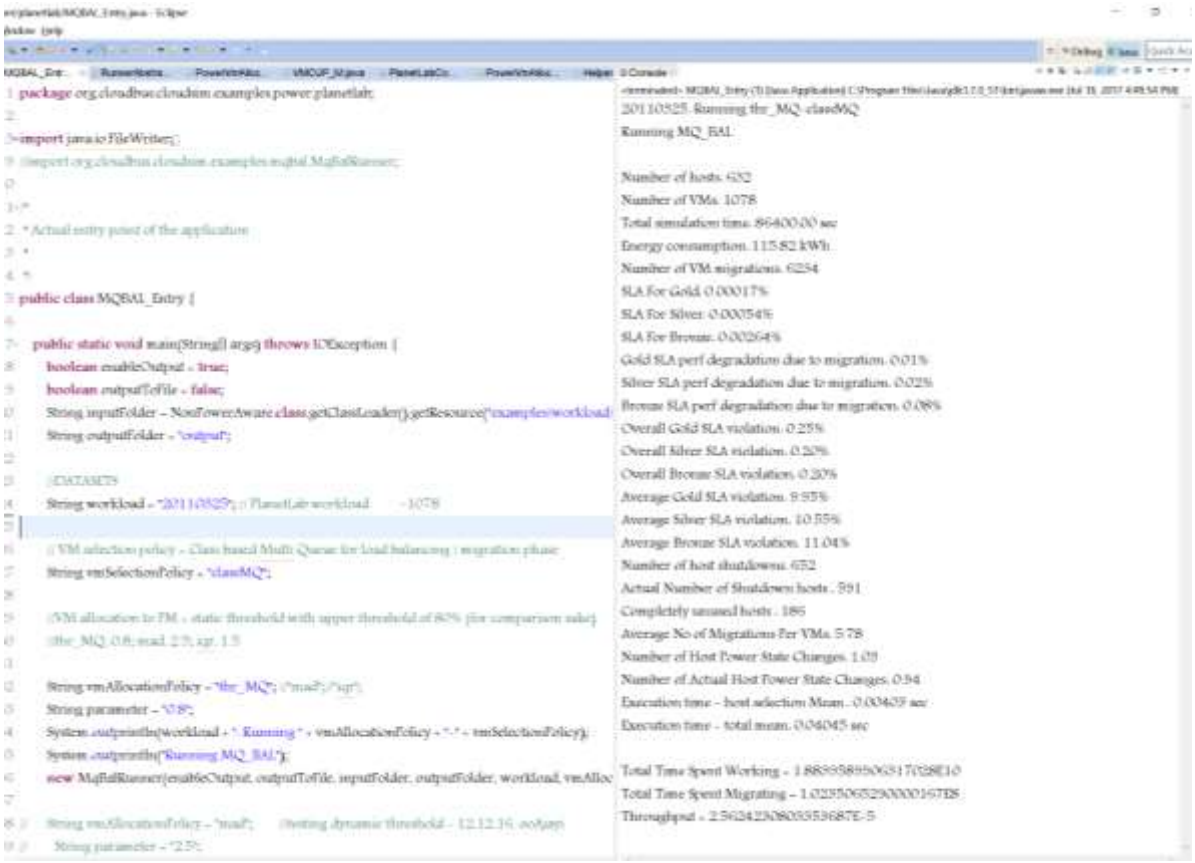
```
                double utilization1 = data[time1];
                double utilization2 = data[time2];
                double delta = (utilization2 - utilization1) / ((time2 - time1) *
getSchedulingInterval());
                double utilization = utilization1 + delta * (time - time1 *
getSchedulingInterval());
                return utilization;

        }

        /**
         * Sets the scheduling interval.
         *
         * @param schedulingInterval the new scheduling interval
         */
        public void setSchedulingInterval(double schedulingInterval) {
                this.schedulingInterval = schedulingInterval;
        }

        /**
         * Gets the scheduling interval.
         *
         * @return the scheduling interval
         */
        public double getSchedulingInterval() {
                return schedulingInterval;
        }
}
```

