Energy-Efficient Management for Dynamic Consolidation of Virtual Machines in Cloud Data Centers under QoS Constraints

Loiy Alsbatin

Submitted to the
Institute of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Computer Engineering

Eastern Mediterranean University
January 2019
Gazimağusa, North Cyprus

Approval of the Institute of Graduate Studies and Research

Assoc. Prof. Dr. Ali Hakan Ulusoy
Acting Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Doctor of Philosophy in Computer Engineering.

Prof. Dr. Işık Aybay
Chair, Department of Computer Engineering

We certify that we have read this thesis and that in our opinion it is fully adequate in scope and quality as a thesis for the degree of Doctor of Philosophy in Computer Engineering.

Assoc. Prof. Dr. Ali Hakan Ulusoy
Co-Supervisor

Assoc. Prof. Dr. Gürcü Öz
Supervisor

Examining Committee

1. Prof. Dr. Işık Aybay
2. Prof. Dr. Mehmet Ufuk Çağlayan
3. Prof. Dr. Turhan Tunalı
4. Assoc. Prof. Dr. Alexander Chefranov
5. Assoc. Prof. Dr. Gürcü Öz

ABSTRACT

Dynamic Virtual Machine (VM) consolidation effectively reduces energy consumption and improves the utilization of resources in data centers. Reallocating VMs from an overloaded Physical Machine (PM) maximizes the utilization and energy efficiency with providing the required Quality of Service (QoS). The goal of consolidation of VMs ensuring an efficient utilization can be achieved through the use of VMs migration across different PMs. We present an overview of energy efficient design of cloud virtualized data centers to guide future design and development efforts. New VM placement algorithms are proposed and compared with a number of known VM placement algorithms. The algorithms are evaluated through simulations with real-world workload traces from more than a thousand VMs running on PMs located in more than 500 places around the world and it is shown that the proposed algorithms outperform the known algorithms. Moreover, a PM overload detection algorithm and a combination of PM overload detection algorithm and VM quiescing are proposed to maximize the time until migration, while meeting QoS goals. A number of benchmark PM overload detection algorithms are implemented using different parameters to compare with the proposed PM overload detection algorithm. We evaluate this algorithm through simulations with real world workload traces and results show that the proposed algorithm outperforms the benchmark PM overload detection algorithms under QoS constraints.

Keywords: Cloud computing, dynamic consolidation, energy efficiency, virtualization

ÖZ

Dinamik Sanal Makine (VM) konsolidasyonu, enerji tüketimini etkili bir şekilde azaltır ve veri merkezlerindeki kaynakların kullanımını iyileştirir. VM'lerin aşırı yüklü bir Fiziksel Makineden (PM) yeniden ayrılması, yüksek Hizmet Kalitesi (QoS) sağlayarak kullanımını ve enerji verimliliğini en üst düzeye çıkarır. VM'lerin konsolidasyon amacı verimli bir kullanım sağlamak olup farklı PM'ler arasında VM'lerin geçişi ile sağlanabilir. Gelecekteki tasarım ve geliştirme çalışmalarına rehberlik etmek için sanallaştırılmış bulut veri merkezlerinin verimli enerji tasarımına genel bir bakış sunuyoruz. VM yerleştirme algoritmaları önerilmiş ve bir dizi VM yerleştirme algoritması ile karşılaştırılmıştır. Algoritmalar, gerçek dünyadaki iş yükü izlemeleri kullanılarak benzetimler aracılığıyla değerlendirilmiş ve önerilen algoritmaların bilinen algoritmalardan daha iyi performans gösterdiği gösterilmiştir. Ayrıca, QoS hedefi karşılanırken, geçişe kadar geçen süreyi en üst düzeye çıkarmak için, PM aşırı yük algılama algoritması ve VM sıralaması ile birlikte kullanılan PM aşırı yük algılama algoritması önerilmiştir. Önerilen PM aşırı yük algılama algoritması ile karşılaştırmak için farklı parametreler kullanılarak bir dizi PM aşırı yük algılama algoritması uygulanmıştır. Algoritmaları gerçek dünyadaki iş yükü izlemeleri kullanılarak benzetimler aracılığıyla değerlendirdik ve sonuçlar, önerilen algoritmanın QoS kısıtlamaları altında PM aşırı yük algılama algoritmalarını geride bıraktığını gösterdi.

Anahtar Kelimeler: Bulut bilişimi, dinamik konsolidasyon, enerji verimliliği, sanallaştırma

DEDICATION

To My Family

ACKNOWLEDGMENT

PhD teaches me a lot, and I am glad that I have completed my PhD thesis. I would not be able to complete it without all who helped me along my study. I would like to record my gratitude to my supervisors, Assoc. Prof. Dr. Gürcü Öz and Assoc. Prof. Dr. Ali Hakan Ulusoy who have given me invaluable advice, guidance, constant encouragement and support in various ways.

I would like to acknowledge the members of my graduate committee for their advice and guidance for all their advice and encouragement. I am grateful in every possible way. My thanks go to my friends such as Saed Qaraleh, Anas Ibrahim, Aiman Afaneh, Firas Zawaideh, Tariq Fatafteh, Qusai Salameh, Amjad Taha, and Raed Basbous for their friendship, help during the period of my studies and my PhD.

I would like to thank my parents, brothers, and sisters for their encouragement and support along my study. Finally, I would like to heartily thank my wife Tasnim, who supported and inspired me, and filled my heart with love, joy and happiness.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

BFD	Best-Fit Decreasing
CO_2	Carbon Dioxide
CPBFD	CPU Priority based Best-Fit Decreasing
DCPBFD	Dynamic CPU Priority based Best-Fit Decreasing
DVFS	Dynamic Voltage and Frequency Scaling
DVS	Dynamic Voltage Scaling
E	Energy
ESV	Energy and Service Level Agreement violation
FFD	First-Fit Decreasing
HPC	High Performance Computing
IaaS	Infrastructure as a Service
IQR	Interquartile Range
LLC	Limited Lookahead Control
LR	Local Regression
LRR	Local Regression Robust
MAD	Median Absolute Deviation
MAOTF	Maximum Allowed Overload Time Fraction
MHOD	Markov Host Overload Detection
MIPS	Millions Instructions Per Second
MMT	Minimum Migration Time
NAS	Network Attached Storage
OTF	Overload Time Fraction
OTFT	Overload Time Fraction Threshold

PABFD	Power-Aware Best-Fit Decreasing
PDM	Performance Degradation due to Migrations
POTFT	Percentage of Overload Time Fraction Threshold
POTFT_Q	Percentage of Overload Time Fraction Threshold with VM Quiescing
PM	Physical Machine
QoS	Quality of Service
RAM	Random Access Memory
SLA	Service Level Agreement
SLAV	Service Level Agreement violation
SLATAH	Service Level Agreement violation Time per Active Host/ Physical Machine
SM	Simple Method
THR	Threshold based algorithm
VM	Virtual Machine
VMM	Virtual Machine Monitor

Chapter 1

INTRODUCTION

Power consumption of USA data centers has increased by 62.5% from 2005 to 2013 and expected to increase by 150% in 2020 [1]. Most of the energy consumption of data centers is consumed by the computing resources. Accordingly, it is important to invest on resource management ensuring that the applications efficiently utilize the available computing resources. Switching the idle nodes to sleep mode to eliminate idle power consumption can achieve a reduction in energy consumption. Dynamic Virtual Machine (VM) consolidation effectively reduces energy consumption and improves the utilization of resources in data centers. Reallocating VMs from an overloaded Physical Machine (PM) maximizes the utilization and energy efficiency while providing Quality of Service (QoS) requirements. The goal of consolidation of VMs ensuring an efficient utilization can be achieved through the use of VMs migration across different PMs.

The designers of computing systems have been interested on the improvement of the system performance. The performance per watt ratio is increased, but the total energy consumed by resources is hardly decreasing. If this trend continues, the energy consumption cost of PMs in their lifetime will be more than the cost of hardware [2]. Impact of energy efficiency on end-users is usually determined by the total resource usage cost of a resource provider. Results of high energy consumption increase not only electricity bills but also additional requirements for power delivery

infrastructure and cooling system. The cooling problem becomes crucial with the increasing density of computer components; thus, heat dissipation should be improved [3].

One efficient way to improve the utilization of cloud data center resources is the dynamic consolidation of VMs [4-13]. The dynamic consolidation reallocates VMs periodically using migration to reduce the number of active PMs required to handle requests. The objective of this approach is mainly to minimize energy consumption and maximize of QoS provided by the system.

It is complex to solve dynamic VM consolidation problem analytically as a whole [5, 6]. In general, the problem can be decomposed into tasks as following [9]:

1. PM underload detection: This is the phase when a PM is considered as being underloaded. So, all VMs running on an underloaded PM should be migrated to other PMs and the underloaded PM should be switched to the sleep mode (to reduce the number of active PMs).
2. PM overload detection: This is the phase when a PM is considered as being overloaded. So, some VMs running on an overloaded PM should be migrated to another active PM (to avoid violation QoS requirements).
3. VM selection: This is the phase to select VMs to be migrated from the overloaded PM.
4. VM migration: This is the phase to perform VM migration with minimal service downtime during the migration process.
5. VM placement: This is the phase to place selected VMs for migration onto another active PM.

In this study, we present solutions to energy-efficient dynamic VM consolidation for data centers while meeting QoS requirements. The efficiency of solutions are evaluated through simulations with real world workload traces.

1.1 Research Challenges

The following research challenges are investigated by this thesis:

- When to migrate VMs. In dynamic consolidation of VMs, there are two main operations: (1) migrating VMs from underloaded PMs to minimize energy consumption by the system and improve the resource utilization; and (2) migrating VMs from overloaded PMs to meet QoS requirements and avoid performance degradation. The time to migrate VMs should be determined efficiently in both situations to minimize energy consumption and improve the resource utilization while meeting QoS requirements.
- Which VMs to migrate. Selecting one or more VMs from a PM are required when a decision is made to migrate VMs to other PMs. To provide the most useful system reconfiguration, the subset of VMs to migrate should be determined efficiently.
- Where to migrate selected VMs for migration. Determining the best placement of selected VMs for migration to other PMs is necessary to improve energy consumption and the quality of VM consolidation.
- When and which PMs to switch off/on. Dynamic power switching of PMs should be combined with VM consolidation to eliminate power consumption in the idle PM. It is required to determine when and which PMs should be reactivated to avoid performance degradation by handling the increasing in the demand for resources, or deactivated to improve energy consumption.

- Explore and give an overview of the existing researches for energy-efficient cloud data centers to guide future design and development efforts.
- Develop algorithms for energy-efficient dynamic VM consolidation for cloud environments with meeting QoS constraints.

1.2 Contributions

We propose an efficient and scalable approach to manage the energy-performance tradeoff. The contributions of the thesis are:

1- A survey study of energy-efficient cloud data centers.

2- An approach for energy-efficient dynamic consolidation of VMs using proposed VM placement algorithms, which leads to a significant improvement in performance by maximizing utilization while meeting QoS requirements and a significant reduction in energy consumption using real workload traces.

3- A PM overload detection algorithm and combination of PM overload detection algorithm and VM quiescing that effectively improve mitigating overload while meeting QoS requirements.

1.3 Thesis Organization

The rest of the thesis is organized as follows:

- Chapter 2 presents literature review of energy-efficient resource management systems and VM consolidation for cloud data centers.
- Chapter 3 proposes PM placement algorithms for consolidation of VMs.
- Chapter 4 proposes a PM overload detection algorithm and combination of PM overload detection algorithm and VM quiescing for consolidation of VMs.
- Chapter 5 concludes the thesis and discusses future works.

1.4 Publications

The publications related to the thesis are:

- Loiy Alsbatin, Gürcü Öz, and Ali Hakan Ulusoy, "An overview of energy-efficient cloud data centres," in Proceedings of the International Conference of computer and applications, Dubai, United Arab Emirates, 6-7 September 2017, pp. 211- 214.
- Loiy Alsbatin, Gürcü Öz, Ali Hakan Ulusoy, "A Novel Physical Machine Overload Detection Algorithm Combined with Queiscing for Dynamic Virtual Machine Consolidation in Cloud Data Centers," The International Arab Journal of Information Technology (Accepted on 18 September 2018).
- Loiy Alsbatin, Gürcü Öz, Ali Hakan Ulusoy, "Efficient Virtual Machine Placement Algorithms for Consolidation in Cloud Data Centers," in review.

Chapter 2

LITERATURE RIVIEW OF ENERGY-EFFICIENT RESOURCE MANAGEMENT, DYNAMIC VM CONSOLIDATION, AND VM QUIESCING FOR CLOUD DATA CENTERS

2.1 Introduction

The designers of computing systems are interested on the improvements of computing performance that are driven by the demand of consumer, business, and scientific applications. However, growth of computing performance has been limited due to the increasing of energy consumption of cloud data centers as a result of carbon dioxide (CO_2) footprints and high electricity bills [3]. Therefore, the designers of computing systems have started research to improve energy efficiency. In this chapter, we present an overview of energy efficient design of cloud virtualized data centers and show the recent level of development to guide future design and development efforts.

The rest of the chapter is organized as follows. In Section 2.2, we introduce the literature review of energy-efficient resource management for cloud data centers. In Section 2.3, we introduce the literature review of dynamic consolidation of VMs for cloud data centers.

2.2 Literature Review of Energy-Efficient Resource Management for Cloud Data Centers

Most energy-efficient resource management approaches for cloud computing aim to improve consolidation of the workload into a minimum number of PMs. Idle resources can be switched off, which reduces energy consumption, and increases resource utilization. However, the consolidation has to meet the Service-Level Agreement (SLA) requirements, as well as minimize the performance degradation and energy consumption. In this section, we give an overview of different approaches of efficient management of tradeoff between performance and energy in virtualized data centers. Table 2.1 shows the most important reviewed research works, which aim to minimize energy with meeting performance requirements.

Table 2.1: Cloud data centers research works

Authors	**Resources**	**Techniques**
Nathuji and Schwan [14]	CPU	Soft scaling, switching off PM, VM consolidation, DFVS
Raghavendra et al. [15]	CPU	Switching off PM, VM consolidation, DVFS
Verma et al. [5]	CPU	Switching off PM, VM consolidation, DVFS
Kusic et al. [16]	CPU	Switching off PM, VM consolidation
Song et al. [17]	RAM, CPU	Resource throttling
Stillwell et al. [18]	CPU	VM consolidation, resource throttling
Cardosa et al. [19]	CPU	Soft scaling, DFVS
Gmach et al. [7]	RAM, CPU	VM consolidation, switching off PM
Kumar et al. [20]	RAM, CPU, Network	DVFS, VM consolidation
Buyya et al. [23]	CPU	Leveraging heterogeneity, DVFS
Beloglazov and Buyya [9, 25]	CPU, RAM	Switching off PM, dynamic VM consolidation, DVFS
Horri et al. [26]	CPU, RAM	Switching off PM, Dynamic VM consolidation
Fu and Zhou [8]	CPU, RAM	Switching off PM, Dynamic VM consolidation
Arianyan et al. [27]	CPU, RAM	Switching off PM, Dynamic VM consolidation
Han Et al [10]	CPU, RAM	Switching off PM, Dynamic VM consolidation

An approach for power-efficient resource management in data centers has been proposed for the first time in the context of virtualized systems by Nathuji and Schwan [14], applying a new power management technique called “soft resource scaling". Soft resource scaling uses VM manager’s scheduling ability to provide less time for utilizing the resources to a VM. “Soft” scaling is suitable when hardware scaling provides a little decrease in energy or is not supported. It has been found that the combination of “soft” and “hard” scaling improves energy consumption because of the typically limited number of hardware scaling states.

An approach for power management for a data center by combination of different power management strategies has been proposed by Raghavendra et al. [15]. The authors find that it is more likely to apply different solutions from multiple vendors even though all aspects of power management can be handled by implementing a centralized solution. The existing solutions are classified by a number of attributes, such as software/hardware, local/global policies, the objective function, and performance constraints. The authors focus on individual solutions instead of solving the whole problem. They have applied a feedback control loop to coordinate the controllers’ actions. However, the proposed approach is unable to guarantee that the system meets QoS requirements. Thus, the approach is not suitable for cloud computing providers, where a comprehensive support for SLA and more reliable QoS are essential, but for enterprise environments.

The problem of dynamic VM placement has been solved by a heuristic bin packing algorithm by Verma et al. [5]. The pMapper application placement framework has been proposed to minimize the power consumption and maintain SLA. It consists of an arbitrator, performance manager, power manager and migration manager. The

authors claim that the proposed framework is general enough to combine different performance and power management strategies under SLA's restrictions. The problem has been formulated as a continuous improvement. VM placement should be improved at each time frame to maximize the performance and minimize the power consumption. Moreover, the solution can be used to any type of the workload. However, SLA cannot be met because of unforeseeable workloads and instability. The authors have defined a bin packing algorithm using variable costs and bin sizes to solve the placement problem. A proposed algorithm and the pMapper architecture have been implemented with performing extensive experiments to evaluate the system efficiency. The authors have suggested numerous future works such as more advanced applications of idle states, consideration of memory bandwidth, and extending the theoretical formulation of the problem.

An approach for performance and power efficient resource management in virtualized computing systems has been proposed by Kusic et al. [16]. SLA for each application in the computing system is defined as the request processing rate. The goal is to raise the profit of resource providers by reducing SLA violation and power consumption. A sequential optimization using Limited Lookahead Control (LLC) is used to solve the problem. To improve the performance, a method based on neural networks have been applied. However, only 10 hosts are used in the experiments, which is not sufficient to demonstrate the applicability of an approach.

An approach for the efficient resource allocation in multi-application virtualized data centers has been proposed by Song et al. [17]. The goal is to reduce energy consumption by improving the resource utilization. According to the application priorities, the resources are allocated to applications using several VMs instantiated

on different PMs. Only Random Access Memory (RAM) and CPU utilizations are taken into consideration in the decisions. A drop in the performance occurs for the low-priority applications in cases of limited resources, since critical applications are only used the resources. VMs are preinstantiated on a set of PMs, and they are assigned by fractions of the total resources. The limitation of the proposed approach is that the migration of VMs is not used for adjusting the allocation at runtime. Moreover, machine learning is required to obtain the resource utilization functions. Such approach is appropriate for an enterprise environment, where the priorities of applications can be clearly defined.

The resource allocation problem for High Performance Computing (HPC) applications in virtualized homogeneous clusters have been studied by Stillwell et al. [18]. The goal is to maximize the utilization of resources by improving user-centric metric that is denoted as the yield, which is part of the maximum achievable computing rate. A Mixed Integer Programming Model has been proposed to define problem and to solve small instances of the problem, but not to solve large-scale problem instances. It is assumed that the requirement of resources is known in advance, which is not practical. Moreover, CPU is only considered in resource management decisions.

An approach for the power-efficient VM allocation in virtualized enterprise computing environments has been proposed by Cardosa et al. [19], which uses min, max, and shares parameters. Min and max parameters specify minimum and maximum allowed amount of CPU resources allocated to a VM. Shares parameter sets percentages, where CPU resources will be allocated to each VM that share the same resource. This approach is suitable for enterprise environments, where a

comprehensive support for SLA and the priorities of applications are not essential. The proposed algorithm enhances the ability to reduce the resources required by a VM to the minimum and expand it when there are available spare resources to bring additional profit. The limitation of the proposed approach is that the migration of VMs is not used for adjusting the allocation at runtime. Another problem is that CPU is only considered by the model in resource management decisions. Moreover, clear definition of the application priorities are required for the proposed approach, which makes this approach not applicable in real-world environments.

An approach for the energy-efficient dynamic VM consolidation for enterprise systems has been proposed by Gmach et al. [7], combining a migration controller and a workload placement controller has been proposed. The workload placement controller improves the allocation under QoS requirements using historical resource usage information collected from VMs in the data center. Multiobjective improvement is performed to find a new VM placement, which minimizes the required VM migrations and the average number of PMs in the system. The migration controller is run in parallel to solve the underloading and overloading of PMs.

Kumar et al. [20] have investigated the problem for dynamic consolidation of VMs using probabilistic estimation of the effective reallocation of VM in the future. Several strategies are implemented for integrated VM placement considering CPU, memory, network and power budget requirements. A time-varying probability density function is used to predict future resource demands of applications. The distribution parameters including the mean and standard deviation are assumed to be known in advance. The resource utilization is assumed to follow a normal

distribution, however many researches [21, 22] showed that the resource usage of an application is usually not simple to be modelled using normal distributions. The experiments showed that power consumption is reduced by 10%, VM migrations by 54%, and SLA violations by 71% compared to the benchmark system.

The GreenCloud project that aimed at developing of energy-efficient provisioning of cloud resources under QoS constraints has been proposed by Buyya et al. [23]. A real-time VM model has been introduced. Three policies have been proposed using Dynamic Voltage and Frequency Scaling (DVFS) to improve the energy efficiency and maximize the request acceptance rate, while meeting QoS constraints. The policies are the lowest Dynamic Voltage Scaling (DVS) policy, the d-advanced-DVS policy, and the adaptive-DVS policy. The approach has been evaluated using the CloudSim toolkit [24].

Belaglazov and Buyya [9] have proposed energy-efficient dynamic consolidation of VMs applied in virtualized data centers containing heterogeneous PMs. The authors focus on Infrastructure as a Service (IaaS) cloud environments. The proposed resource management system is able to handle arbitrary workloads. A workload independent QoS metric is used to determine QoS requirements in SLAs to constrain acceptable performance degradation and the degree of VM consolidation. To improve dynamic VM consolidation it is necessary to oversubscribe CPU to achieve higher levels of utilization. However, higher levels may lead to performance degradation of applications. In additional, Belaglazov and Buyya [25] have proposed an architecture of a framework for dynamic consolidation of VM. The authors have proposed a benchmark suite to evaluate dynamic VM consolidation algorithms using real-world workload traces.

A QoS-aware VMs consolidation approach has been proposed by Horri et al. [26] for cloud environments, which adopts a method based on resource utilization history of VMs. The authors have evaluated proposed algorithms using CloudSim simulator. Experimental evaluations show that there is improvement in energy consumption and QoS metrics. In addition, it proves a tradeoff between QoS and energy consumption in the cloud environment.

VM placement and selection for dynamic consolidation in cloud computing environment has been proposed by Fu and Zhou [8]. VM selection policy not only considers CPU utilization, but also defines a variable that represents the degree of resource satisfaction to select VMs. Moreover, authors have proposed a novel VM placement policy based on selecting PM with minimum correlation coefficient to be placed by a migratable VM. Results show that the proposed policies perform better than existing policies according to SLA violation percentage, VM migration time, and energy consumption.

Arianyan et al. [27] have proposed as an effective VM consolidation to save energy in cloud data centers. The authors have proposed a new holistic cloud resource management procedure as well as novel heuristics based on multi-criteria decision making method to solve underloaded PMs and VM placement problems. The results of simulations using Cloudsim simulator validates the applicability of the proposed policies, which shows up reductions in the number of VM migrations, SLA violation, and energy consumption in comparison with benchmark works.

Han et al. [10] have proposed a VM placement algorithm, and an algorithm to find underloaded PMs to switch them to sleep mode for energy saving. The authors have

applied a combination of these two algorithms for cloud data centers to complete the process of VM consolidation. Simulation results have shown that there exists a tradeoff between SLA violations and the energy consumption of cloud data centers. Moreover, VM placement algorithm is able to deal with variable workload to prevent PMs from overloading after VM placement and to reduce SLA violations dramatically.

2.3 Literature Review of Dynamic VM Consolidation

2.3.1 Literature Review of VM Placement

In the past few years, many approaches to the dynamic consolidation of VMs have been proposed [4-13]. Comparative studies of various existing consolidation of VM algorithms using real-world workload traces were presented. A scheduling algorithm to assign VMs to PMs in a data center was proposed by Mohen Sharifi et al. in [28]. The goal was to improve energy efficiency by taking into consideration the conflicts between the costs of VM migration and CPU and disk utilizations. Four models were presented to identify the conflicts, namely the migration model, the energy model, the application model, and the target system model.

PM overload and underload detection algorithms and VM placement algorithm based on dynamic thresholds and probable future load were proposed by Shaw et al. in [29]. They used simple exponential smoothing technique to predict CPU utilization and calculate dynamic upper and lower utilization thresholds. A VM consolidation algorithm with utilization prediction of multiple resource types based on the local history of PMs was proposed by Nguyen et al. in [30] to improve the energy efficiency of cloud data centers. Two adaptive energy-aware algorithms for minimizing SLA violation and maximizing energy efficiency in cloud data centers

were proposed by Zhou et al. in [31]. CPU, memory resources and application types were taken into account during the deployment of VMs.

Kakadia et al. in [32] proposed a greedy consolidation algorithm based on VMs placement algorithm to improve the network usage and performance of applications in the data centers. The greedy consolidation algorithm reduced the number of migrations and sped up the placement decisions. Mattias et al. in [33] proposed two algorithms, which could be used together for live migration of multiple VMs. The proposed VM migration algorithms depended on three factors that were the cost of migration, the expected distribution of workload and the state of PM after migration. The algorithms distributed the workload efficiently in the system. In spite of that, the paper did not discuss how to meet SLA. In [5], the problem of dynamic VM placement was solved by a heuristic bin packing algorithm. However, SLA cannot be met because of unforeseeable workloads and instability.

Gupta et al. [34] has proposed a VM placement algorithm used a resource usage model, which leads to maximize CPU utilization of the PMs and reduce the number of underloaded PMs. The algorithm evaluated using Amazon EC2 Instances. Halacsy and Mann [35] have proposed a modified First-Fit Decreasing heuristic to solve VM placement problem. The approach aims to reduce power consumption in data centers. Bin packing with heterogeneous bin types has been used, where the cost of a bin depends on its load. To optimize VM consolidation in the cloud data centers, Yousefipour et al. [36] have proposed a genetic algorithm to solve VM placement problem based on meta-heuristic algorithm. The approach aims to minimize the number of active PMs, which leads to minimize energy consumption and maximize CPU utilization.

A dynamic consolidation of VMs for web applications was implemented by Guenter et al. [11]. In this study, the response time was used to define SLA. Weighted linear regression was applied to get the future workload and improve the distribution of workload.VM consolidation algorithms under QoS expectations were evaluated using the CloudSim toolkit showing high improvement of cost savings and energy efficiency using dynamic workload scenarios [9, 37]. Authors proposed maximum correlation, random selection and Minimum Migration Time (MMT) policies for VM selection from the overloaded PM and utilized interquartile range, median absolute deviation, robust local regression and Local Regression (LR) algorithms for PM overload detection. The Simple Method (SM) was used to find underloaded PM which was with the least resource utilization. For VM placement, they proposed Power-Aware Best-Fit Decreasing (PABFD) algorithm, which based on sorting VMs by CPU utilization in decreasing order and placing a VM in PM that will have the minimum expected increase in power consumption. The results showed that the combination of LR for PM overload detection and MMT for VM selection had better performance in the number of VM migrations, energy consumption, and SLA violations.

In this thesis, algorithms named as CPU Priority based Best-Fit Decreasing (CPBFD) and Dynamic CPU Priority based Best-Fit Decreasing (DCPBFD) are proposed for VM placement. LR method was used for PM overload detection, SM policy was used for PM underload detection, MMT policy was used for VM selection. For comparison purposes with the proposed CPBFD and DCPBFD VM placement algorithms we used well-known VM placement algorithms used in most recent research for comparison purposes [8, 10, 34, 35, 36, 40], which are PABFD, First-

Fit Decreasing (FFD) [38-40] and Best-Fit Decreasing (BFD) [39,40] algorithms which are well-known algorithms for bin-packing problem. VM placement algorithm based on FFD sorts VMs by CPU utilization in decreasing order and places a VM in the first PM that will fit it [40]. VM placement algorithm based on BFD sorts VMs by CPU utilization in decreasing order and places a VM in PM that will have the maximum CPU utilization after allocating VM [40].

2.3.2 Literature Review of PM Overload Detection and VM Quiescing

Some of VM consolidation algorithms based on different heuristics on the legitimate PM were analyzed in by Heena Kaushar et al. [41]. An adaptive heuristics energy-aware algorithm that used an upper threshold of CPU utilization for PM overload detection and dynamic VM selection algorithms was proposed in [42]. An adaptive threshold-based algorithm was proposed by Deng et al. in [43]. The overload threshold of CPU utilization and the average utilization of active PMs were used for PM underload detection algorithm, and minimum average utilization difference of the data center was used for VM placement algorithm. Several dynamic VM consolidation algorithms were proposed by Khoshkholghi et al. in [44] to improve the utilization, energy consumption and SLA violations based on CPU, RAM and bandwidth. They used an iterative weighted linear regression method for PM overload detection and a vector magnitude squared of resources for PM underload detection. They also proposed a SLA and power-aware VM selection algorithm and a VM placement algorithm.

Managing resource allocation to improve response time using control loops at the server and cluster levels were applied in [45]. The server migrated a VM if the server's resource capacity was not enough to meet SLA of application. In [46], two

algorithms that could be used together for live migration of multiple VMs were proposed. The proposed VM migration depended on three factors that were the cost of migration, the expected distribution of workload and the state of PM after migration. The algorithms distributed the workload efficiently in the system. In spite of that, the research did not discuss how to meet SLA. Zhou et al. [47] have proposed a comprehensive strategy for VM consolidation based on modified forecasting method. Authors have applied specific adjustment of threshold migration to improve PM overload and underload detection and VM migration. The strategy has been evaluated using the real workload trace from Google.

Baset et al. [48] have developed a framework for understanding overload in a cloud data centers. Authors have presented overload mitigation mechanisms. They have evaluated the performance of overload mitigation mechanisms, which perform live migration of VMs to mitigate overload. Because of network constraints on the number of VM migrations in a given time period, authors suggested as a future work a combination of VM quiescing (temporary turning off VM) and VM live migration to mitigate overload in cloud data centers.

In general, current solutions to PM overload detection problem are based on heuristic or statistical analysis of historical data. These approaches do not clearly specify QoS target. We propose the dynamic consolidation of VMs under the specified QoS targets, which optimally solves the problem of short time intervals between sequential VM migrations under QoS goal. The average time between sequential VM migrations is necessary to be maximized to optimize consolidation of VMs [4]. VM consolidation is used to reduce the number of active PMs, so a lower average number of active PMs represents a better consolidation of VMs.

PM overload detection can initiate a VM migration from an overloaded PM. There are two possible cases of VM migrations due to the overload: (1) a new PM must be activated for a VM, which should be migrated from an overloaded PM due to not enough resources on another active PM, and (2) a VM, which should be migrated, can be allocated to another active PM. The goal of PM overload detection is to maximize the average time between sequential VM migrations (to minimize the number of migrations) [4]. Therefore, the activity time should be maximized for overloaded PMs, while the activity time should be minimized for underloaded PMs.

At every moment of time, each VM on a PM takes a fraction of CPU utilization as required by VM's workload. PM's workload is constituted by CPU utilization created by a set of VMs that is currently placed on a PM. CPU utilization of PM is monitored by a controller. A PM overload detection algorithm decides when a VM migration needs to be done to meet QoS goals, while maximizing the average time until migration.

We mainly focus on PM overload detection problem. PM overload detection directly influences QoS, because performance degradation is very likely to occur if the resources are completely utilized. PM overload detection problem is complex because of the need to improve the system response time, while handling a set of heterogeneous workloads placed on a single PM [4]. When a PM is considered as being underloaded, its VMs are consolidated into other active PMs and it should be switched to the sleep mode. However, when a PM is considered as being overloaded, VMs require a higher performance, therefore, other PMs in sleep mode may be reactivated to migrate VMs.

Since there were constraints on the allowed number of migration of VMs in a certain period of time, prior works might not be the only effective methods. We propose a PM overload detection algorithm and combination of PM overload detection algorithm and VM quiescing, which maximize the time between sequential VM migrations (to minimize the number of migration of VMs) under the specified QoS constraint.

Chapter 3

EFFICIENT VIRTUAL MACHINE PLACEMENT ALGORITHMS FOR CONSOLIDATION IN CLOUD DATA CENTERS

3.1 Introduction

Dynamic VM consolidation is a successful approach to improve energy efficiency and resource utilization in cloud environments. Consequently, optimizing the online energy-performance tradeoff directly influences QoS. In this chapter, algorithms named as CPBFD and DCPBFD are proposed for VM placement. A number of VM placement algorithms are then implemented and compared with the proposed algorithms. The algorithms are evaluated through simulations with real-world workload traces and it is shown that the proposed algorithms outperform the known algorithms. The results show that CPBFD and DCPBFD provide the least SLA violations, least VM migrations, and an efficient energy consumption. The contribution of the Chapter is new VM placement algorithms based on well-known BFD algorithm, which are able to improve performance and energy efficiency compared with a number of known VM placement algorithms.

The rest of the chapter is organized as follows. We introduce system model in Section 3.2. The metrics used to show the performance of the algorithms are described in Section 3.3. We present the proposed VM placement algorithms in

Section 3.4. Finally, the experimental setup, evaluations and results are discussed in Section 3.5.

3.2 System Model

We use the system model presented in [9] to evaluate VM placement algorithms by using the CloudSim [25] toolkit. The system consists of *X* heterogeneous PMs in a large-scale data center. Characteristics of each PM are defined by CPU performance denoted by RAM, network bandwidth, and execution speed in Millions Instructions Per Second (MIPS). The storage of servers for VM live migration is called Network Attached Storage (NAS). Multiple independent users request for supplying *Y* VMs characterized by requirements denoted by RAM, MIPS and network bandwidth. The workloads created by independent users utilize the resources simultaneously. SLA is established by the users and the resource provider. A penalty is paid by the provider to users in cases of SLA violations.

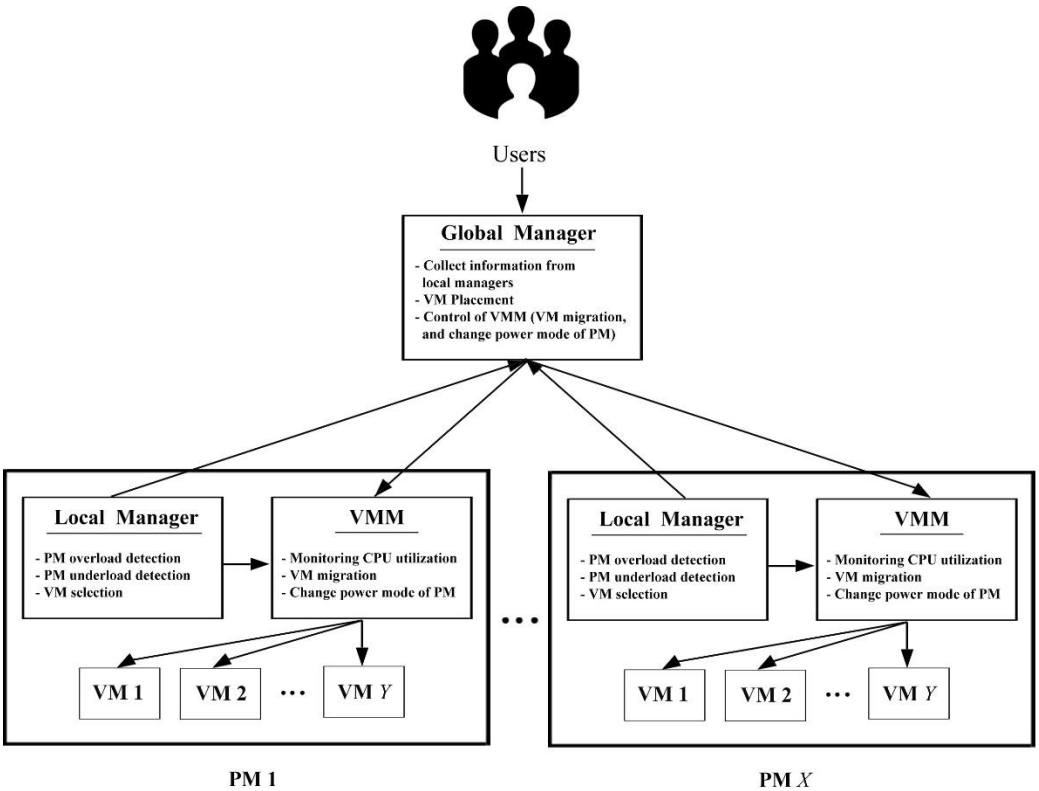

Figure 3.1: The system model [9]

As shown in Figure 3.1, the system includes global and local managers. The local manager on each PM monitors CPU utilization of PM using Virtual Machine Monitor (VMM). The local manager decides when and which VMs should be migrated to other PMs using PM overload detection, PM underload detection, and VM selection algorithms. The global manager, which acts as the controller in the system, collects information of the utilization of PMs from the local managers and decides VM placement, and how VMMs migrate VMs and change the power mode of PMs.

3.3 Metrics

3.3.1 Power Model

In the data centers, power consumption of PMs is usually defined by CPU, cooling systems, power supplies, memory, and disk storage [49]. The power consumption of PMs can be defined using linear relationship of CPU utilization even if DVFS is used [16, 37, 50]. Due to the fact that the limited number of the frequency and voltage states of a CPU and other system components, such as network interfaces and memory, the voltage and frequency scaling are not used. Since analytical models of power consumption is a complex research problem for modern multi-core CPUs [9], real power consumption benchmark results provided by the SPECpower [51] is used.

The work in [37, 52] shows that a PM when it is idle uses approximately 70% of its maximum energy consumption. As presented in [37, 52], the power consumption of PM can be defined as

$$P(u) = 0.7 \times Pmax + 0.3 \times Pmax \times u \quad (3.1)$$

where *Pmax* is the maximum power of a fully utilized PM and is set to 250 W as presented in [37, 52]. *u* is CPU utilization. Since CPU utilization changes over time,

it is presented as a function of time $u(t)$. As presented in [42], energy consumption can be obtained as

$$E = \int_t P(u(t))dt. \quad (3.2)$$

Since the energy consumption of a PM is determined by CPU utilization, CPU utilization is taken into consideration in the proposed VM placement algorithms to reduce the energy consumption in the system as presented in [8, 37, 52].

3.3.2 Cost of Live Migration of VMs

Live migration of VMs transfers VMs between PMs without suspension. However, a large number of live VM migrations may drop the performance of applications. So, the number of VM migrations should be reduced. The behavior of applications causes performance degradation. We use cost model for VM migration presented in [8, 9] to avoid performance degradation. The authors stated in [8, 9] that CPU utilization can be increased by 10% for each VM migration. So, each VM migration can cause SLA violations. Therefore, VM migrations should be reduced, and VM with minimum memory should be selected.

3.3.3 CPU Utilization

We use CPU model presented in [9]. Each PM is equipped with a multi-core CPU. A multi-core CPU with *x* cores each with capacity of *y* MIPS is modeled as a single core CPU with *xy* MIPS. Each VM allocated to a PM utilizes a part of CPU capacity according to its workload at each point in time. Therefore, CPU utilization of PM measured at discrete time steps by a set of VMs allocated to PM. Moreover, as we mentioned in Section 3.3.2, CPU utilization is also increased by 10% for each VM migration. Therefore, the increase in CPU utilization of destination PM due to VM migration equals the result of multiplying 10%, the length of VM migration, and the value of VM utilization in source PM. The length of VM migration equals the total

memory size used by VM divided by available network bandwidth for VM. Thus, for our experiments we define PM's CPU utilization in MIPS as shown in (3.3), and PM's CPU utilization as shown in (3.4). VM's CPU utilization in MIPS is calculated according to VM's CPU utilization traces.

$$U_{MIPS_i}(t) = \sum_{j=1}^{Y} u_{MIPS_j}(t) + \sum_{j \in VM_j in\ migration} 0.1 \times \frac{M_j}{B_j} \times u_{MIPS_j}(t) \quad (3.3)$$

$$U_i(t) = \begin{cases} \dfrac{U_{MIPS_i}(t)}{U_{MIPS_{i_{total}}}}, U_{MIPS_i}(t) \leq U_{MIPS_{i_{total}}} \\ 1, \qquad U_{MIPS_i}(t) > U_{MIPS_{i_{total}}} \end{cases} \quad (3.4)$$

where $U_{MIPS_i}(t)$ is CPU utilization of PM *i* in MIPS at time *t*, *Y* is number of VMs in PM *i*, $u_{MIPS_j}(t)$ is CPU utilization by VM *j* in MIPS at time *t*, $B_j$is the available network bandwidth for VM *j*, $M_j$is the memory size of VM *j*, U_i is CPU utilization of PM at time *t*, and $U_{MIPS_{i_{total}}}$is a total MIPS rating of PM.

3.3.4 SLA Violation Metrics

In cloud computing environments, it is highly important to meet QoS requirements. QoS is usually defined in the form of SLA that is defined through some characteristics such as maximum response time or minimum throughput of the system [9]. To evaluate QoS requirements, SLA metric is defined as a workload independent metric for any loads in IaaS. Two metrics are used to measure SLA violations: The fraction of time when CPU utilization of PM have been 100%, SLA violation Time per Active Host/PM (SLATAH) as shown in (3.5); and Performance Degradation due to Migrations (PDM) as shown in (3.6) [9].

$$\text{SLATAH} = \sum_{i=1}^{X} \frac{T_{s_i}}{T_{a_i}} \quad (3.5)$$

$$\text{PDM} = \frac{1}{Y} \sum_{j=1}^{Y} \frac{C_{d_j}}{C_{r_j}} \quad (3.6)$$

X represents the number of PMs, T_{s_i} is the time when the utilization of i-th PM is 100% which leads to an SLA violation, T_{a_i} is the time when i-th PM is active, Y represents the number of VMs, C_{d_j} is the estimated performance degradation caused by j-th VM migrations, C_{r_j} is the total CPU capacity requested by j-th VM. C_{d_j} is estimated to be 10% of CPU utilization in MIPS during the j-th VM migrations [9].

The level of SLA violations is independently characterized by both SLATAH and PDM metrics. So, we use a metric presented in [9] that includes performance degradation caused by both overloaded PM and VM migrations, denoted as SLA Violation (SLAV), which is calculated as:

$$\text{SLAV} = \text{SLATAH} \times \text{PDM}. \tag{3.7}$$

VM consolidation objective is to reduce both Energy Consumption (EC) and SLAV (ESV). ESV metric presented in [9] that equals the product of EC and SLA violations is used as in (3.8).

$$\text{ESV} = \text{EC} \times \text{SLAV}. \tag{3.8}$$

3.4 VM Placement Algorithms

We propose novel VM placement algorithms based on giving priority of PM with highest CPU utilization between two sided limits, then giving priority to PM with CPU utilization outside the two-sided limits and nearest to the two-sided limits. Second priority is given to PMs with CPU utilization outside the two-sided limits, since selecting PMs with low load or high load leads to less active PMs and less energy consumption than waking up PMs from sleep mode. We modified well-known BFD algorithm [39, 40] to be suitable for VM placement by limiting the upper CPU utilization threshold [17, 53, 54, 55, 56] and lower CPU utilization threshold and implementing the above-mentioned priority. We denote proposed

algorithm that uses static upper and lower CPU utilization thresholds as CPBFD, and proposed algorithm that uses dynamic upper and lower CPU utilization thresholds as DCPBFD. Not setting an upper limit for CPU utilization of allocated PMs may cause frequent overloading of allocated PMs, which leads to performance degradation and increases the number of VM migrations. We propose to limit the upper threshold of CPU utilization of allocated PM to avoid performance degradation caused by VM migrations to PM with high load and to minimize the number of VM migrations. Furthermore, not setting a lower limit for CPU utilization of allocated PMs may cause allocating underloaded PMs, which leads to more active PMs and more energy consumption. We propose to limit the lower threshold of CPU utilization of allocated PM to improve energy consumption.

3.4.1 CPBFD VM Placement Algorithm

In CPBFD, all VMs are sorted in the decreasing order of their current CPU utilizations and allocate each VM to a PM with maximum CPU utilization less than upper CPU utilization threshold *a* and more than lower CPU utilization threshold *b*. If there are no PMs with CPU utilization between upper and lower threshold, a PM with the nearest CPU utilization to upper or lower thresholds will be allocated. The priority may be given to PM with CPU utilization nearest to upper threshold or lower threshold by adjusting parameter *c*. PM with CPU utilization that nearest to *c* and outside of *a* to *b* range will be selected. How VMs are allocated to PM based on CPBFD and DCPBFD is shown in Figure 3.2.