APPENDIX II



```
import java.io.FileWriter;
public class VMCUP_PlanetLab {
  public static void main(String[] args) throws IOException {
    boolean enableOutput = true;
    boolean outputToFile = false;
    String inputFolder = NonPowerAware.class.getClassLoader().getResource("examples/workload/planetlab").getPath();
    String outputFolder = "output";

        // 03 March
        //If you want to evaluate other day of the PlanetLab VMs workload, just change the workload here
    String workload = "20110303"; // PlanetLab workload
    //String workload = "GCD";
    //String workload = "GCDWorkload";
        String vmSelectionPolicy = "mmt"; // Minimum Migration Time (MMT) VM selection policy

        // THR threshold
        String vmAllocationPolicy = "thr"; // Static Threshold (THR) VM allocation policy
        String parameter = "0.8"; // the static utilization threshold

        System.out.println("workload = " + workload + " - vmAllocationPolicy = " + vmSelectionPolicy);
        System.out.println("Without UP");
        new PlanetLabRunner(
                enableOutput,
                outputToFile,
                inputFolder,
                outputFolder,
                workload,
                vmAllocationPolicy,
                vmSelectionPolicy,
                parameter);

        System.out.println("With UP");
```

```
With UP

Number of hosts: 552
Number of VMs: 1078
Total simulation time: 86400.00 sec
Energy consumption: 163.89 kWh
Number of VM migrations: 1949
SLA: 0.00004%
SLA perf degradation due to migration: 0.00%
Overall SLA violation: 0.05%
Average SLA violation: 9.65%
Total Number of host shutdowns: 550
Actual Number of Shutdown hosts: 504
Completely unused hosts: 0
Average Number of Migrations Per VM: 1.81
Number of Host Power State changes: 0.98
Number of Actual Host Power State Changes: 0.50
Execution time - host selection mean: 0.01279 sec
Execution time - total mean: 0.06181 sec

Total Time Spent Working - 7.645553635852328E9
Total Time Spent Migrating - 2866546.2319998685
Throughput - 2.5959237158615688E-5
20110525: Running sqr_mmt
Without UP

Number of hosts: 552
Number of VMs: 1078
Total simulation time: 86400.00 sec
Energy consumption: 175.24 kWh
Number of VM migrations: 24308
SLA: 0.00503%
```



```
1 package org.cloudbus.cloudsim.examples.power.planetlab;
2
3 import java.io.FileWriter;
9 //import org.cloudbus.cloudsim.examples.mqbal.MqBalRunner;
0
1 /**
2  * Actual entry point of the application
3  *
4  *
5 public class MQBAL_Entry {
6
7     public static void main(String[] args) throws IOException {
8         boolean enableOutput = true;
9         boolean outputToFile = false;
0         String inputFolder = NonPowerAware.class.getClassLoader().getResource("examples/workload
1         String outputFolder = "output";
2
3         //DATASETS
4         String workload = "20110525"; // PlanetLab workload       - 1078
5
6         // VM selection policy - Class based Multi Queue for load balancing + migration phase
7         String vmSelectionPolicy = "classMQ";
8
9         //VM allocation to PM - static threshold with upper threshold of 80% (for comparison sake)
0         //thr_MQ 0.8, mad 2.5, iqr 1.5
1
2         String vmAllocationPolicy = "thr_MQ"; //"mad";//"iqr";
3         String parameter = "0.8";
4         System.out.println(workload + " Running " + vmAllocationPolicy + "-" + vmSelectionPolicy);
5         System.out.println("Running MQ_BAL");
6         new MqBalRunner(enableOutput, outputToFile, inputFolder, outputFolder, workload, vmAllo
7
8 //        String vmAllocationPolicy = "mad";      //testing dynamic threshold - 12.12.16, 10.45am
9 //         String parameter = "2.5";
```

```
20110525: Running thr_MQ_classMQ
Running MQ_BAL

Number of hosts: 552
Number of VMs: 1078
Total simulation time: 86400.00 sec
Energy consumption: 115.82 kWh
Number of VM migrations: 6254
SLA For Gold: 0.00017%
SLA For Silver: 0.00054%
SLA For Bronze: 0.00264%
Gold SLA perf degradation due to migration: 0.01%
Silver SLA perf degradation due to migration: 0.02%
Bronze SLA perf degradation due to migration: 0.08%
Overall Gold SLA violation: 0.25%
Overall Silver SLA violation: 0.20%
Overall Bronze SLA violation: 0.20%
Average Gold SLA violation: 9.95%
Average Silver SLA violation: 10.55%
Average Bronze SLA violation: 11.04%
Number of host shutdowns: 652
Actual Number of Shutdown hosts: 591
Completely unused hosts: 186
Average No of Migrations Per VMs: 5.78
Number of Host Power State Changes: 1.03
Number of Actual Host Power State Changes: 0.94
Execution time - host selection Mean: 0.03403 sec
Execution time - total mean: 0.04045 sec

Total Time Spent Working - 1.8889989506317028E10
Total Time Spent Migrating - 1.0235065290000167E8
Throughput - 2.5924230803553687E-5
```

APPENDIX III
PUBLICATIONS FROM THE THESIS

Ajayi O. and Oladeji F. (2015). An Overview of Resource Management Challenges in Cloud Computing. *Book of Proc. for the 10th Unilag Research Conf. & Fair*, 2(3), (pp. 554-560).

Ajayi O., Oladeji F. and Uwadia C. (2015). Analysis of Two-Phased Approaches to Load Balancing in Cloud Computing. *Journal of Computer Science and its Applications*, 22(2), pp. 123-131.

Ajayi O., Oladeji F. and Uwadia C. (2016). Multi-Class Load Balancing for QoS and Energy Conservation in Cloud Computing. *West African Journal of Industrial and Academic Research*, 17, pp. 28-36.

Ajayi O., Oladeji F. and Uwadia C. (2017). An Energy-Aware Class-Based Load Balancing Model for Resource Management in Cloud Computing. *FUW Trends in Science and Technology Journal*, 2(1), pp. 295-300.