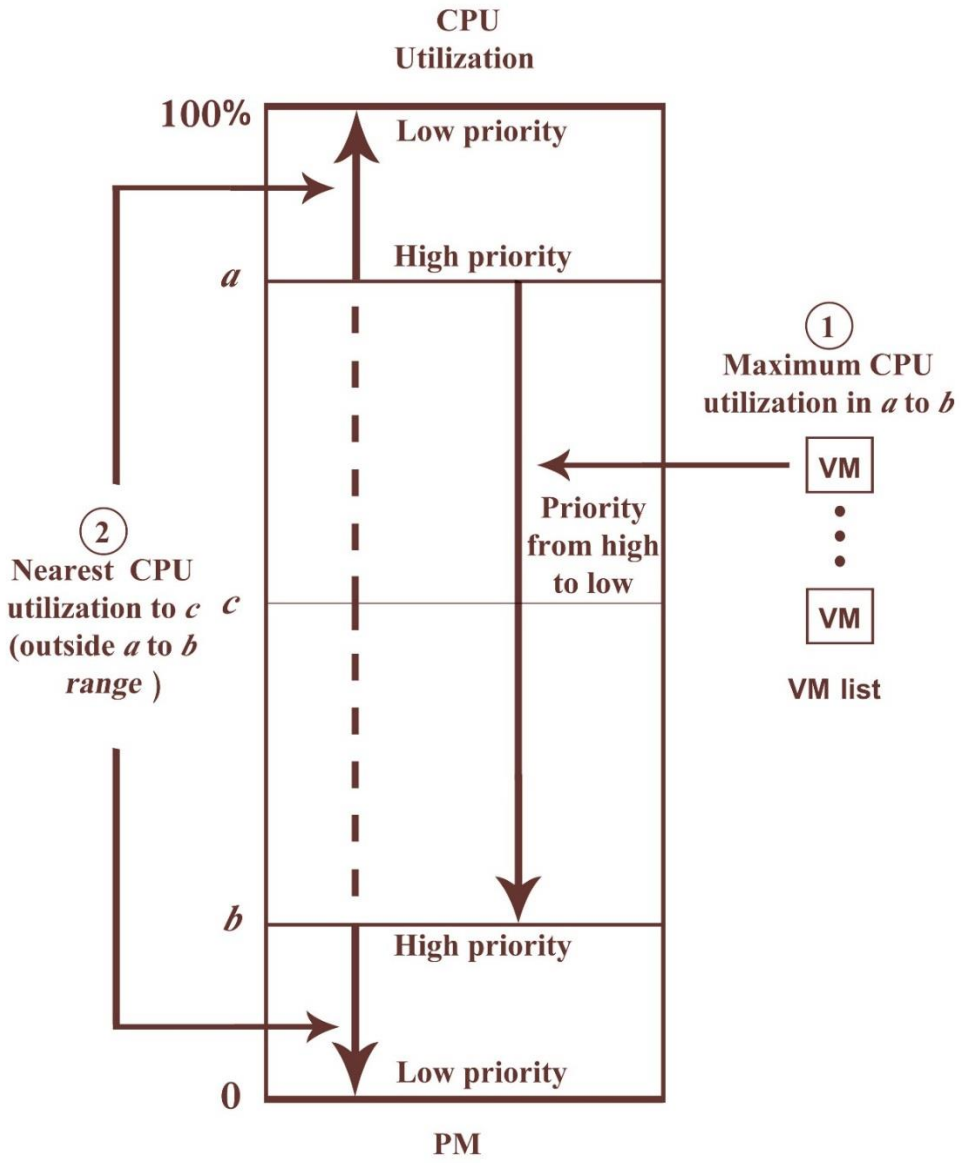

Figure 3.2: The priority of VMs placement to PM based on CPBFD and DCPBFD algorithms

There is no specific optimal upper CPU threshold value among the researchers [17, 53, 54, 55, 56]. Selecting high upper CPU threshold may significantly drop the performance of VMs running on a PM, while selecting low upper CPU threshold value makes consolidation inefficient to reduce energy consumption. Furthermore, there is no specific optimal lower CPU threshold value. So selecting the suitable value for upper and lower CPU threshold is important. However, we propose selecting dynamic optimal *a* and *b* in DCPBFD algorithm based on results of static *a*

and *b* in CPBFD algorithm. The upper and lower threshold of CPU utilization are modified according to parameter *a* and *b*, respectively. CPBFD VM placement algorithm is shown in Algorithm 3.1.

Algorithm 3.1: CPBFD VM Placement Algorithm

Input: pmList, vmList, *a*, *b*, *c*

Output: allocatedPm

```
1: vmList.sortDecreasingUtilization()
2: for each vm in vmList do
3:     maxCpu = 0
4:     minCpu = 1
5:     allocatedPm = null
6:     allocatedPm2 = null
7:     for each pm in pmList do
8:         if pm is excluded pm then
9:             continue
10:        end if
11:        if pm has enough resources for vm then
12:            if pm.Cpu ≠ 0 and pm is over utilized after allocation vm then
13:                continue
14:            end if
15:            if pm.Cpu ≤ a and pm.Cpu ≥ b then
16:                if pm.Cpu > maxCpu then
17:                    allocatedPm = pm
18:                    maxCpu = pm.Cpu
```

```
19:                 end if
20:             else if Abs(pm.Cpu - c) < minCpu then
21:                 allocatedPm2 = pm
22:                 minCpu = Abs(pm.Cpu - c)
23:                  end if
24:             end if
25:         end if
26:     end for each
27:     if allocatedPm = null then
28:         allocatedPm = allocatedPm2
29:     end if
30:     if allocatedPm ≠ null then
31:          allocation.add(vm,allocatedPm)
32:     end if
33: end for each
34: return allocatedPm
```

where pmList is the list of PMs, vmList is the list of VMs, allocatedPm is PM that is selected for VM allocation, sortDecreasingUtilization is the function that sorts VMs by CPU utilization in decreasing order, maxCpu is the variable used to select PM with maximum CPU between *a* and *b*, minCPU is the variable used to select PM with nearest CPU utilization to *c* (outside *a* to *b* range), pm.Cpu is CPU utilization of PM, Abs(pm.Cpu - *c*) is absolute value of the difference between CPU utilization of PM and *c*, which is used to select PM with nearest CPU utilization to *c*, and allocation.add is the function that is used to allocate VM on the selected PM.

3.4.2 DCPBFD VM Placement Algorithm

CPU resource utilization model of PM of DCPBFD algorithm is the same as that used in CPBFD shown in Figure 3.2. The only difference between DCPBFD and CPBFD is that upper and lower CPU utilization thresholds in DCPBFD are dynamic, but in CPBFD they are static as discussed in Section 3.4.1. In DCPBFD, all VMs are sorted in the decreasing order of their current CPU utilizations and allocate each VM to a PM with maximum CPU utilization less than dynamic upper CPU utilization threshold *a* and more than dynamic lower CPU utilization threshold *b*. If there are no PMs with CPU utilization between upper and lower thresholds, a PM with the nearest CPU utilization to upper or lower thresholds will be allocated. The priority may be given to PM with CPU utilization nearest to upper threshold or lower threshold by adjusting parameter *c*. PM with CPU utilization that is nearest to *c* and outside of *a* to *b* range will be selected.

As we mentioned in Section 3.4.1, there is no specific optimal upper and lower CPU utilization threshold values among the researchers. In DCPBFD, the proposed optimal value of dynamic upper and lower CPU utilization threshold *a* and *b* are based on fixed maximum and minimum CPU utilization for upper CPU utilization threshold *a*, fixed maximum and minimum CPU utilization for lower CPU utilization threshold *b*, Median Absolute Deviation (MAD) of historical values of PM CPU utilization, and median of historical values of PM CPU utilization divided by number of VMs allocated to PM.

The fixed maximum and minimum CPU utilization is used for upper CPU threshold *a* as maximum upper CPU threshold *a* (*maxA*) and minimum upper CPU threshold *a* (*minA*), respectively. In addition, the fixed maximum and minimum CPU utilization

is used for lower CPU threshold *b* as maximum lower CPU threshold *b* (*maxB*) and minimum lower CPU threshold *b* (*minB*), respectively. MAD is defined as the median of the absolute deviations from the median of historical values of PM CPU utilization. MAD gives an idea about CPU utilization variability, which is important to calculate suitable upper and lower CPU thresholds. Median of historical values of PM CPU utilization divided by number of VMs gives an idea about the utilization of VMs that will be allocated to PM, which is also important to calculate suitable upper and lower CPU thresholds. *a* and *b* is calculated as shown in (3.9) and (3.10), respectively.

$$a = \begin{cases} maxA - s \times \left(CpuMAD + \frac{CpuMedian}{Number\ of\ VMs}\right), \\ \left(maxA - s \times \left(CpuMAD + \frac{CpuMedian}{Number\ of\ VMs}\right)\right) \geq minA \\ minA, \left(maxA - s \times \left(CpuMAD + \frac{CpuMedian}{Number} of\ VMs\right)\right) < minA \end{cases} \quad (3.9)$$

$$b = \begin{cases} minB + s \times \left(CpuMAD + \frac{CpuMedian}{Number\ of\ VMs}\right), \\ \left(minB + s \times \left(CpuMAD + \frac{CpuMedian}{Number\ of\ VMs}\right)\right) \leq maxB \\ maxB, \left(minB + s \times \left(CpuMAD + \frac{CpuMedian}{Number\ of\ VMs}\right)\right) > maxB \end{cases} \quad (3.10)$$

where *s* is a parameter that allows the adjustment of the dynamic upper and lower CPU utilization limits, the lower *s*, the wider two sided limits and the less the energy consumption, but the higher the level of SLA violations caused by the consolidation. For the stability of CPU utilization threshold, *minA is* used to avoid that *a* reaches low inappropriate value, *maxA* is used to avoid that *a* reaches high inappropriate value, *minB is* used to avoid that *b* reaches low inappropriate value, and *maxB* is used to avoid that *b* reaches high inappropriate value.

DCPBFD VM placement algorithm is shown in Algorithm 3.2.

Algorithm 3.2: DCPBFD VM Placement Algorithm

Input: pmList, vmList, *maxA*, *minA*, *maxB*, *minB*, *c*, *s*

Output: allocatedPm

1: vmList.sortDecreasingUtilization()

2: **for each** vm in vmList **do**

3: maxCpu = 0

4: minCpu = 1

5: allocatedPm = null

6: allocatedPm2 = null

7: **for** each pm in pmList **do**

8: **if** pm is excluded pm **then**

9: continue

10: **end if**

11: **if** pm has enough resources for vm **then**

12: **if** pm.Cpu $\neq$ 0 and pm is over utilized after allocation vm **then**

13: continue

14: **end if**

15: **if** (pm.CPUHistory.length $\geq$ 12) **then**

16: $a = maxA - s \times$ (pm.CPUHistoryMAD + pm.CPUHistoryMedian / #ofvm)

17: $b = minB + s \times$ (pm.CPUHistoryMAD + pm.CPUHistoryMedian / #ofvm)

18: **else**

```
19:                 a = maxA - 0.05
20:                 b = minB + 0.05
21:             end if
22:             if a < minA
23:                 a = minA
24:             end if
25:             if b > maxB
26:                 b = maxB
27:             end if
28:             if pm.Cpu ≤ a and pm.Cpu ≥ b then
29:                 if pm.Cpu > maxCpu then
30:                     allocatedPm = pm
31:                     maxCpu = pm.Cpu
32:                 end if
33:             else if Abs(pm.Cpu - c) < minCpu then
34:                 allocatedPm2 = pm
35:                 minCpu = Abs(pm.Cpu - c)
36:                 end if
37:             end if
38:         end if
39:     end for each
40:     if allocatedPm = null then
41:         allocatedPm = allocatedPm2
42:     end if
43:     if allocatedPm ≠ null then
```

44: allocation.add(vm,allocatedPm)

45: **end if**

46: **end for each**

47: **return** allocatedPm

where pm.CPUHistory.length is the number of historical values of PM's CPU utilization, pm.CPUHistoryMAD is MAD of historical values of PM's CPU utilization, pm.CPUHistoryMedian is the median of historical values of PM's CPU utilization, and #ofvm is the number of VMs.

Calculation of MAD starts when at least 12 historical values of CPU utilization of PM are obtained. 12 is used as a safe value to calculate MAD [9]. We suggest to have the initial values of *a* and *b* before obtaining the first MAD value. The initial value of *a* is adjusted to (*maxA* – 5%), and the initial value of *b* is adjusted to (*minB* + 5%), which are supposed to be appropriate to avoid that *a* reaches *maxA* and *b* reaches *minB*, respectively. The values of *maxA*, *minA*, *maxB* and *minB* are selected based on CPBFD results.

3.5 Performance Evaluation

3.5.1 Experiment Setup

A cloud computing user accesses infinite computing resources. Large scale and repeatable experiments which are necessary to analyze and compare the algorithms is very difficult on a real-world infrastructure [9]. We use simulations for ensuring the repeatability of experiments. We use the CloudSim toolkit [24, 57] as a simulation platform that allows the energy consumption modeling on cloud computing environments. Appendix A presents the installation and running of the CloudSim

toolkit. Appendix B presents the code of calculation CPU utilization in MIPS. Appendix C presents the source code of CPBFD and DCPBFD algorithms. Appendix P presents the CloudSim architecture.

As presented in [9], we simulate a data center containing 800 heterogeneous PMs. PMs are 400 HP ProLiant ML110 Generation 4 (G4) and 400 HP ProLiant ML110 Generation 5 (G5). CPU frequencies of the servers are mapped onto MIPS rating: 1,860 MIPS for each core of HP ProLiant ML110 G4 server, and 2,660 MIPS for each core of HP ProLiant ML110 G5 server. Each server is modeled to have 1 GB/s network bandwidth. VM characteristics correspond to Amazon EC2 instance types including Extra Large Instance (3.75 GB, 2,000 MIPS); High-CPU Medium Instance (0.85 GB, 2,500 MIPS); Micro Instance (613 MB, 500 MIPS); and Small Instance (1.7 GB, 1,000 MIPS). Initialization of VMs allocation is done according to the resource requirements of VM types. However, VM's useless resources during the lifetime according to the workload create opportunities for dynamic consolidation. Appendix D presents code of reading from input files, creating VMs list, creating PMs list, VM instance types and PM types for CloudSim toolkit.

3.5.2 Workload Data

To make the evaluation of simulation applicable, we use real-world workload traces provided as a monitoring infrastructure for PlanetLab [58], which is a part of the CoMon project. We use CPU utilization traces presented in [9] from more than a thousand VMs running on PMs located in more than 500 places around the world. CPU utilization values are collected every 5 minutes. Random 10 days from the collected workload traces are used. The workload characteristics for each day are presented in Table 3.1. In the simulations, each VM is randomly assigned a workload

trace from one of VMs from the corresponding day. VM consolidation is not limited by the memory bounds to avoid the constraint on the consolidation. Appendix E presents some CPU utilization of workload traces for CloudSim toolkit

Table 3.1: Workload data characteristics [9]

Workload	Number of VMs	Mean of CPU utilization	Standard deviation of CPU utilization	Median of CPU utilization
1	1052	12.31%	17.09%	6%
2	898	11.44%	16.83%	5%
3	1061	10.70%	15.57%	4%
4	1516	9.26%	12.78%	5%
5	1078	10.56%	14.14%	6%
6	1463	12.39%	16.55%	6%
7	1358	11.12%	15.09%	6%
8	1233	11.56%	15.07%	6%
9	1054	11.54%	15.15%	6%
10	1033	10.43%	15.21%	4%

3.5.3 Simulation Results

The algorithms are evaluated using the CloudSim and the workload traces are presented in Section 3.5.2. We compare proposed CPBFD algorithm to FFD, BFD and PABFD. We simulate all combinations of SM underload detection algorithm, LR overloading detection algorithm, MMT VM selection policy and four VM placement algorithms (FFD, BFD, PABFD and proposed CPBFD). Algorithms that are used in dynamic VM consolidation problem are shown in Figure 3.3. SM underload detection algorithm, and LR overloading detection algorithm are periodically invoked for every PM. In the case of underload detection, all VMs running on an underloaded PM will be allocated to other PMs using one of VM placement algorithms and the underloaded PM will be switched to the sleep mode. In the case of overload detection, MMT VM selection policy will be invoked to select VMs to

be migrated from the overloaded PM and the selected VMs will be allocated to other PMs using one of VM placement algorithms.

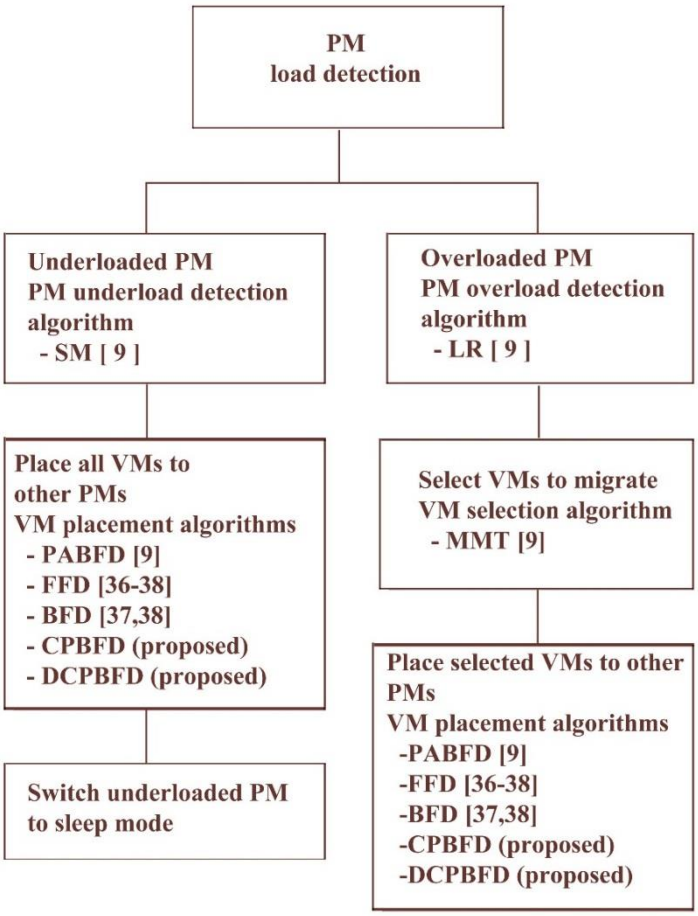

Figure 3.3: Dynamic VM consolidation problem and algorithms in cloud data centers

For the proposed CPBFD algorithm, two-sided limits (*a* and *b*) of CPU utilization is varied from wider to narrower based as 90% to 10%, 80% to 20%, 70% to 30% and 60% to 40%. Moreover, *c* parameter is varied to give more priority for low CPU utilization, equal priority to low or high CPU utilization, and more priority for high CPU utilization as 0.45, 0.50, and 0.55, respectively. According to these variations, combinations of *a*, *b* and *c* parameters are varied as (0.9, 0.1, 0.45), (0.9, 0.1, 0.5), (0.9, 0.1, 0.55), (0.8, 0.2, 0.45), (0.8, 0.2, 0.5), (0.8, 0.2, 0.55), (0.7, 0.3, 0.45) (0.7, 0.3, 0.5), (0.7, 0.3, 0.55), (0.6, 0.4, 0.45) (0.6, 0.4, 0.5), and (0.6, 0.4, 0.55). The

purposes of these variations are to get better upper and lower CPU utilization threshold and better priority for low or high CPU utilization for the proposed CPBFD algorithm and to use these better parameters in proposed DCPBFD algorithm. Appendix F presents simulation process steps for VM placement algorithms (FFD, BFD, PABFD, CPBFD and DCPBFD). In addition, Appendix G presents the results of VM placement algorithms for all workloads.

Figures 3.4 to 3.9 show the average results with 95% confidence intervals of ESV metric, energy consumption, SLAV metric, number of migrations, PDM metric, and SLATAH metric for all algorithms combination in 10 workload cases, respectively.

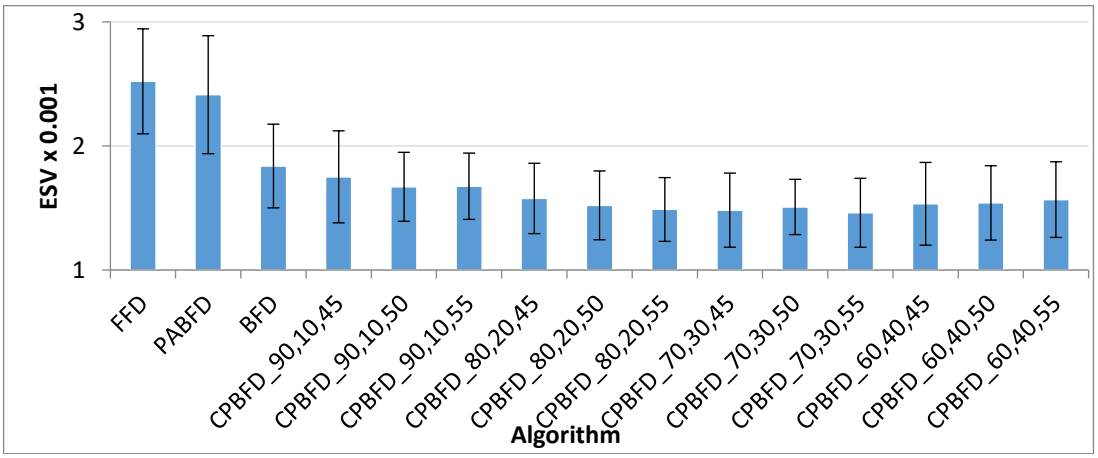

Figure 3.4: ESV metric of VM placement algorithms

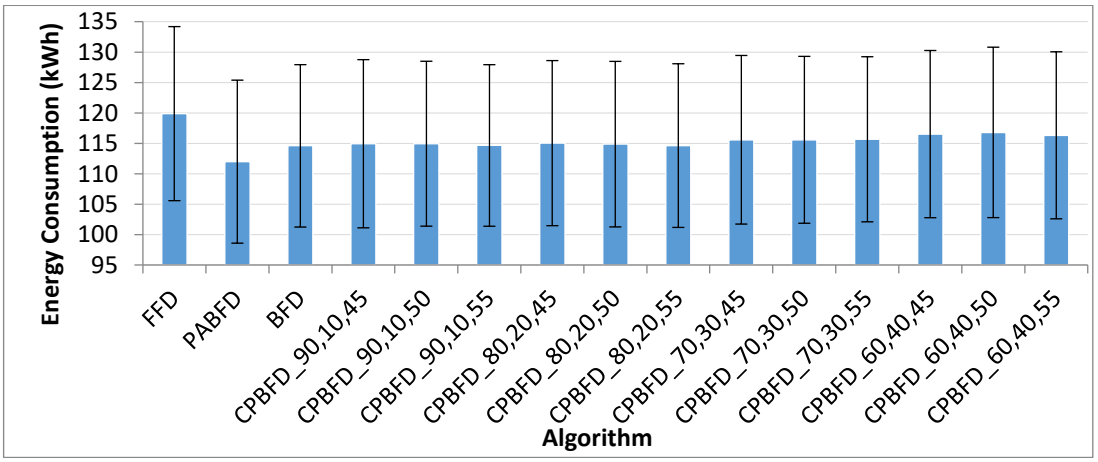

Figure 3.5: The energy consumption of VM placement algorithms

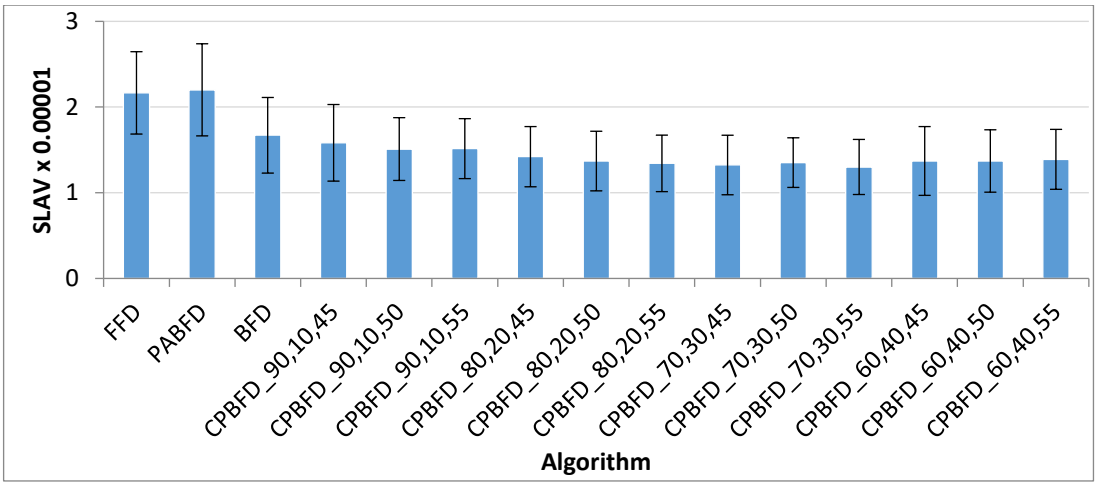

Figure 3.6: SLAV metric of VM placement algorithms

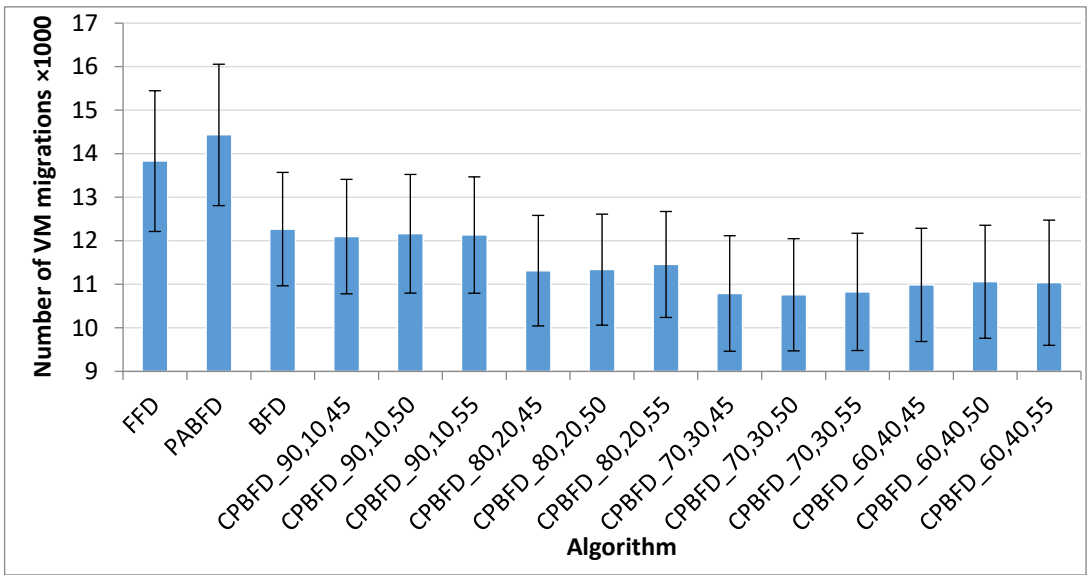

Figure 3.7: Number of VM migrations of VM placement algorithms

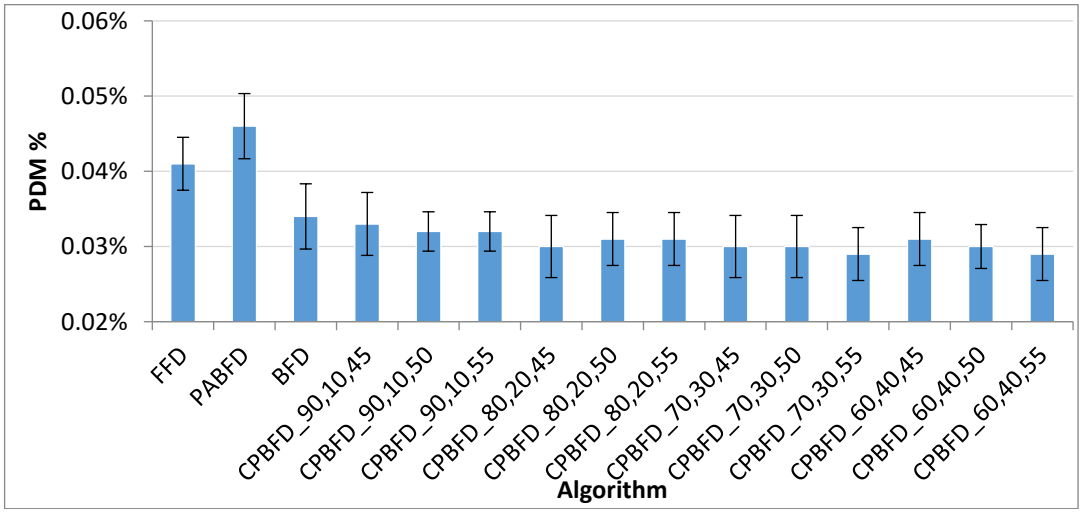

Figure 3.8: PDM metric of VM placement algorithms

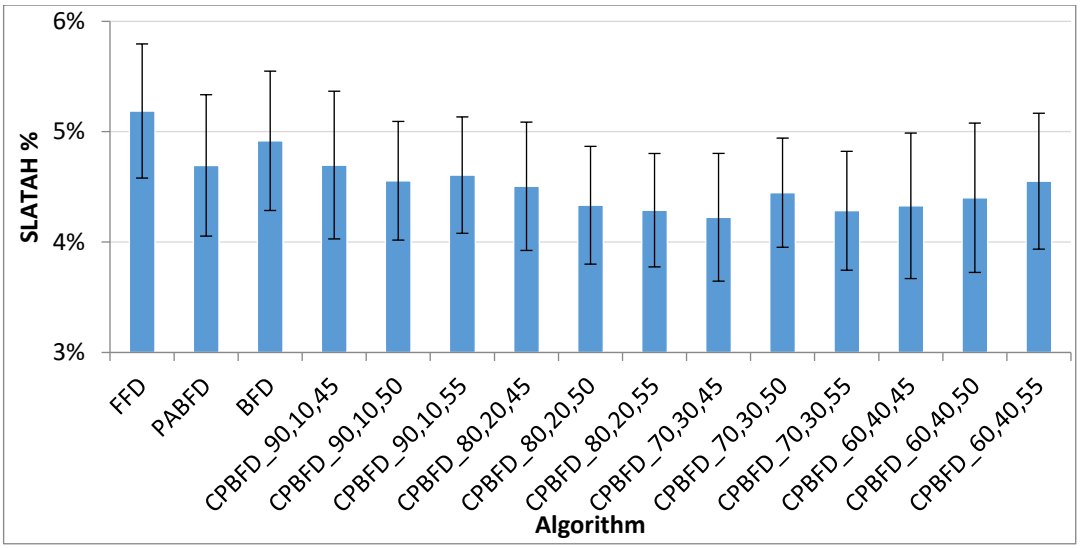

Figure 3.9: SLATAH metric of VM placement algorithms

Results in Figures 3.4 to 3.9 show that CPBFD leads to better results in regard to all parameters compared to FFD. CPBFD leads to better of ESV metric, SLAV metric, number of migrations, PDM metric, and SLATAH metric compared to BFD and PABFD. Moreover, BFD leads to better results in regard to ESV metric, SLAV metric, number of migrations, PDM metric compared to FFD and PABFD. On the other hand, PABFD only leads to the least energy consumption compared to all algorithms. Figure 3.4 shows that CPBFD has better ESV on average approximately 61.1%, 54.2% and 17.5% than FFD, PABFD, and BFD, respectively. Figure 3.5 shows that CPBFD has less energy consumption on average approximately 3.8% than FFD and more energy consumption on average approximately 0.75%, and 3% than BFD, and PABFD, respectively. Figure 3.6 shows that CPBFD has less SLAV on average approximately 54.2%, 56.7% and 19% than FFD, PABFD, and BFD, respectively. Figure 3.7 shows that CPBFD has fewer number of migration on average approximately 22.1%, 27.4%, and 8.3% than FFD, PABFD, and BFD, respectively. Figure 3.8 shows that CPBFD has less PDM on average approximately 35.9%, 52.5% and 12.7% than FFD, PABFD, and BFD, respectively. Figure 3.9

shows that CPBFD has less SLATAH on average approximately 19.3%, 8% and 13.1% than FFD, PABFD, and BFD, respectively.

CPBFD_70,30,55 has better ESV and SLAV on average approximately 7.7% and 8.7% than CPBFD with other parameters. CPBFD_80,20,55 has better efficient energy consumption on average approximately 0.8% than CPBFD with other parameters. CPBFD_80,20,55 has almost the same energy consumed by BFD, and more energy consumption on average approximately 2.3% than PABFD. CPBFD with (a = 70, b = 30) of CPU has better ESV on average approximately 14.5%, 3% and 4.3% than CPBFD with (a = 90, b = 10), CPBFD with (a = 80, b = 20), and CPBFD with (a = 60, b = 40), respectively. CPBFD with (a = 70, b = 30) of CPU has better SLAV on average approximately 15.9%, 3.9% and 3.9% than CPBFD with (a = 90, b = 10), CPBFD with (a = 80, b = 20), and CPBFD with (a = 60, b = 40), respectively. CPBFD with (a = 80, b = 20) has almost the same energy consumed by CPBFD with (a = 90, b = 10), and better efficient energy consumption on average approximately 0.7%, and 1.5% than CPBFD with (a = 70, b = 30), and CPBFD with (a = 60, b = 40).

The energy consumption changes slightly compared to SLA violation. Therefore, the impact of SLA violation on ESV metric is greater than energy consumption. However, the suitable value of a, b and c should be selected to make a tradeoff between meeting QoS and improving energy efficiency. From the simulation results, we observe that CPBFD with (a = 70, b = 30) provides best ESV metric, SLA violations and number of migrations. In addition, CPBFD with (a = 80, b = 20) provides best efficient energy consumption. Moreover, CPBFD when c parameter equals 0.55 leads to a little better ESV metric, SLA violations and energy

consumption. Furthermore, we observe that selecting moderate two-sided limits of CPU utilization between (a = 70, b = 30) and (a = 80, b = 20) for CPBFD is better than selecting too wide (a = 90, b = 10) or too narrow (a = 60, b = 40) sided limits of CPU utilization.

For the proposed DCPBFD algorithm, the maximum higher CPU utilization limit (*maxA*) is set to 80%, the minimum higher CPU utilization limit (*minA*) is set to 60%, the maximum lower CPU utilization limit (*max*B) is set to 40%, and the minimum lower CPU utilization limit (*min*B) is set to 20%, which are suitable moderate values according to the results obtained from CPBFD algorithm. *s* parameter of two-sided limits (*a* and *b*) of CPU utilization is varied as 1, 0.75, and 0.5. Moreover, *c* parameter is set to 0.55 to give more priority for high CPU utilization, which gives the best result according to the results obtained from CPBFD algorithm. According to these variations, combinations of *maxA*, *minA*, *maxB* and *minB*, *c* and *s* parameters are varied as (0.8, 0.6, 0.4, 0.2, 0.55, 1), (0.8, 0.6, 0.4, 0.2, 0.55, 0.75), and (0.8, 0.6, 0.4, 0.2, 0.55, 0.5).

Figures 3.10 to 3.15 show the average results with 95% confidence intervals of ESV metric, energy consumption, SLAV metric, number of migrations, PDM metric, and SLATAH metric for DCPBFD algorithm with all combination of parameters, and CPBFD_80,20,55, CPBFD_75,25,55, and CPBFD_70,30,55 that have the best results compared to CPBFD with other parameters.

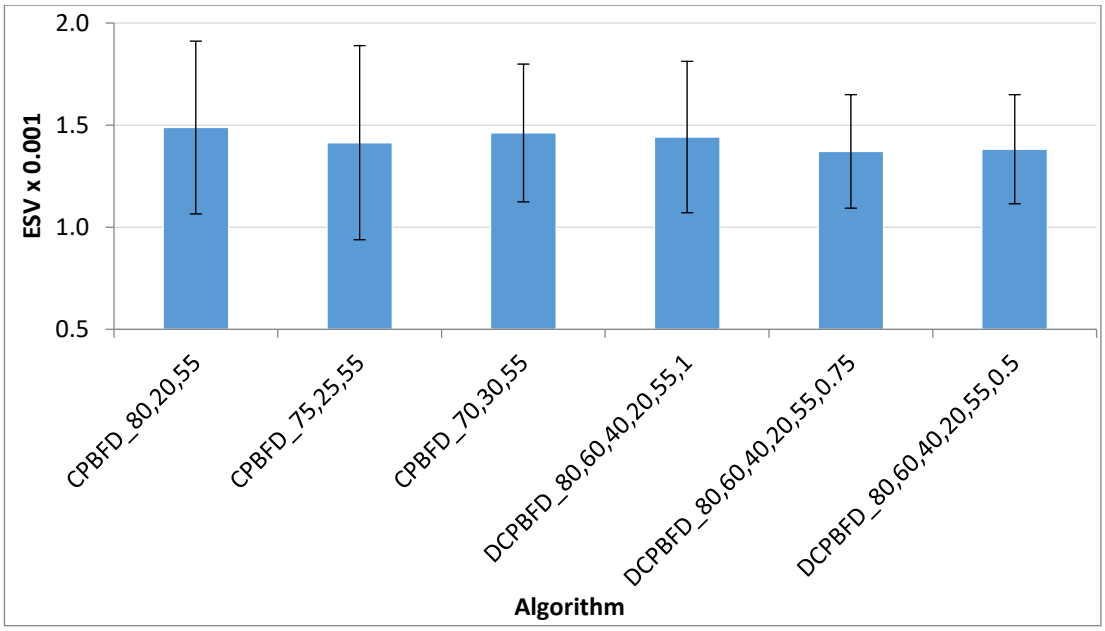

Figure 3.10: ESV metric of CPBFD and DCPBFD algorithms

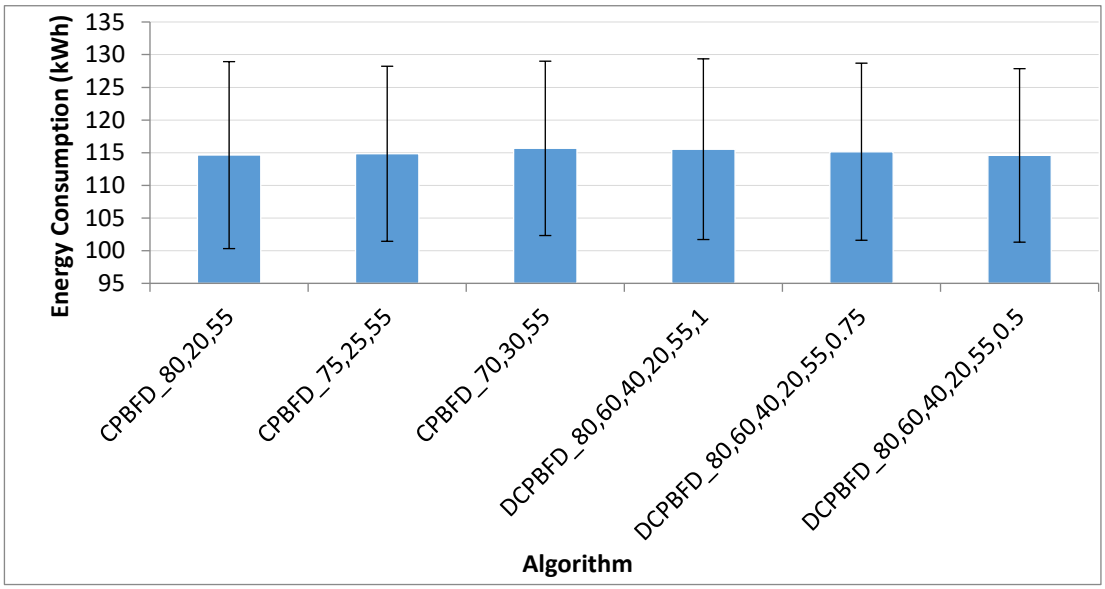

Figure 3.11: The energy consumption of CPBFD and DCPBFD algorithms

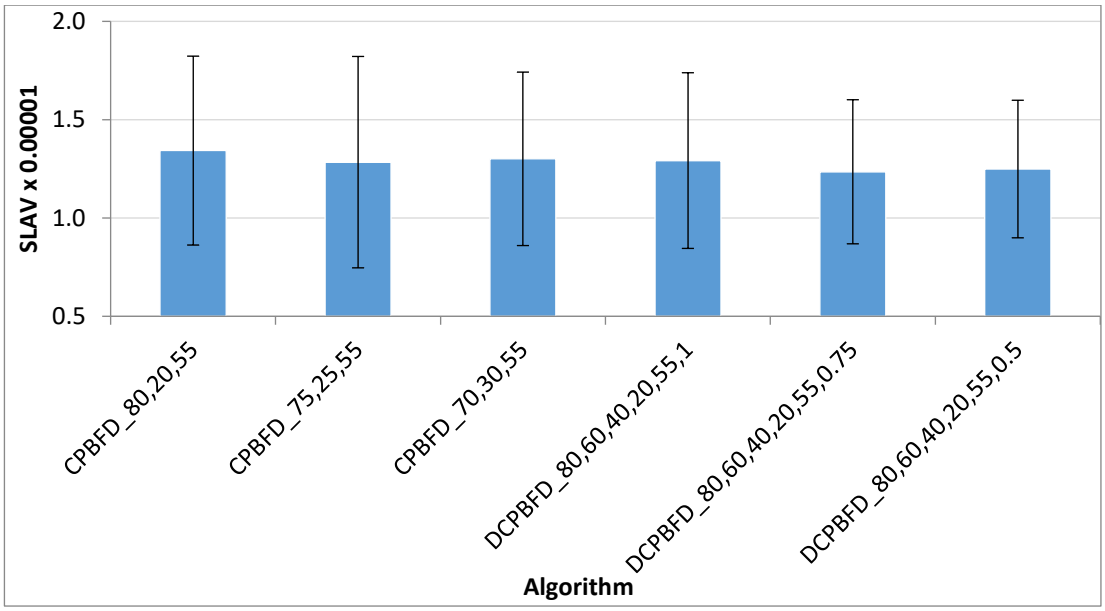

Figure 3.12: SLAV metric of CPBFD and DCPBFD algorithms

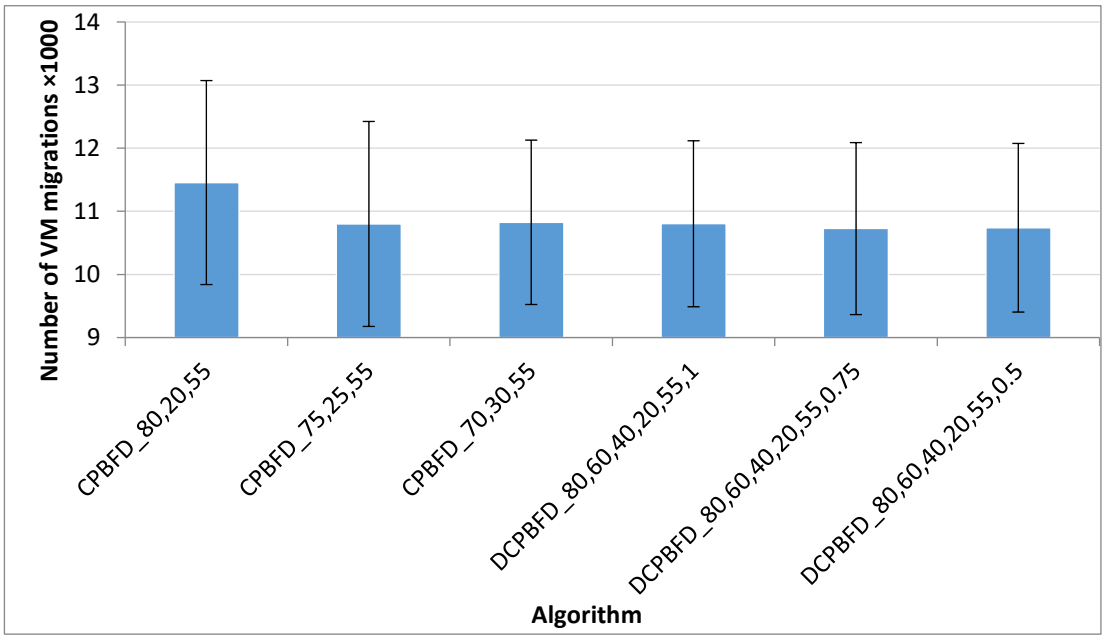

Figure 3.13: Number of VM migrations of CPBFD and DCPBFD algorithms

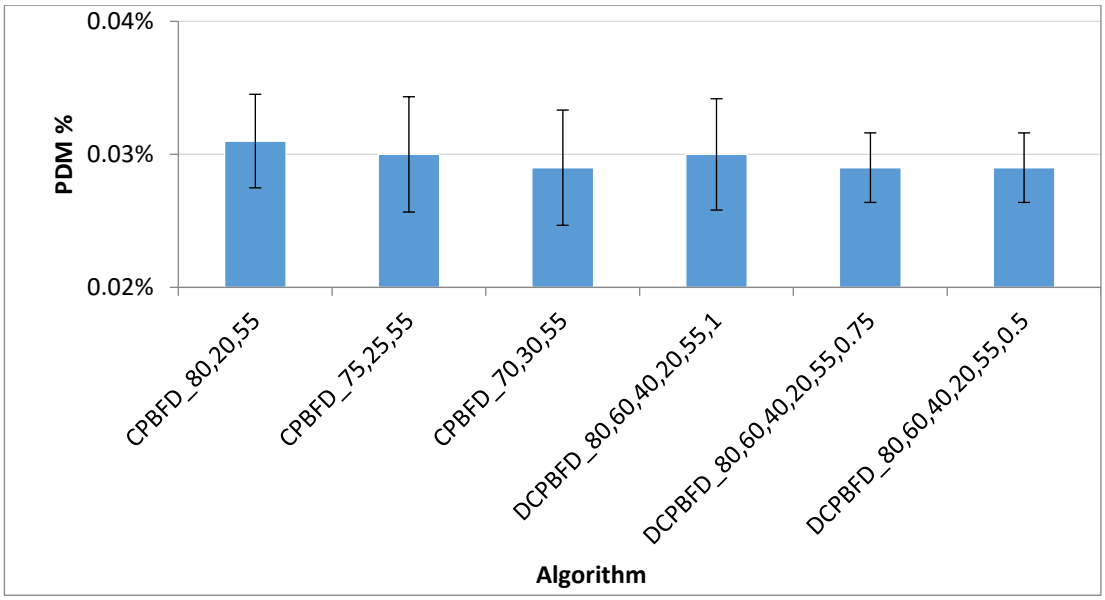

Figure 3.14: PDM metric of CPBFD and DCPBFD algorithms

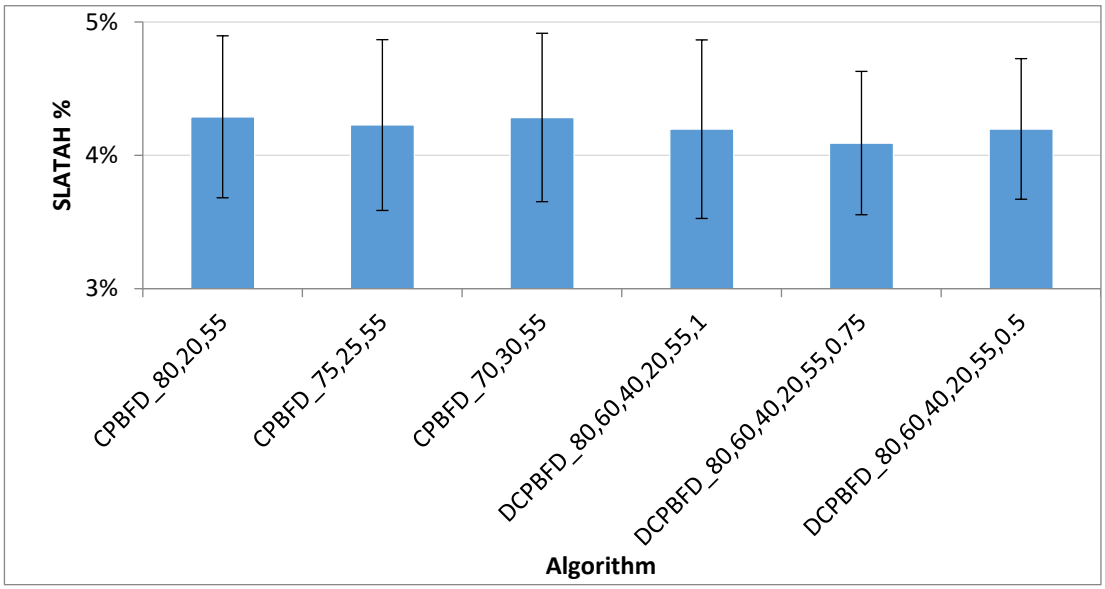

Figure 3.15: SLATAH metric of CPBFD and DCPBFD algorithms

Results in Figures 3.10 to 3.15 show that DCPBFD_80,60,40,20,55,0.75 leads to better results compared to DCPBFD with other parameters, and CPBFD_75,25,55 leads to better results compared to CPBFD with other parameters. Results in Figures 3.10, 3.12, 3.13, 3.14, and 3.15 show that DCPBFD_80,60,40,20,55,0.75 leads to better of ESV metric, SLAV metric, number of migrations, PDM metric, and SLATAH metric compared to DCPBFD and CPBFD with other parameters.

Moreover, results in Figure 3.11 show that DCPBFD_80,60,40,20,55,0.5 provides a little better efficient energy consumption compared to DCPBFD and CPBFD with other parameters. However, PABFD algorithm leads to a little better energy consumption than DCPBFD_80,60,40,20,55,0.5 algorithm. We observe that selecting moderate two-sided limits of CPU utilization for DCPBFD by selecting moderate highest CPU utilization and lowest CPU utilization and adjusting *s* parameter to moderate value is better than selecting too wide or too narrow sided limits of CPU utilization.

Chapter 4

PHYSICAL MACHINE OVERLOAD DETECTION ALGORITHMS FOR DYNAMIC VIRTUAL MACHINE CONSOLIDATION IN CLOUD DATA CENTERS

4.1 Introduction

In this chapter, an approach known as Percentage of Overload Time Fraction Threshold (POTFT) is proposed that decides to migrate a VM if the current Overload Time Fraction (OTF) value of PM exceeds the defined percentage of maximum allowed OTF value, to avoid exceeding the maximum allowed resulting OTF value after a decision of VM migration or during VM migration. The proposed POTFT algorithm is also combined with VM quiescing to maximize the time until migration, while meeting QoS goals. A number of benchmark PM overload detection algorithms are implemented using different parameters to compare with POTFT with and without VM quiescing. We evaluate the algorithms through simulations with real world workload traces and results show that the proposed approaches outperform the benchmark PM overload detection algorithms. The results also show that proposed approaches lead to better time until migration by keeping average resulting OTF values less than allowed values. Moreover, POTFT algorithm with VM quiescing is able to minimize number of migrations according to QoS requirements and meet OTF constraint with a few quiescings. The contribution of the Chapter is a novel combination of modified PM overload detection algorithm and VM quiescing, which

is able to meet constraints on the allowed number of migration of VMs in a certain period of time.

The rest of the chapter is organized as follows. In Section 4.2, we introduce proposed PM overload detection algorithm and combination of PM overload detection algorithm and VM quiescing. In Section 4.3, we introduce the experimental settings of the CPU model. Finally, the experimental evaluations and results are discussed in Section 4.4.

4.2 Consolidation of Virtual Machines

4.2.1 A Workload QoS Metric

We use a workload QoS metric from [4] to impose QoS requirements on the system. PM can be according to its CPU utilization: serving normal load or serving high load. It is assumed that VMs allocated to an overloaded PM might not provide the required performance level, which leads to performance degradation. We use OTF metric, which allows measuring the performance degradation in regard to the overload state over interval of time [4]. OTF metric is calculated as in (4.1):

$$OTF\ (100\%) = \frac{t_o\ (100\%)}{t_a} \tag{4.1}$$

where t_0 is the total time during which PM is in the overload state when its CPU utilization is 100%, and t_a is the total time of PM being in the active state. OTF is monitored continuously, and is recalculated every time PM overload detection is invoked.

4.2.2 Proposed PM Overload Detection Algorithm

We propose a PM overload detection algorithm based on OTF Threshold (OTFT) algorithm [4]. OTFT algorithm decides to migrate a VM if the current OTF value of PM exceeds the defined Maximum Allowed OTF (MAOTF) value. However, OTFT

algorithm fails to meet SLA requirements since OTF threshold equals to MAOTF and the average resulted OTF value exceeds the maximum allowed resulting OTF value [4].

The proposed overload detection algorithm decides to migrate a VM if the current OTF value of PM (pm.OTF) exceeds the defined percentage (p) of MAOTF value to avoid exceeding the maximum allowed resulting OTF value after a decision of VM migration or during VM migration. We refer to this algorithm as POTFT algorithm. POTFT algorithm is presented below.

Algorithm 4.1: POTFT PM Overload Detection Algorithm

Input: MAOTF, pm.OTF, p

Output: PM's status

1: PM's status = normal loaded

2: **if** pm.OTF > (MAOTF × p) **then**

3: PM's status = overloaded

4: **end if**

5: **return** PM's status

4.2.3 Proposed PM Overload Detection Algorithm with VM Quiescing

Current research may not be able to limit the number of VM migrations to be less than maximum allowed number of VM migrations and at the same time do not allow PM to exceed maximum allowed OTF. This study focuses on PM overload detection algorithms and proposes the combination of proposed PM overload detection algorithms with VM quiescing [48], which is able to minimize number of VM migrations and meet OTF constraint. VM quiescing should be applied with efficient PM overload detection algorithm to minimize number of VM quiescings required.

Therefore, we propose to combine VM quiescing with POTFT algorithm to minimize the number of VM migrations to be less than maximum allowed number of VM migrations and limit OTF to be less than MAOTF according to QoS constraints.

PM overload detection algorithm is invoked periodically to decide if a PM is overloaded or not. The proposed algorithm turns off one of VMs randomly from the set of generated by Algorithm 4.3, if PM is highly loaded according to POTFT algorithm that its OTF value exceeds specified percentage of MAOTF, and current simulation time does not exceed minimum allowed time until migration. Turned off VMs are restarted when PM no longer suffers from overload. We supposed that PM is no longer considered suffering from overload when OTF value of PM is less than 10% subtracted from specified percentage of MAOTF value. If PM still suffers from overload and current simulation time exceeds minimum allowed time until migration, a VM should be migrated from PM. The maximum allowed number of VM migrations initiated over n time steps (n/T) is considered as QoS requirement, where T is the minimum allowed time until migration and is set according to the maximum allowed number of VM migration in time. Combination of POTFT and VM quiescing algorithm is presented in Algorithm 4.2.

Algorithm 4.2: Combination of POTFT and VM Quiescing Algorithm

Input: A set of generated VMs, a set of turned off VMs, pm.OTF, T, current simulation time, MAOTF, p

Output: A set of turned off VMs and a decision on whether to migrate a VM

1: a decision on whether to migrate a VM =false

2: **if** pm.OTF > (MAOTF × p) **then**

3: **if** current simulation time > T **then**

4: a decision on whether to migrate a VM = true

5: **else**

6: Select VM randomly from a set of generated VMs

7: Remove a CPU utilization trace of a selected VM

8: Add a VM to the set of turned off VMs

VM Quiescing

9: **end if**

10: **else**

11: **if** the set of turned off VMs ≠ null and pm.OTF < (MAOTF × (p - 0.1))

12: Select a VM randomly from turned off VMs

13: Assign a CPU utilization trace for the selected VM

14: Remove a VM from the set of turned off VMs

VM

15: **end if**

16: **end if**

17: **return** (a set of turned off VMs, a decision on whether to migrate a VM)

4.2.4 Benchmark PM Overload Detection Algorithms

We evaluate the proposed algorithms using a number of well-known benchmark PM overload detection algorithms used in most recent research for comparison purposes [8, 31, 41, 47]. The first benchmark algorithm is a simple heuristic algorithm based on specifying fixed CPU utilization Threshold (THR), which is applied in a number of related works [7, 59, 60, 61]. PM's CPU utilization is monitored and if the specified upper threshold is exceeded, a VM is migrated. The next two algorithms are based on the statistical analysis to adjust CPU utilization threshold dynamically: based on MAD and Interquartile Range (IQR) [9]. MAD adjusts upper and lower PM's utilization thresholds and keeps the total utilization of VMs between these thresholds. IQR adjusts an upper PM's utilization threshold based on a measure of

statistical dispersion, being equal to the first quartile subtracted from the third quartile. The next two algorithms estimate the future CPU utilization using regression-based approach and a modification of the robust regression method, which is robust to outliers [9]. These algorithms are denoted as LR and Local Regression Robust (LRR) respectively. The main idea of LR, which is proposed by Guenter et al. [11], is to fit simple models to localized subsets of data to build up a curve that approximates the original data. LR algorithm is in line with robust regression method. Another algorithm is Markov Host Overload Detection (MHOD), which is proposed by Beloglazov et al. [4]. MHOD algorithm uses multisize sliding window technique to predict future workload based on a Markov chain model. Two other algorithms discussed in Section 4.2.2 are OTFT and the proposed POTFT algorithm. In this thesis, the benchmark PM overload detection algorithms are compared to proposed POTFT algorithm and POTFT with VM quiescing.

4.3 CPU Modeling

We use the CPU model presented in [4]. The model is appropriate for single core and multi-core CPU architectures. The single core CPU capacity is modeled according to its clock frequency, *F*. A CPU utilization of VMs, u_i,is a fraction of CPU utilization of PM, *U*, and is relative to CPU frequency of VMs, f_i. CPU utilization of PM at time *t* equals the summation of fractions of *L* VMs running on PM as in (4.2).

$$U(t)=\frac{1}{F}\sum_{i=1}^{L} f_i u_i(t) \tag{4.2}$$

We model a multi-core CPU for the investigation of PM overload detection problem. A clock frequency F_c of a multi-core CPU with *k* cores is modeled as kF_c frequency of a single core CPU, which means *F* in (4.2) is replaced by kF_c. Using a time-shared scheduling algorithm, each VM is randomly assigned to one of CPU's cores. The only restriction is that VM's CPU capacity cannot exceed the single core capacity. If

this restriction is removed, a VM will be required to be executed on multi-cores in parallel.

4.4 Performance Evaluation of PM Overload Detection Algorithms using CPU Model

We use simulations to evaluate the proposed POTFT algorithm and POTFT with VM quiescing. We use the source code of benchmark PM overload detection algorithms which is written in Clojure [4] and add the code of proposed POTFT algorithm and VM quiescing algorithm. Appendix H presents installation and running of the Clojure. Appendix I presents the code of calculation CPU utilization of PM. Appendix J and K present the code of POTFT algorithm and POTFT with VM quiescing, respectively. Appendix Q presents simulation model structure build using Clojure.

4.4.1 Evaluation of PM Overload Detection Algorithms using Planet-Lab Workload Traces

In multiple PMs environment, PM overload detection algorithm works independently in a decentralized way on every PM. So, a variety of heterogeneous VMs is served using a single PM with a quad-core CPU to evaluate PM overload detection algorithm under a real world workload. The clock frequency of each core of PM is set to 3 GHz, which transforms into 12 GHz according to CPU model in Section 4.3. The above CPU characteristics correspond to a medium range type in the cloud physical Amazon EC2 servers [62]. It is assumed that the memory size of PM is enough for VMs. CPU frequency of each created VM is randomly set to: 1.7 GHz, 2 GHz 2.4 GHz, or 3 GHz, which corresponds to the types of Amazon EC2 instance [63]. CPU utilization used in the simulations is based on workload traces from the CoMon project, a monitoring tool for Planet-Lab [58]. The provided workload traces

are collected every 5 minutes from more than a thousand VMs located at more than 500 places around the world. This trace was taken from March to April in 2011.

To run a simulation, a set of VMs is generated randomly with the assigned CPU utilization traces allocated on PM. PM overload detection technique at each time step decides if a VM migration should be done or not. The simulation ends when a VM is decided to be migrated, or when all workload traces are assigned. When a simulation ends, the average OTF is calculated according to (4.1). A set of VMs is assigned with the workload traces by the workload trace assignment algorithm, which is presented in Algorithm 4.3 [4]. The original workload traces is filtered to assign more dynamic workloads to PM overload detection algorithms. After the first 30 time steps, the minimum overall OTF is limited to 20%, and the maximum allowed OTF is limited to 10%. 100 various sets of VMs that meet the specified OTF requirements is regenerated by the workload trace assignment algorithm, and each PM overload detection algorithm is run for all sets of VMs [4]. Appendix L presents some CPU utilization traces. Appendix M presents the code of workload generator.

Algorithm 4.3: Workload Trace Assignment Algorithm

Input: A set of CPU utilization traces

Output: A set of generated VMs

1: Select PM's minimum CPU utilization randomly from (80%, 85%, 90%, 95%, and 100%) at the time 0

2: **while** PM's CPU utilization < PM's minimum CPU utilization at the time 0

3: Randomly select CPU frequency of new VM

4: Randomly assign a CPU utilization trace

5: Add new VM to the set of generated VMs

6: **end while**

7: **return** the set of generated VMs

The output of Algorithm 4.3 is used as one of the inputs to Algorithm 4.2.

4.4.2 Simulation Results

In this section, we compare THR, MAD, IQR, LR, LRR, OTFT, MHOD, POTFT, and POTFT with VM Quiescing (POTFT_Q) algorithms with the experimental environment settings presented in Section 4.4.1. For each overload detection technique, the parameters are used as presented in Table 4.1.

Table 4.1: Parameters used in PM overload detection algorithms

Algorithm	**Parameter**	**Value 1**	**Value2**	**Value3**
THR	CPU utilization threshold [59- 61]	80%	90%	100%
MAD	The median of the absolute deviations [9, 59]	2	3	-
IQR	The median of the absolute deviations [9, 59]	1	2	-
LR	Estimated trend line [9, 59]	1.2	1.1	1.0
LRR				
OTFT	OTF threshold (MAOTF) [4]	10%	20%	30%
MHOD	OTF threshold (MAOTF) [4]	10%	20%	30%
POTFT	OTF threshold (MAOTF × *p*)	(10%×80%) 8%	(20%×80%) 16%	(30%×80%) 24%
		(10%×85%) 8.5%	(20%×85%) 17%	(30%×85%) 25.5%
		(10%×90%) 9%	(20%×90%) 18%	(30%×90%) 27%

MAOTF value of OTFT, MHOD and POTFT is set to such as 10%, 20% and 30%. OTF threshold for OTFT is set to equal MAOTF (10%, 20% and 30%) as presented in [4]. And OTF threshold for POTFT is set to 8%, 8.5%, 9%, 16%, 17% 18%, 24%,

25.5%, and 27% which equal to varied percentages (85%, 90% and 95%) multiplied by MAOTF (10%, 20% and 30%). Average value of resulted OTF was not exceeding 10%, 20% and 30% respectively, by tuning parameters of PM overload detection algorithm to be increased from first to third parameter as shown in Table 4.1. Minimum allowed time until migration (T) is set to 40,000 seconds and 80,000 seconds for all algorithms. OTF parameter values that are monitored continuously and recalculated every time PM overload detection is invoked as stated in Section 4.2.1 and the minimum allowed time until migration are varied according to various supposed QoS constraints. 46 various combinations of the algorithms and parameters result by these variations. VM quiescing may occur many times before migration. The better algorithm is the one that maximizes the time until migration and does not let resulting OTF value to exceed maximum allowed value, which satisfies QoS constraint. Appendix N presents simulation process steps for all PM overload detection algorithms. In addition, Appendix O presents the results of PM overload detection algorithms.

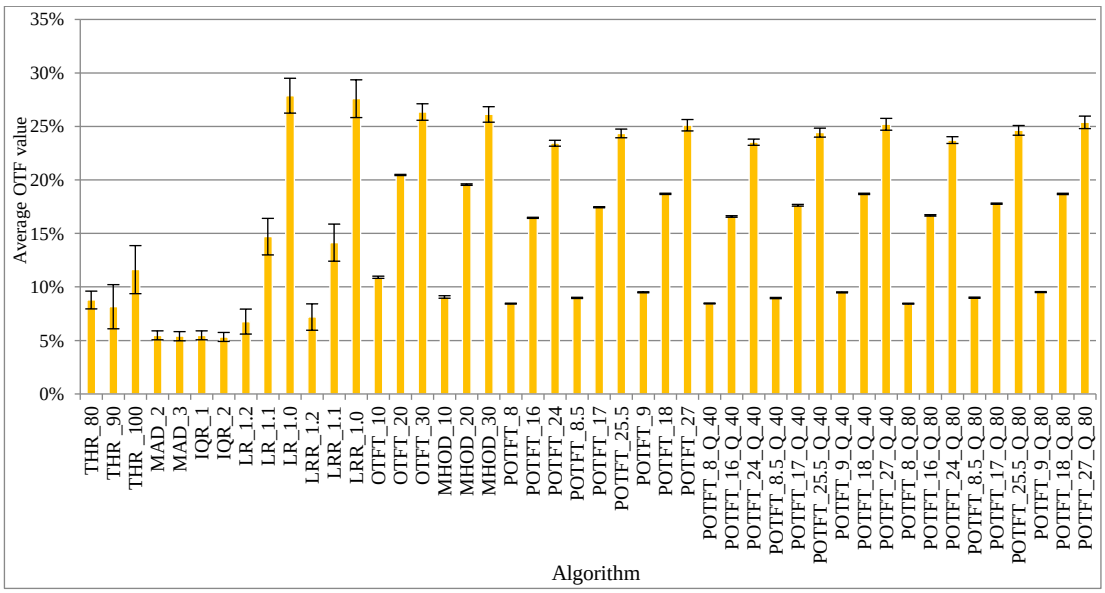

Figure 4.1: Average OTF values with 95% confidence interval of PM overload detection algorithms

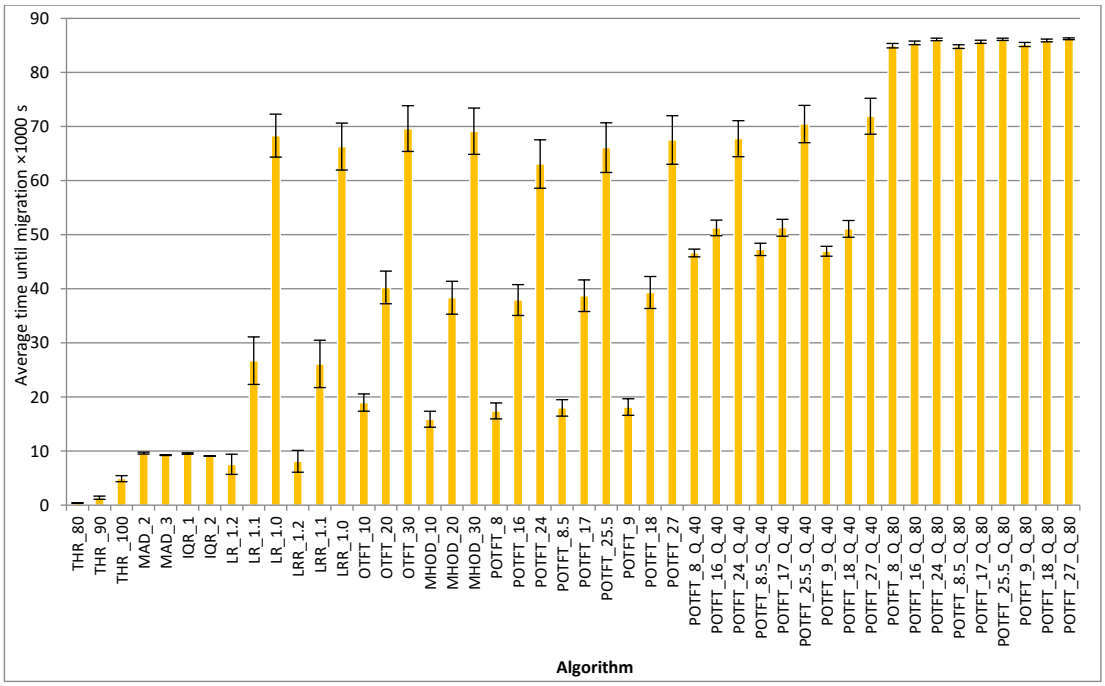

Figure 4.2: Average time until a VM migration with 95% confidence interval of PM overload detection algorithms

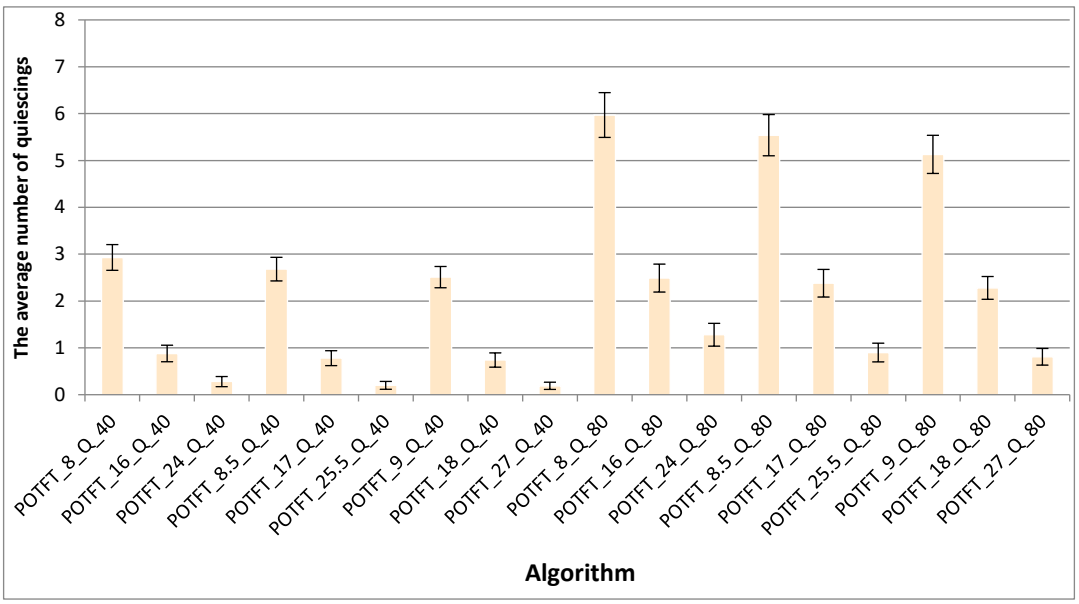

Figure 4.3: Average number of quiescings with 95% confidence interval of combination of POTFT algorithm and VM quiescing

Figure 4.1 presents the average OTF values with 95% confidence interval for benchmark PM overload detection algorithms, proposed POTFT without quiescing, and proposed POTFT_Q. It can be seen from the results in Figure 4.1 that MHOD,

POTFT and POTFT_Q with all parameters are the only competitive algorithms. They maximize resulting OTF value without allowing it to exceed maximum allowed value (10%, 20%, and 30%) and maximize the time until migration, while POTFT_Q is able to maximize time until migration to be more than minimum allowed time until migration as shown in Figure 4.2.

Figure 4.2 presents the average time until a migration with 95% confidence interval for PM overload detection algorithms. The results in Figure 4.3 show that OTFT, proposed POTFT without quiescing and MHOD have longer time until migration compared to other overload detection algorithms, and POTFT_Q is able to maximize time until migration to be more than minimum allowed time until migration (40,000 seconds and 80,000 seconds). The results in Figure 4.3 show that there is a statistically significant difference in the average time until a migration produced by LR_1, LRR_1, OTFT_30, MHOD_30, POTFT_24, POTFT_25.5, and POTFT_27 algorithms compared to other algorithms except POTFT_Q. OTFT and POTFT algorithms have better average time until a migration compared to LR, and LRR. Moreover, the resulting OTF of POTFT and POTFT_Q and the time until migration of POTFT are increased when *p* parameter is increased from 80% to 90%.

Figure 4.3 presents the average number of VM quiescings with 95% confidence interval for POTFT_Q to maximize the time until migration to 40,000 seconds and 80,000 seconds. A large number of quiescings may highly affect QoS. However, Figure 4.3 show that the POTFT_Q with varied parameters has a small number of quiescings. Therefore, POTFT_Q is able to maximize time until migration according to QoS requirements with a few quiescings.

Table 4.2: SLA violations by OTFT and POTFT

OTF Parameter	OTFT	MHOD	POTFT_80%	POTFT_85%	POTFT_90%
10%	100/100	0/100	0/300	0/300	3/300
20%	100/100	1/100	0/300	0/300	0/300
30%	44/100	0/100	0/300	0/300	0/300
Overall	81.33%	0.33%	0%	0%	0.33%

Table 4.2 presents the levels of SLA violations caused by OTFT, MHOD and POTFT algorithms. The results show that POTFT and MHOD significantly outperform OTFT algorithm according to SLA violations levels. POTFT with 80% and 85% of maximum allowed OTF leads to 0% SLA violations, POTFT with 90% of maximum allowed OTF causes only 0.33% SLA violations, MHOD allowed OTF causes only 0.33% SLA violations, whereas OTFT causes 81.33% SLA violations.

The results in Figure 4.2 show that the average time until a migration of POTFT algorithm is slightly lower than OTFT algorithm, however POTFT is able to avoid SLA violations as shown in Table 4.2. The experimental results show that the proposed POTFT and MHOD lead to higher average time until a migration compared to benchmark algorithms while meeting the specified OTF goal. Moreover, POTFT_Q is able to minimize number of migrations according to QoS requirements and meet OTF constraint with a few quiescings.

Chapter 5

CONCLUSIONS AND FUTURE WORKS

5.1 Conclusions

Energy efficiency is the most important design requirement for modern computing systems, as they continue to consume large amounts of electrical power. Apart from high operating costs incurred by computing resources, this leads to significant emissions of CO_2 into the environment. Most of the energy consumption of data centers is consumed by computing resources. Accordingly, resource management is important to ensure that the applications efficiently utilize the available computing resources. We have overviewed of a number of recent research works in energy-efficient resource management in virtualized data centers. It is necessary to study the existing resource management approaches to facilitate further improvements in this area.

CPBFD and DCPBFD VM placement algorithms are proposed to improve energy efficiency, and SLA in cloud data centers. A number of VM placement algorithms are simulated for comparison with the proposed algorithms. We compare the algorithms through simulations with real-world workload traces. CPBFD and DCPBFD algorithms produce better results by avoiding VM migrations to PM with high load that may cause SLA violations or low load, which lead to more active PMs and more energy consumption. The simulation results show that CPBFD and DCPBFD with moderate two-sided limits of CPU utilization provide the least ESV,

least SLA violations, least VM migrations, and efficient energy consumption. However, PABFD algorithm leads to a little better energy consumption than CPBFD and DCPBFD algorithms.

Other studies may not be able to limit the number of VM migrations to be less than maximum allowed number of VM migrations and at the same time do not allow PM to exceed MAOTF according to QoS constraints. Therefore, we proposed dynamic VM consolidation based on a new PM overload detection algorithm and a combination of PM overload detection algorithm and VM quiescing. The goal of the model is to improve the utilization of resources and energy efficiency in cloud data centers. We simulated a number of PM overload detection algorithms using different parameters to compare with the proposed POTFT algorithm and POTFT with VM quiescing. The algorithms are evaluated through simulations using real world workload traces. The results of our experiments show that proposed PM overload detection algorithm outperforms the benchmark PM overload detection algorithms and leads to higher performance of benchmark PM overload detection algorithms, while meeting the QoS target. The results show that proposed PM overload detection algorithm leads to better time until migration with keeping average resulting OTF values less than allowed values. Moreover, the proposed POTFT algorithm with VM quiescing is able to minimize number of migrations according to QoS requirements and meet OTF constraint with a few quiescings.

5.2 Future Works

For future work, we suggest to improve network topologies established between VMs for virtualized data centers to reduce the load of network devices and the network communication overhead. We also propose to investigate cloud federations

including geographically distributed data centers. Another research direction is to investigate the control of user over the energy consumption in data centers, in addition to support the flexible SLA negotiation between users and resource providers.

As a future work, we plan to implement the proposed algorithms on a software framework for dynamic and energy efficient consolidation of VMs called OpenStack Neat [64] applied in existing OpenStack Clouds [65] deployments and in research on dynamic consolidation of VMs to optimize the resource utilization and energy efficiency.

REFERENCES

[1] Weber, W.D., Fan, X., & Barroso, L. A. (2013). *Powering the data center. U.S.* Patent No. 8,595,515. Washington, DC: U.S. Patent and Trademark Office.

[2] Barroso, L., (2005). The Price of Performance. *ACM Queue*, 3(7), 53.

[3] Beloglazov, A., Buyya, R. ,Lee, Y. C., & Zomaya , A. (2011). A taxonomy and survey of energy-efficient data centers and cloud computing systems. *Advances in computers*, 82(2), 47-111.

[4] Beloglazov, A., & Buyya, R. (2013). Managing overloaded hosts for dynamic consolidation of virtual machines in cloud data centers under quality of service constraints. *IEEE Transactions on Parallel and Distributed Systems*, 24(7), 1366-1379.

[5] Verma, A., Ahuja, P., & Neogi, A. (2008). pMapper: Power and migration cost aware application placement in virtualized systems. *Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware*, Leuven, Belgium, 1-5 December 2008, 243-264.

[6] Jung, G., Hiltunen, M.A., Joshi, K.R., Schlichting, R.D., & Pu, C. (2010). Mistral: Dynamically managing power, performance, and adaptation cost in cloud infrastructures. *Proceedings of the 30th International Conference on Distributed Computing Systems (ICDCS)*, Genova, Italy, 21-25 June 2010, 62-73.

[7] Gmach, D., Rolia, J., Cherkasova, L., & Kemper, A. (2009). Resource pool management: Reactive versus proactive or lets be friends. *Computer Networks*, 53(17), 2905-2922.

[8] Fu, X., & Zhou, C. (2015). Virtual machine selection and placement for dynamic consolidation in Cloud computing environment. *Frontiers of Computer Science*, 9(2), 322-330.

[9] Beloglazov, A., & 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 (CCPE)*, 24, 1397-1420.

[10] Han, G., Que, W., Jia, G., & Shu, L. (2016). An efficient virtual machine consolidation scheme for multimedia cloud computing. *Sensors*, 16(2), 246.

[11] Guenter, B., Jain, N., & Williams, C. (2011). Managing cost, performance, and reliability tradeoffs for energy-aware server provisioning. *Proceedings of the 30st Annual IEEE International Conference on Computer Communications (INFOCOM)*, Shanghai, China, 10-15 April 2011, 1332-1340.

[12] Zheng, W., Bianchini, R., Janakiraman, G., Santos, J., & Turner, Y. (2009). JustRunIt: Experiment-based management of virtualized data centers. *Proceedings of the 2009 USENIX Annual Technical Conference*, San Diego, California, 14-19 June 2009 , 18-33.

[13] Alsbatin, L., Oz, G., & Ulusoy, A. H. (2017) An overview of energy-efficient cloud data centers. *Proceedings of the International Conference of computer and applications (ICCA2017),* Dubai, United Arab Emirates, 6-7 September 2017, 253- 258.

[14] Nathuji, R., & Schwan, K. (2007). Virtualpower: Coordinated power management in virtualized enterprise systems. *ACM SIGOPS Operating Systems Review*, 41(6), 265-278.

[15] Raghavendra, R., Ranganathan, P., Talwar, V., Wang, Z., & Zhu, X. (2008). No ”power” struggles: Coordinated multi-level power management for the data center. *SIGARCH Computer Architecture News*, 36(1), 48–59.

[16] Kusic, D., Kephart, J. O., Hanson, J. E., Kandasamy, N., & Jiang, G. (2009). Power and performance management of virtualized computing environments via lookahead control. *Cluster Computing*, 12(1), 1–15.

[17] Song, Y., Wang, H., Li , Y., Feng, B., & Sun Y. (2009). Multi-tiered on-demand resource scheduling for VM-based data center. *Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2009),* Shanghai, China, 18-21 May 2009. 148–155.

[18] Stillwell, M., Schanzenbach, D., Vivien, F., & Vivien H. (2009). Resource allocation using virtual clusters. *Proceedings of the 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, Shanghai, China, 18-21 May 2009, 260–267.

[19] Cardosa, M., Korupolu, M., & Singh, A. (2009). Shares and utilities based power consolidation in virtualized server environments. *Proceedings of the 11th IFIP/IEEE International Symposium on Integrated Network Management (IM)*, Long Island, NY, USA, 1-5 June 2009, 327–334.

[20] Kumar, S., Talwar, V., Kumar, V., Ranganathan, P., & Schwan, K. (2009). vManage: Loosely coupled platform and virtualization management in data centers. *Proceedings of the 6th International Conference on Autonomic Computing (ICAC)*, Barcelona, Spain, 15-19 June 2009, 127–136.

[21] Barford, P. & Crovella, M. (1998). Generating representative web workloads for network and server performance evaluation. *ACM SIGMETRICS Performance Evaluation Review*, 26(1), 151–160.

[22] Li, H. (2009). Workload dynamics on clusters and grids. *The Journal of Supercomputing*, 47(1), 1–20.

[23] Buyya, R., Beloglazov, A., & Abawajy, J. (2010). Energy-efficient management of data center resources for Cloud computing: A vision, architectural elements, and open challenges. *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, Las Vegas, NV, USA, 12-15 July 2010, 1–12.

[24] Calheiros, R. N., Ranjan, R., Beloglazov, A., Rose C.A.F.D., & Buyya, R. (2011). CloudSim: a toolkit for modeling and simulation of cloud computing

environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience, Wiley Press*, New York, USA, 41(1), 23–50.

[25] Beloglazov, A., & Buyya, R. (2014). OpenStack Neat: A Framework for Dynamic and Energy-Efficient Consolidation of Virtual Machines in OpenStack Clouds. Concurrency and Computation: *Practice and Experience (CCPE)*, 27(5), 1310-1333.

[26] Horri, A., Mozafari, M. S., & Dastghaibyfard, G. (2014). Novel resource allocation algorithms to performance and energy efficiency in cloud computing. *Journal of Supercomputing*, 69, 1445–1461.

[27] Arianyan, E., Taheri, H., & Sharifian, S. (2015). Novel energy and SLA efficient resource management heuristics for consolidation of virtual machines in cloud data centers. *Computers and Electrical Engineering*, 47, 222-240.

[28] Sharifi, M., Salimi, H., & Najafzadeh, M. (2012). Power-efficient distributed scheduling of virtual machines using workload-aware consolidation techniques. *Journal of Supercomputing*, 61(1), 46-66.

[29] Shaw, S. B., & Singh, A. K. (2015). Use of proactive and reactive hotspot detection technique to reduce the number of virtual machine migration and energy consumption in cloud data center. *Computers and Electrical Engineering*, 47, 241-254.

[30] Nguyen, T. H., Francesco, M. D., & Yla-Jaaski, A. (2017) Virtual Machine Consolidation with Multiple Usage Prediction for Energy-Efficient Cloud Data Centers. *IEEE Transactions on Services Computing*, 99, 1-14.

[31] Zhou, Z., Abawajy, J., Chowdhury, M., Hu, Z., Li, K., Cheng, H., Alelaiwi, A. A., & Li, F. (2018). Minimizing SLA violations and power consumption in cloud data centers using adaptive energy-aware algorithms. *Future Generation Computer Systems*, 86, 836-850.

[32] Kakadia, D., Kopri, N., & Varma, V. (2013) Network-aware virtual machine consolidation for large data centers. *Proceedings of the Third International Workshop on Network-Aware Data Management*, ACM. Denver, CO, USA, 17-21 November 2013.

[33] Mattias, F., Andreas, G., Lars, L., & Dragos, L. (2015). Algorithms for automated live migration of virtual machines. *Journal of System and Software*, 101, 110-126.

[34] Gupta, M.K., Jain, A., & Amgoth, T. (2018). Power and resource-aware virtual machine placement for IaaS cloud. *Sustainable Computing: Informatics and Systems*, 19, 52-60.

[35] Halacsy, G., & Mann, Z. A. (2018). Optimal energy-efficient placement of virtual machines with divisible sizes. *Information Processing Letters*, 138, 51-56.

[36] Yousefipour, A., Rahmani, A. M., & Jahanshahi, M. (2018). Energy and cost-aware virtual machine consolidation in cloud computing. *Software: Practice and Experience*, 48(10), 1758–1774.

[37] Beloglazov, A., Abawajy, J., & Buyya, R. (2012). Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future Generation Computer Systems*, 28(5), 755-768.

[38] Yue, M. (1991). A simple proof of the inequality FFD (L)< 11/9 OPT (L)+ 1,for all l for the FFD bin-packing algorithm. *Acta Mathematicae Applicatae Sinica (English Series)*, 7(4), 321-331.

[39] Coffman, E. G., Garey, M. R., & Johnson, D. S. (1996). Approximation algorithms for bin-packing: A survey. *Approximation algorithms for NP-hard problems*, 46-93.

[40] Shi, L., Furlong, J., & Wang, R. (2013). Empirical evaluation of vector bin packing algorithms for energy efficient data centers. *Proceedings of IEEE Symposium on Computers and Communications*, Split, Croatia, 7-10 July 2013, 9-15.

[41] Kaushar, H., Ricchariya, P., & Motwani, A. (2014). Comparison of SLA based energy efficient dynamic virtual vachine consolidation algorithms. *International Journal of Computer Applications*, 102(16), 31-36.

[42] Yadav, R. & Zhang, W. (2017) MeReg: Managing Energy-SLA Tradeoff for Green Mobile Cloud Computing. *Wireless Communications and Mobile Computing*, 2017, 1-11.

[43] Deng, D., He, K., & Chen, Y. (2016). Dynamic virtual machine consolidation for improving energy efficiency in cloud data centers. *Proceedings of 4th International Conference on Cloud Computing and Intelligence Systems*, Beijing, China, 17-19 August 2016, 366-370.

[44] Khoshkholghi, M. A., Derahman, M. N., Abdullah, A., Subramaniam, S., & Othman, M. (2017). Energy-efficient algorithms for dynamic virtual machine consolidation in cloud data centers. *IEEE Access*, 5, 10709-10722.

[45] Wang, X., & Wang, Y. (2011). Coordinating power control and performance management for virtualized server clusters. *IEEE Transactions on Parallel and Distributed Systems*, 22(2), 245-259.

[46] Forsman, M., Glad, A., Lundberg, L., & Ilie, D. (2015). Algorithms for automated live migration of virtual machines. *Journal of Systems and Software*, 101, 110-126.

[47] Zhou, H., Li, Q., Choo, K. R., & Zhu, H. (2018). DADTA: A novel adaptive strategy for energy and performance efficient virtual machine consolidation. *Journal of Parallel and Distributed Computing*, 121, 15-26.

[48] Baset S. A., Wang L., & Tang C. (2012). Towards an Understanding of Oversubscription in Cloud. *Proceedings of the 2nd USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*, San Jose, CA, USA, 25-27 April 2012, 7-12.

[49] Minas, L., & Ellison, B. (2009). *Energy efficiency for information technology: How to reduce power consumption in servers and data centers*. Intel Press

[50] Fan, X., Weber, W. D., Barroso, L.A. (2007). Power provisioning for a warehouse-sized computer. *ACM SIGARCH Computer Architecture News*, 35(2), 13-23.

[51] Lange, K. D. (2009). Identifying shades of green: *The SPECpower benchmarks. Computers*, 42(3), 95-97.

[52] Beloglazov, A., & Buyya R. (2010). Adaptive threshold-based approach for energy-efficient consolidation of virtual machines in cloud data centers. *Proceedings of the 8th International Workshop on Middleware for Grids, Clouds and e-Science*, Bangalore, India, 29 November to 3 December 2010, 826-831.

[53] Murtazaev, A., Oh, S. (2011). Sercon: Server consolidation algorithm using live migration of virtual machines for green computing. *IETE Techical Review*, 28(3), 212-231.

[54] Vogels, W. (2008). Beyond Server Consolidation. *ACM Queue*, 6(1), 20-26.

[55] Magesh, H., & Smith, J. (2008). Server consolidation through virtualization with quad-core intel xeon processors. *White Paper by Intel Corporation and Infosys Technologies*.

[56] Wood, T., Shenoy, P., Venkataramani, A., & Yousif, M. (2007). Black-box and gray-box strategies for virtual machine migration. *Proceedings of the 4th USENIX Symposium on Networked Systems Design & Implementation*, Cambridge, MA, USA, 11-13 April 2007, 229-242.

[57] Buyya, R., Ranjan, R., & Calheiros, R. N. (2009) . Modeling and simulation of scalable cloud computing environments and the CloudSim toolkit: Challenges and opportunities. *Proceedings of 2009 Conference on High Performance Computing & Simulation Conference*, Leipzig, Germany, 21-24 June 2009, 1-11.

[58] Park, K. S., & Pai, V. S. (2006). CoMon: a mostly-scalable monitoring system for PlanetLab. *ACM SIGOPS Operating Systems Review*, 40(1), 65-74.

[59] Gmach D., and Rolia J., Cherkasova, L., Belrose, G., Turicchi, T., & Kemper, A. (2008). An integrated approach to resource pool management: Policies, efficiency and quality metrics. *Proceedings of the 38th IEEE Intl. Conference on Dependable Systems and Networks (DSN)*, Anchorage, AK, USA, 24-27 June 2008, 326-335.

[60] Zhu, X., Young, D., Watson, B. J., Wang, Z., Rolia, J., Singhal, S., McKee, B., & Hyser, C. (2008). 1000 islands: Integrated capacity and workload

management for the next generation data center. *Proceedings of the 5th International Conference on Autonomic Computing (ICAC)*, Chicago, IL, USA, 2-6 June 2008, 172-181.

[61] VMware Inc. (2010) VMware distributed power management concepts and use. Information Guide.

[62] Mills, K., Filliben, J., & Dabrowski, C. (2011.) Comparing vm-placement algorithms for on-demand clouds. *Proceedings of the 3rd IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, Athens, Greece, 29 November to 1 December 2011, 91-98.

[63] The Amazon Instance Types. (2016, October 31) Retrieved from https://aws.amazon.com/ec2/instance-types/

[64] The OpenStack Neat framework. (2016, October 31) Retrieved from http://openstack-neat.org/

[65] The OpenStack platform. (2016, October 31) Retrieved from http://openstack.org/

[66] Urgaonkar, B., Shenoy, P., & Roscoe, T. (2002). Resource overbooking and application profiling in shared hosting platforms. *Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, Boston, MA, USA, 9-11 December 2002, 36(SI), 239-254.

[67] Guo, L., Zhang, Y., & Zhao, S. (2017). Heuristic Algorithms for Energy and Performance Dynamic Optimization in Cloud Computing. *Computing and Informatics*, 36(6), 1335-1360.

[68] Liu, Y., Sun, X., Wei, W., & Jing, W. (2018). Enhancing Energy-Efficient and QoS Dynamic Virtual Machine Consolidation Method in Cloud Environment. *IEEE Access*, 6, 31224- 31235.

APPENDICES

Appendix A: Installation and Running the CloudSim Toolkit

We need to download the CloudSim 3.0.3 and latest version of the Java Development Toolkit (JDK). We just need to unpack the CloudSim file to install the CloudSim. We do not need to compile CloudSim source code. JAR files are provided to compile and to run CloudSim applications.

Appendix B: Code of Calculation PM CPU Utilization in MIPS

```
protected double getUtilizationOfCpuMips(PowerHost host) {
                double hostUtilizationMips = 0;
                for (Vm vm2 : host.getVmList()) {
                        if (host.getVmsMigratingIn().contains(vm2)) {
                // calculate additional potential CPU usage of a migrating in VM
                                hostUtilizationMips += 0.1 *
host.getTotalAllocatedMipsForVm(vm2) *
vm2.getCurrentAllocatedSize()/vm2.getCurrentAllocatedBw();
                        }
                        hostUtilizationMips +=
host.getTotalAllocatedMipsForVm(vm2);
                }
          if (hostUtilizationMips> host.getTotalMips()){
          hostUtilizationMips= host.getTotalMips();
          }
                return hostUtilizationMips;
         }
```

Appendix C: Source Code of CPBFD and DCPBFD Algorithms

1. Code of CPBFD VM Placement Algorithm:

```
public abstract class PowerVmAllocationPolicyMigrationAbstract extends
PowerVmAllocationPolicyAbstract {
protected List<Map<String, Object>> getNewVmPlacement(
                        List<? extends Vm> vmsToMigrate,
                        Set<? extends Host> excludedHosts) {
                List<Map<String, Object>> migrationMap = new
LinkedList<Map<String, Object>>();
                PowerVmList.sortByCpuUtilization(vmsToMigrate);// 1:
vmList.sortDecreasingUtilization()
                for (Vm vm : vmsToMigrate) { //2:for each vm in vmList do
                        PowerHost allocatedHost = findHostForVm(vm,
excludedHosts); // Line 3-29
                        if (allocatedHost != null) { //30:if allocatedPm≠ null then
                                allocatedHost.vmCreate(vm);// 31:
    allocation.add(vm, allocatedPm)
                                Log.printLine("VM #" + vm.getId() + " allocated to
host #" + allocatedHost.getId());
                                Map<String, Object> migrate = new HashMap<String,
Object>();
                                migrate.put("vm", vm);
                                migrate.put("host", allocatedHost);
                                migrationMap.add(migrate);
                        }//32:  endif
                }//33: end for each
                return migrationMap; // 34:return allocatedPm
    }
public PowerHost findHostForVm(Vm vm, Set<? extends Host>
excludedHosts) {
                    double maxCpu = 0; // 3: maxCpu=0
                    double minCpu = 1; // 4: minCpu=1
                PowerHost allocatedHost = null; // 5: allocatedPm= NULL
               PowerHost allocatedHost2 = null; // 6: allocatedPm2= NULL
                for (PowerHost host : this.<PowerHost> getHostList()) { //7: for each
pm in pmList do
                        if (excludedHosts.contains(host)) { //8: if excluded pms
contains pm then
                                continue;  //9: continue
                        }   //10: endif
                        if (host.isSuitableForVm(vm)) { //11: if pm has enough
resources for vm then
                                if (getUtilizationOfCpuMips(host) != 0 &&
isHostOverUtilizedAfterAllocation(host, vm)) { //12: if pm.Cpu ≠ 0 and pm is
over utilized after allocation vm then
                                        continue; //13: continue
                                } //14: endif
```

```
if((getUtilizationOfCpuMips(host)/host.getTotalMips())<=(a)&&(getUtilization
OfCpuMips(host)/host.getTotalMips())>=(b)){// 15:if pm.Cpu ≤ a and pm.Cpu ≥
b then//values of a and b is entered manually
  if ((getUtilizationOfCpuMips(host)/host.getTotalMips()) > maxCpu) { //16:
if pm.Cpu > maxCpu then
allocatedHost = host; //17:          allocatedPm = pm
maxCpu =(getUtilizationOfCpuMips(host)/host.getTotalMips());//18: maxCpu =
pm.Cpu
                }//19:                  end if
}  else{ if
((java.lang.Math.abs((getUtilizationOfCpuMips(host)/host.getTotalMips())-c)) <
minCpu) {//20: else if Abs(pm.Cpu - c) < minCpu then //value of c is entered
manually
allocatedHost2 = host; //21:                    allocatedPm2 = pm
minCpu
=java.lang.Math.abs((getUtilizationOfCpuMips(host)/host.getTotalMips())-c);
//22:    minCpu = Abs(pm.Cpu - c)  //value of c is entered manually
                        }//23:      end if
                }//24:          end if
    } // 25:          end if
} // 26:     end for each
if(allocatedHost==null){ //27:     if allocatedPm = null then
    allocatedHost=allocatedHost2; //28:          allocatedPm = allocatedPm2
}//29:     end if
                return allocatedHost;
    }
```

2. Code of DCPBFD VM Placement Algorithm:

```
public abstract class PowerVmAllocationPolicyMigrationAbstract extends
PowerVmAllocationPolicyAbstract {
protected List<Map<String, Object>> getNewVmPlacement(
        List<? extends Vm> vmsToMigrate,
        Set<? extends Host> excludedHosts) {
        List<Map<String, Object>> migrationMap = new LinkedList<Map<String,
Object>>();
        PowerVmList.sortByCpuUtilization(vmsToMigrate);// 1:
vmList.sortDecreasingUtilization()
        for (Vm vm : vmsToMigrate) { //2:for each vm in vmList do
        PowerHost allocatedHost = findHostForVm(vm, excludedHosts); // Line 3-42
        if (allocatedHost != null) { //43:if allocatedPm≠ null then
                allocatedHost.vmCreate(vm);// 44:    allocation.add(vm, allocatedPm)
                Log.printLine("VM #" + vm.getId() + " allocated to host #" +
allocatedHost.getId());
                Map<String, Object> migrate = new HashMap<String, Object>();
                migrate.put("vm", vm);
                migrate.put("host", allocatedHost);
                migrationMap.add(migrate);
                }//45:   endif
        }//46: end for each
```

```
return migrationMap; // 41:return allocatedPm
}
public PowerHost findHostForVm(Vm vm, Set<? extends Host> excludedHosts) {
        double maxCpu = 0; // 3: maxCpu=0
        double minCpu = 1; // 4: minCpu=1
        PowerHost allocatedHost = null; // 5: allocatedPm= NULL
        PowerHost allocatedHost2 = null; // 6: allocatedPm2= NULL
        for (PowerHost host : this.<PowerHost> getHostList()) { //7: for each pm in
pmList do
        if (excludedHosts.contains(host)) { //8: if excluded pms contains pm then
                continue;  //9: continue
        }   //10: endif
        if (host.isSuitableForVm(vm)) { //11: if pm has enough resources for vm
then
                if (getUtilizationOfCpuMips(host) != 0 &&
isHostOverUtilizedAfterAllocation(host, vm)) { //12: if pm.Cpu ≠ 0 and pm is over
utilized after allocation vm then
                        continue; //13: continue
                } //14: endif
PowerHostUtilizationHistory _host = (PowerHostUtilizationHistory) host;
double[] data = _host.getUtilizationHistory();
double a=0, b=0;

if (MathUtil.countNonZeroBeginning(data) >= 12) { //15: if (pm.CPUHistory.length
≥ 12) then

a= 0.8- MathUtil.mad(data)*0.75-
MathUtil.median(data)*0.75/host.getVmList().size();//16: a = maxA – s ×
(pm.CPUHistoryMAD + pm.CPUHistoryMedian / #ofvm) // values of maxA and s
are entered manually

b=0.2+MathUtil.mad(data)*0.75+
MathUtil.median(data)*0.75/host.getVmList().size(); }//17:  b = minB + s ×
(pm.CPUHistoryMAD + pm.CPUHistoryMedian / #ofvm)  // values of minB and s
are entered manually

else {//18:             else

a= maxA-0.05; //19:  a = maxA - 0.05 ) // value of maxA is entered manually

b= minB+0.05; //20:  b = minB + 0.05) // value of minB is entered manually

} //21:          end if
if (a<0.6){ //22:          if a<minA // value of  minA  is entered manually

a=0.6; //23:              a=minA

}//24:          end if

if (b>0.4){//25:          if b>maxB  // value of  maxB  is entered manually

b=0.4;//26:              B=maxB

}//27:          end if
if((getUtilizationOfCpuMips(host)/host.getTotalMips())<=(a)&&(getUtilizationOfC
puMips(host)/host.getTotalMips())>=(b)){// 28:if pm.Cpu ≤ a and pm.Cpu ≥ b then
```

```
 if ((getUtilizationOfCpuMips(host)/host.getTotalMips()) > maxCpu) { //29:
if pm.Cpu > maxCpu then
allocatedHost = host; //30:     allocatedPm = pm
maxCpu =(getUtilizationOfCpuMips(host)/host.getTotalMips());//31: maxCpu =
pm.Cpu
                }//32:                 end if
}  else{ if
((java.lang.Math.abs((getUtilizationOfCpuMips(host)/host.getTotalMips())-c)) <
minCpu) {//33: else if Abs(pm.Cpu - c) < minCpu then //value of c is entered
manually
allocatedHost2 = host; //34:                         allocatedPm2 = pm
minCpu =java.lang.Math.abs((getUtilizationOfCpuMips(host)/host.getTotalMips())-
c); //35:    minCpu = Abs(pm.Cpu - c)  //value of c is entered manually
                        }//36:      end if
                }//37:          end if
        }  //38:          end if
}  //39:     end for each
if(allocatedHost==null){ //40:     if allocatedPm = null then
        allocatedHost=allocatedHost2; //41:          allocatedPm = allocatedPm2
}//42:     end if
                return allocatedHost;
        }
```

Appendix D: Code of Reading from Input Files, Creating VMs List, Creating PMs List, VM Instance Types and PM Types for CloudSim Toolkit

1. Code of Reading from Input Files

```
cloudletList = PlanetLabHelper.createCloudletListPlanetLab(brokerId, inputFolder);
public class PlanetLabHelper {
        * Creates the cloudlet list planet lab.
        * @param brokerId the broker id
        * @param inputFolderName the input folder name
        * @return the list
        * @throws FileNotFoundException the file not found exception
       public static List<Cloudlet> createCloudletListPlanetLab(int brokerId,
String inputFolderName)
                        throws FileNotFoundException {
                List<Cloudlet> list = new ArrayList<Cloudlet>();
                long fileSize = 300;
                long outputSize = 300;
                UtilizationModel utilizationModelNull = new UtilizationModelNull();
                File inputFolder = new File(inputFolderName);
                File[] files = inputFolder.listFiles();
                for (int i = 0; i < files.length; i++) {
                        Cloudlet cloudlet = null;
                        try {
                                cloudlet = new Cloudlet(
                                                i,
                                                Constants.CLOUDLET_LENGTH,
                                                Constants.CLOUDLET_PES,
                                                fileSize,
                                                outputSize,
                                                new
UtilizationModelPlanetLabInMemory(
        files[i].getAbsolutePath(),
        Constants.SCHEDULING_INTERVAL), utilizationModelNull,
utilizationModelNull);
                        } catch (Exception e) {
                                e.printStackTrace();
                                System.exit(0);
                        }
                        cloudlet.setUserId(brokerId);
                        cloudlet.setVmId(i);
                        list.add(cloudlet);
                }
                return list;
        }
}
```

2. Code of Creating VMs List:

```
vmList = Helper.createVmList(brokerId, cloudletList.size());
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;
    }
```

3. Code of Creating PMs List:

```
hostList = Helper.createHostList(PlanetLabConstants.NUMBER_OF_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;
        }
```

4. VM Instance Types:

```
public class Constants {
    public final static boolean ENABLE_OUTPUT = true;
        public final static boolean OUTPUT_CSV    = false;
        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 MB/s
        public final static int VM_SIZE             = 2500; // 2.5 MB
```

5. PM 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 GB/s
        public final static int HOST_STORAGE = 1000000; // 1 GB
        public final static PowerModel[] HOST_POWER = {
                new PowerModelSpecPowerHpProLiantMl110G4Xeon3040(),
                new PowerModelSpecPowerHpProLiantMl110G5Xeon3075()
        }
```

Appendix E: Some CPU Utilization Traces of Input Files for CloudSim Toolkit

Table E.1: Some CPU utilization traces of input files for CloudSim toolkit

	Utilization of VM No. 0 in Worload No. 1 (%)	Utilization of VM No. 1 in Worload No. 1 (%)	Utilization of VM No. 0 in Worload No.2 (%)	Utilization of VM No. 1 in Worload No.2 (%)
Time 1	7	4	4	6
Time 2	5	2	5	2
Time 3	6	2	4	4
Time 4	4	13	2	0
Time 5	4	2	4	2

Appendix F: CloudSim Simulation Process Steps

- Simulation Process Steps for PABFD

  - Code of PABFD is already writen in simulation.
  - run LrMmt.java (LR 1.2 overload detection algorithm with, MMT selection algorithm and PABFD placement algorithm) with modifying the workload variable to name of workload folder.
  - Obtaining resuls from output files and creating result figures using MS excel.

- Simulation Process Steps for FFD

  - Write FFD placement algorithm on cloudsim-3.0.3/sources org.cloudbus.cloudsim.power/ PowerVmAllocationPolicyMifrationAbstract.java org.cloudbus.cloudsim.examples.power.planetlab.
  - run LrMmt.java (LR 1.2 overload detection algorithm with, MMT selection algorithm and FFD placement algorithm) with modifying the workload variable to name of workload folder.
  - Obtaining resuls from output files and creating result figures using MS excel.

- Simulation Process Steps for BFD

  - Write BFD placement algorithm on cloudsim-3.0.3/sources org.cloudbus.cloudsim.power/ PowerVmAllocationPolicyMifrationAbstract.java org.cloudbus.cloudsim.examples.power.planetlab.

- run LrMmt.java (LR 1.2 overload detection algorithm with, MMT selection algorithm and BFD placement algorithm) with modifying the workload variable to name of workload folder.
- Obtaining resuls from output files and creating result figures using MS excel.

- Simulation Process Steps for CPBFD

- Write CPBFD placement algorithm on cloudsim-3.0.3/sources org.cloudbus.cloudsim.power/ PowerVmAllocationPolicyMifrationAbstract.java org.cloudbus.cloudsim.examples.power.planetlab.
- run LrMmt.java (LR 1.2 overload detection algorithm with, MMT selection algorithm and CPBFD placement algorithm) with modifying the workload variable to name of workload folder.
- Obtaining resuls from output files and creating result figures using MS excel.

- Simulation Process Steps for DCPBFD

- Write DCPBFD placement algorithm on cloudsim-3.0.3/sources org.cloudbus.cloudsim.power/ PowerVmAllocationPolicyMifrationAbstract.java org.cloudbus.cloudsim.examples.power.planetlab.
- run LrMmt.java (LR 1.2 overload detection algorithm with, MMT selection algorithm and DCPBFD placement algorithm) with modifying the workload variable to name of workload folder.
- Obtaining resuls from output files and creating result figures using MS excel.

Sample output file:

Experiment name: 20110303_lr_mmt_1.2 //LR 1.2 overload detection algorithm, MMT selection algorithm and DCPBFD_ 80,20,55,0.75placement algorithm workload No. 1.

Number of hosts: 800

Number of VMs: 1052

Total simulation time: 86400.00 sec

Energy consumption: 120.85 kWh // Energy consumption Figure 3.5

Number of VM migrations: 9275 // Number of VM migrations Figure 3.7

SLA: 0.00076% //SLAV Figure 3.6

SLA perf degradation due to migration: 0.02% //PDM Figure 3.8

SLA time per active host: 3.28% // SLATAH Figure 3.9

Overall SLA violation: 0.11%

Average SLA violation: 9.72%

Number of host shutdowns: 878

Mean time before a host shutdown: 2542.72 sec

StDev time before a host shutdown: 8832.41 sec

Mean time before a VM migration: 13.21 sec

StDev time before a VM migration: 6.18 sec

Execution time - VM selection mean: 0.00181 sec

Execution time - VM selection stDev: 0.01119 sec

Execution time - host selection mean: 0.01059 sec

Execution time - host selection stDev: 0.02119 sec

Execution time - VM reallocation mean: 0.07272 sec

Execution time - VM reallocation stDev: 0.03634 sec

Execution time - total mean: 0.13055 sec

Execution time - total stDev: 0.12894 sec

ESV is calculated in MS excel file by multiplying (Energy consumption: 120.85 kWh) and (SLA: 0.00076%)

ESV: 0.00091846= 0.91846 (x 0.001) // ESV Figure 3.4

Appendix G: Results of VM Placement Algorithms for All Workloads

Table G.1 shows mean, standard deviation and 95% confidence interval of ESV metric, energy consumption, SLAV metric, number of migrations, PDM metric, and SLATAH metric for all VM placement algorithms. Mean, standard deviation and 95% confidence interval have been calculated using exel formulas, which are AVERAGE(data range), STDEV(data range), [AVERAGE-CONFIDENCE(0.05,stdev,10), AVERAGE+ CONFIDENCE(0.05, stdev, 10)] , respectively.

Table G.1: Mean, standard deviation and 95% confidence interval of ESV metric, energy consumption, SLAV metric, number of migrations, PDM metric, and SLATAH metric for all VM placement algorithms

Parameter	Algorithm	Mean	Standard Deviation	95% confidence interval
ESV (×0.001)	FFD	2.5209	0.6826	[2.0978, 2.9440]
	PABFD	2.4131	0.7669	[1.9378, 2.8885]
	BFD	1.8383	0.5444	[1.5009, 2.1758]
	CPBFD_90 ,10,45	1.7511	0.5983	[1.3803, 2.1220]
	CPBFD_90 ,10,50	1.6710	0.4484	[1.3930, 1.9489]
	CPBFD_90 ,10,55	1.6754	0.4312	[1.4082, 1.9427]
	CPBFD_80 ,20,45	1.5772	0.4574	[1.2937, 1.8607]
	CPBFD_80 ,20,50	1.5211	0.4476	[1.2436, 1.7985]
	CPBFD_80 ,20,55	1.4882	0.4144	[1.2314, 1.7450]
	CPBFD_70 ,30,45	1.4827	0.4819	[1.1840, 1.7814]
	CPBFD_70 ,30,50	1.5084	0.3595	[1.2856, 1.7313]
	CPBFD_70 ,30,55	1.4617	0.4483	[1.1838, 1.7396]
	CPBFD_60	1.5339	0.5380	[1.2005, 1.8674]

	,40,45			
	CPBFD_60 ,40,50	1.5411	0.4830	[1.2417, 1.8405]
	CPBFD_60 ,40,55	1.5676	0.4917	[1.2628, 1.8723]
Energy (kWh)	FFD	119.877	23.085	[105.568, 134.185]
	PABFD	111.987	21.613	[98.5915, 125.382]
	BFD	114.587	21.527	[101.244, 127.929]
	CPBFD_90 ,10,45	114.928	22.306	[101.102, 128.753]
	CPBFD_90 ,10,50	114.935	21.866	[101.382, 128.487]
	CPBFD_90 ,10,55	114.645	21.420	[101.369, 127.921]
	CPBFD_80 ,20,45	115.028	21.896	[101.457, 128.599]
	CPBFD_80 ,20,50	114.860	21.955	[101.252, 128.467]
	CPBFD_80 ,20,55	114.625	21.702	[101.174, 128.075]
	CPBFD_70 ,30,45	115.584	22.354	[101.729, 129.438]
	CPBFD_70 ,30,50	115.576	22.129	[101.860, 129.291]
	CPBFD_70 ,30,55	115.660	21.892	[102.091, 129.228]
	CPBFD_60 ,40,45	116.511	22.175	[102.767, 130.254]
	CPBFD_60 ,40,50	116.795	22.604	[102.785, 130.805]
	CPBFD_60 ,40,55	116.314	22.148	[102.586, 130.041]
SLAV (×0.00001)	FFD	2.166	0.775	[1.6857, 2.6463]
	PABFD	2.201	0.867	[1.6634, 2.7386]
	BFD	1.671	0.712	[1.2299, 2.1121]
	CPBFD_90 ,10,45	1.583	0.721	[1.1363, 2.0297]
	CPBFD_90 ,10,50	1.51	0.591	[1.1436, 1.8764]
	CPBFD_90 ,10,55	1.515	0.565	[1.1651, 1.8649]
	CPBFD_80 ,20,45	1.421	0.567	[1.0695, 1.7725]
	CPBFD_80 ,20,50	1.37	0.561	[1.0221, 1.7179]
	CPBFD_80	1.343	0.532	[1.0131, 1.6729]

	,20,55			
	CPBFD_70 ,30,45	1.324	0.561	[0.9765, 1.6715]
	CPBFD_70 ,30,50	1.352	0.467	[1.0625, 1.6415]
	CPBFD_70 ,30,55	1.301	0.519	[0.9795, 1.6225]
	CPBFD_60 ,40,45	1.371	0.648	[0.9696, 1.7724]
	CPBFD_60 ,40,50	1.371	0.587	[1.0071, 1.7349]
	CPBFD_60 ,40,55	1.39	0.564	[1.0403, 1.7397]
Number of migrations (×1000)	FFD	13.8300	2.607	[12.2140, 15.4460]
	PABFD	14.4319	2.620	[12.8079, 16.0559]
	BFD	12.2683	2.101	[10.9658, 13.5708]
	CPBFD_90 ,10,45	12.0959	2.122	[10.7809, 13.4109]
	CPBFD_90 ,10,50	12.1616	2.199	[10.7989, 13.5243]
	CPBFD_90 ,10,55	12.1321	2.155	[10.7964, 13.4678]
	CPBFD_80 ,20,45	11.3135	2.049	[10.0436, 12.5834]
	CPBFD_80 ,20,50	11.3376	2.058	[10.0621, 12.6131]
	CPBFD_80 ,20,55	11.4559	1.964	[10.2388, 12.6730]
	CPBFD_70 ,30,45	10.7893	2.141	[9.4625, 12.1161]
	CPBFD_70 ,30,50	10.7598	2.080	[9.4709, 12.0487]
	CPBFD_70 ,30,55	10.8262	2.173	[9.4792, 12.1732]
	CPBFD_60 ,40,45	10.9872	2.098	[9.6872, 12.2872]
	CPBFD_60 ,40,50	11.0590	2.094	[9.7609, 12.3571]
	CPBFD_60 ,40,55	11.0372	2.320	[9.5994, 12.4750]
PDM	FFD	0.041%	0.006%	[0.00037, 0.00045]
	PABFD	0.046%	0.007%	[0.00042, 0.00050]
	BFD	0.034%	0.007%	[0.00030, 0.00038]
	CPBFD_90 ,10,45	0.033%	0.007%	[0.00029, 0.00037]
	CPBFD_90	0.032%	0.004%	[0.00029, 0.00035]

	,10,50			
	CPBFD_90 ,10,55	0.032%	0.004%	[0.00029, 0.00035]
	CPBFD_80 ,20,45	0.030%	0.007%	[0.00026, 0.00034]
	CPBFD_80 ,20,50	0.031%	0.006%	[0.00027, 0.00035]
	CPBFD_80 ,20,55	0.031%	0.006%	[0.00027, 0.00035]
	CPBFD_70 ,30,45	0.030%	0.007%	[0.00026, 0.00034]
	CPBFD_70 ,30,50	0.030%	0.007%	[0.00026, 0.00034]
	CPBFD_70 ,30,55	0.029%	0.006%	[0.00025, 0.00033]
	CPBFD_60 ,40,45	0.031%	0.006%	[0.00027, 0.00035]
	CPBFD_60 ,40,50	0.030%	0.005%	[0.00027, 0.00033]
	CPBFD_60 ,40,55	0.029%	0.006%	[0.00025, 0.00033]
SLATAH	FFD	S	0.980%	[0.04581, 0.05795]
	PABFD	4.695%	1.033%	[0.04055, 0.05335]
	BFD	4.918%	1.019%	[0.04287, 0.05549]
	CPBFD_90 ,10,45	4.698%	1.080%	[0.04029, 0.05367]
	CPBFD_90 ,10,50	4.556%	0.867%	[0.04018, 0.05094]
	CPBFD_90 ,10,55	4.607%	0.851%	[0.04080, 0.05134]
	CPBFD_80 ,20,45	4.506%	0.938%	[0.03925, 0.05087]
	CPBFD_80 ,20,50	4.334%	0.861%	[0.03800, 0.04868]
	CPBFD_80 ,20,55	4.289%	0.829%	[0.03775, 0.04803]
	CPBFD_70 ,30,45	4.225%	0.934%	[0.03646, 0.04804]
	CPBFD_70 ,30,50	4.448%	0.798%	[0.03953, 0.04943]
	CPBFD_70 ,30,55	4.284%	0.869%	[0.03745, 0.04823]
	CPBFD_60 ,40,45	4.329%	1.064%	[0.03670, 0.04988]
	CPBFD_60 ,40,50	4.402%	1.092%	[0.03725, 0.05079]
	CPBFD_60 ,40,55	4.552%	0.993%	[0.03936, 0.05168]

Table G.2 shows mean, standard deviation and 95% confidence interval of ESV metric, energy consumption, SLAV metric, number of migrations, PDM metric, and SLATAH metric for CPBFD and DCPBFD algorithms.

Table G.2: Mean, standard deviation and 95% confidence interval of ESV metric, energy consumption, SLAV metric, number of migrations, PDM metric, and SLATAH metric for CPBFD and DCPBFD algorithms

Parameter	**Algorithm**	**Mean**	**Standard Deviation**	**95% confidence interval**
ESV (×0.001)	CPBFD_80,20,55	1.4882	0.4144	[1.2314, 1.7450]
	CPBFD_75,25,55	1.4141	0.4340	[1.1451, 1.6831]
	CPBFD_70,30,55	1.4617	0.4483	[1.1838, 1.7396]
	DCPBFD_80,60,40,20,55,1	1.4418	0.4631	[1.1547, 1.7288]
	DCPBFD_80,60,40,20,55,0.75	1.3714	0.4481	[1.0937, 1.6491]
	DCPBFD_80,60,40,20,55,0.5	1.3820	0.3968	[1.1360, 1.6279]
Energy (kWh)	CPBFD_80,20,55	114.625	21.702	[101.174, 128.075]
	CPBFD_75,25,55	114.833	21.713	[101.375, 128.290]
	CPBFD_70,30,55	115.660	21.892	[102.091, 129.228]
	DCPBFD_80,60,40,20,55,1	115.532	21.937	[101.935, 129.128]
	DCPBFD_80,60,40,20,55,0.75	115.147	21.954	[101.540, 128.754]
	DCPBFD_80,60,40,20,55,0.5	114.582	21.840	[101.045, 128.118]
SLAV (×0.00001)	CPBFD_80,20,55	1.343	0.532	[1.0131, 1.6729]
	CPBFD_75,25,55	1.284	0.565	[0.9338, 1.6342]
	CPBFD_70,30,55	1.301	0.519	[0.9795, 1.6225]
	DCPBFD_80,60,40,20,55,1	1.292	0.556	[0.9472, 1.6368]
	DCPBFD_80,60,40,20,55,0.75	1.235	0.544	[0.8980, 1.5720]
	DCPBFD_80,60,40,20,55,0.5	1.249	0.494	[0.9430, 1.5550]
Number of migrations (×1000)	CPBFD_80,20,55	11.4559	1.964	[10.2388, 12.6730]
	CPBFD_75,25,55	10.8003	1.893	[9.6268, 11.9738]
	CPBFD_70,30,55	10.8262	2.173	[9.4792, 12.1732]
	DCPBFD_80,60,4	10.8027	2.071	[9.5190, 12.0864]

	0,20,55,1			
	DCPBFD_80,60,40,20,55,0.75	10.7265	2.061	[9.4490, 12.0040]
	DCPBFD_80,60,40,20,55,0.5	10.7402	2.054	[9.4673, 12.0131]
PDM	CPBFD_80,20,55	0.031%	0.006%	[0.00027, 0.00035]
	CPBFD_75,25,55	0.030%	0.007%	[0.00026, 0.00034]
	CPBFD_70,30,55	0.029%	0.006%	[0.00025, 0.00033]
	DCPBFD_80,60,40,20,55,1	0.030%	0.007%	[0.00026, 0.00034]
	DCPBFD_80,60,40,20,55,0.75	0.029%	0.006%	[0.00025, 0.00033]
	DCPBFD_80,60,40,20,55,0.5	0.029%	0.006%	[0.00025, 0.00033]
SLATAH	CPBFD_80,20,55	4.289%	0.829%	[0.03775, 0.04803]
	CPBFD_75,25,55	4.227%	0.877%	[0.03683, 0.04771]
	CPBFD_70,30,55	4.284%	0.869%	[0.03745, 0.04823]
	DCPBFD_80,60,40,20,55,1	4.196%	0.915%	[0.03629, 0.04763]
	DCPBFD_80,60,40,20,55,0.75	4.092%	0.916%	[0.03524, 0.04660]
	DCPBFD_80,60,40,20,55,0.5	4.198%	0.828%	[0.03685, 0.04711]

All results of VM placement algorithms for all workloads is shown in the Tables G.3 to G.38 and Figures G.1 to G.60. The results have been taken from output files (text files) and entered manually in MS excel file. Results in Tables G.3 to G.38 are copied manually from MS excel file.

Table G.3: ESV metric of FFD, PABFD and BFD algorithms for each workload (× 0.001)

Workload	**Algorithms**		
	FFD	PABFD	BFD
Workload 1	1.798272	1.81506	1.214121
Workload 2	1.650704	1.064816	1.08204
Workload 3	2.98058	2.60283	2.256525
Workload 4	3.049926	2.91141	1.983036
Workload 5	1.999326	1.841054	1.74108
Workload 6	3.261134	3.259512	2.033766
Workload 7	2.33345	2.41336	1.728135

Workload 8	2.097186	2.190824	1.476195
Workload 9	2.351648	2.304478	1.882368
Workload 10	3.686904	3.728047	2.986002
Average	2.520913	2.4131391	1.8383268

Table G.4: ESV metric of CPBFD_90,10,45, CPBFD_90,10,50 and CPBFD_90,10,55 algorithms for each workload (× 0.001)

Workload	**Algorithms**		
	CPBFD_90,10,45	CPBFD_90,10,50	CPBFD_90,10,55
Workload 1	1.14475	1.170499	1.305929
Workload 2	0.952621	1.109706	1.109706
Workload 3	2.358892	1.77822	1.92052
Workload 4	2.262513	2.000134	2.113408
Workload 5	1.309192	1.482852	1.348096
Workload 6	1.96728	1.87611	1.77595
Workload 7	1.587324	1.504618	1.475984
Workload 8	1.456299	1.447056	1.49676
Workload 9	1.62045	1.685222	1.652088
Workload 10	2.852174	2.655189	2.555853
Average	1.75115	1.6709606	1.6754294

Table G.5: ESV metric of CPBFD_80,20,45, CPBFD_80,20,50 and CPBFD_80,20,55 algorithms for each workload (× 0.001)

Workload	**Algorithms**		
	CPBFD_80,20,45	CPBFD_80,20,50	CPBFD_80,20,55
Workload 1	1.155456	1.295136	1.272106
Workload 2	0.824044	0.704838	0.750792
Workload 3	1.911924	1.740812	1.613059
Workload 4	2.00196	1.857086	1.7208
Workload 5	1.27212	1.2209	1.255331
Workload 6	1.65444	1.562496	1.582308
Workload 7	1.355528	1.437744	1.302336
Workload 8	1.413328	1.330545	1.429789
Workload 9	1.825644	1.66288	1.587159
Workload 10	2.357429	2.398322	2.368276
Average	1.577187	1.5210759	1.4881956

Table G.6: ESV metric of CPBFD_70,30,45, CPBFD_70,30,50 and CPBFD_70,30,55 algorithms for each workload (× 0.001)

Workload	**Algorithms**		
	CPBFD_70,30,45	CPBFD_70,30,50	CPBFD_70,30,55
Workload 1	0.929005	1.141724	0.895696
Workload 2	0.783261	0.968778	0.835107
Workload 3	1.65394	1.948009	1.537936
Workload 4	1.850535	1.880112	1.74986

Workload 5	1.185744	1.30872	1.156708
Workload 6	1.875072	1.600928	1.855573
Workload 7	1.2953	1.36581	1.451824
Workload 8	1.29677	1.408146	1.28112
Workload 9	1.574496	1.39328	1.543114
Workload 10	2.38248	2.068934	2.3101
Average	1.48266	1.5084441	1.4617038

Table G.7: ESV metric of CPBFD_60,40,45, CPBFD_60,40,50 and CPBFD_60,40,55 algorithms for each workload (× 0.001)

Workload	**Algorithms**		
	CPBFD_60,40,45	CPBFD_60,40,50	CPBFD_60,40,55
Workload 1	0.97944	1.005777	1.134414
Workload 2	0.804672	0.745605	0.82377
Workload 3	1.97664	1.93268	2.018016
Workload 4	1.963764	1.987459	2.14347
Workload 5	1.207857	1.334	1.136447
Workload 6	1.698367	1.613692	1.792832
Workload 7	1.265753	1.43209	1.411668
Workload 8	1.321216	1.379395	1.444836
Workload 9	1.5225	1.57896	1.42545
Workload 10	2.599092	2.401331	2.344848
Average	1.53393	1.5410989	1.5675751

Table G.8: ESV metric of CPBFD_75,25,55, DCPBFD_80,60,40,20,55,1, DCPBFD_80,60,40,20,55,0.75 and DCPBFD_80,60,40,20,55,0.5 algorithms for each workload (× 0.001)

Workload	**Algorithms**			
	CPBFD_75,25,55	DCPBFD_80,60,40,20,55,1	DCPBFD_80,60, 40,20,55,0.75	DCPBFD_80,60, 40,20,55,0.5
Workload 1	1.07919	0.94068	0.91846	1.05864
Workload 2	0.81036	0.741522	0.7168	0.77865
Workload 3	1.788231	1.720927	1.642842	1.768025
Workload 4	1.566367	1.835552	1.8075	1.738115
Workload 5	1.207944	1.21011	1.046925	1.005792
Workload 6	1.525808	1.581585	1.480843	1.473836
Workload 7	1.198212	1.40346	1.306839	1.275912
Workload 8	1.157199	1.207008	1.18484	1.213926
Workload 9	1.451622	1.426328	1.36962	1.43154
Workload 10	2.35572	2.350392	2.23941	2.075122
Average	1.414065	1.4417564	1.3714079	1.3819558

Table G.9: The energy consumption of FFD, PABFD and BFD algorithms for each workload (kWh)

Workload	**Algorithms**		
	FFD	PABFD	BFD

Workload 1	124.88	116.35	120.21
Workload 2	93.79	87.28	90.17
Workload 3	104.95	98.22	100.29
Workload 4	126.03	118.35	119.46
Workload 5	110.46	103.43	105.52
Workload 6	170.74	159.78	161.41
Workload 7	133.34	124.40	128.01
Workload 8	130.26	121.04	124.05
Workload 9	113.06	105.71	109.44
Workload 10	91.26	85.31	87.31
Average	119.877	111.987	114.587

Table G.10: The energy consumption of CPBFD_90,10,45, CPBFD_90,10,50 and CPBFD_90,10,55 algorithms for each workload (kWh)

Workload	**Algorithms**		
	CPBFD_90,10,45	CPBFD_90,10,50	CPBFD_90,10,55
Workload 1	120.5	120.67	119.81
Workload 2	89.03	90.22	90.22
Workload 3	101.24	99.9	101.08
Workload 4	120.99	120.49	120.08
Workload 5	105.58	106.68	105.32
Workload 6	163.94	163.14	161.45
Workload 7	128.01	127.51	127.24
Workload 8	124.47	123.68	124.73
Workload 9	108.03	109.43	108.69
Workload 10	87.49	87.63	87.83
Average	114.928	114.935	114.645

Table G.11: The energy consumption of CPBFD_80,20,45, CPBFD_80,20,50 and CPBFD_80,20,55 algorithms for each workload (kWh)

Workload	**Algorithms**		
	CPBFD_80,20,45	CPBFD_80,20,50	CPBFD_80,20,55
Workload 1	120.36	119.92	120.01
Workload 2	89.57	89.22	89.38
Workload 3	101.16	101.21	100.19
Workload 4	120.60	120.59	119.50
Workload 5	106.01	105.25	105.49
Workload 6	162.20	162.76	161.46
Workload 7	127.88	128.37	127.68
Workload 8	126.19	124.35	126.53
Workload 9	109.32	109.40	107.97
Workload 10	86.99	87.53	88.04
Average	115.028	114.860	114.625

Table G.12: The energy consumption of CPBFD_70,30,45, CPBFD_70,30,50 and CPBFD_70,30,55 algorithms for each workload (kWh)

Workload	**Algorithms**		
	CPBFD_70,30,45	CPBFD_70,30,50	CPBFD_70,30,55
Workload 1	120.65	121.46	121.04
Workload 2	90.03	90.54	91.77
Workload 3	100.85	101.99	101.18
Workload 4	120.95	120.52	120.68
Workload 5	105.87	106.40	106.12
Workload 6	164.48	163.36	164.21
Workload 7	129.53	128.85	128.48
Workload 8	125.90	126.86	125.60
Workload 9	109.34	108.85	108.67
Workload 10	88.24	86.93	88.85
Average	115.584	115.576	115.660

Table G.13: The energy consumption of CPBFD_60,40,45, CPBFD_60,40,50 and CPBFD_60,40,55 algorithms for each workload (kWh)

Workload	**Algorithms**		
	CPBFD_60,40,45	CPBFD_60,40,50	CPBFD_60,40,55
Workload 1	122.43	124.17	121.98
Workload 2	91.44	92.05	91.53
Workload 3	102.95	101.72	101.92
Workload 4	121.22	121.93	121.10
Workload 5	106.89	106.72	106.21
Workload 6	164.89	166.36	164.48
Workload 7	130.49	130.19	130.71
Workload 8	127.04	126.55	126.74
Workload 9	108.75	109.65	109.65
Workload 10	89.01	88.61	88.82
Average	116.511	116.795	116.314

Table G.14: The energy consumption of CPBFD_75,25,55, DCPBFD_80,60,40,20,55,1, DCPBFD_80,60,40,20,55,0.75 and DCPBFD_80,60,40,20,55,0.5 algorithms for each workload (kWh)

Workload	**Algorithms**			
	CPBFD_75,25,55	DCPBFD_80,60,40,20,55,1	DCPBFD_80,60, 40,20,55,0.75	DCPBFD_80,60, 40,20,55,0.5
Workload 1	119.91	120.6	120.85	120.3
Workload 2	90.04	89.34	89.6	89.5
Workload 3	101.03	101.83	101.41	101.03
Workload 4	119.57	120.76	120.5	119.87
Workload 5	105.96	106.15	105.75	104.77
Workload 6	162.32	163.05	162.73	161.96
Workload 7	128.84	129.95	129.39	128.88
Workload 8	124.43	125.73	124.72	123.87

Workload 9	108.33	108.88	108.7	108.45
Workload 10	87.90	89.03	87.82	87.19
Average	114.833	115.532	115.147	114.582

Table G.15: SLAV metric of FFD, PABFD and BFD algorithms for each workload (× 0.00001)

Workload	**Algorithms**		
	FFD	PABFD	BFD
Workload 1	1.44	1.56	1.01
Workload 2	1.76	1.22	1.2
Workload 3	2.84	2.65	2.25
Workload 4	2.42	2.46	1.66
Workload 5	1.81	1.78	1.65
Workload 6	1.91	2.04	1.26
Workload 7	1.75	1.94	1.35
Workload 8	1.61	1.81	1.19
Workload 9	2.08	2.18	1.72
Workload 10	4.04	4.37	3.42
Average	2.166	2.201	1.671

Table G.16: SLAV metric of CPBFD_90,10,45, CPBFD_90,10,50 and CPBFD_90,10,55 algorithms for each workload (× 0.00001)

Workload	**Algorithms**		
	CPBFD_90,10,45	CPBFD_90,10,50	CPBFD_90,10,55
Workload 1	0.95	0.97	1.09
Workload 2	1.07	1.23	1.23
Workload 3	2.33	1.78	1.9
Workload 4	1.87	1.66	1.76
Workload 5	1.24	1.39	1.28
Workload 6	1.2	1.15	1.1
Workload 7	1.24	1.18	1.16
Workload 8	1.17	1.17	1.2
Workload 9	1.5	1.54	1.52
Workload 10	3.26	3.03	2.91
Average	1.583	1.51	1.515

Table G.17: SLAV metric of CPBFD_80,20,45, CPBFD_80,20,50 and CPBFD_80,20,55 algorithms for each workload (× 0.00001)

Workload	**Algorithms**		
	CPBFD_80,20,45	CPBFD_80,20,50	CPBFD_80,20,55
Workload 1	0.96	1.08	1.06
Workload 2	0.92	0.79	0.84
Workload 3	1.89	1.72	1.61
Workload 4	1.66	1.54	1.44
Workload 5	1.2	1.16	1.19

Workload 6	1.02	0.96	0.98
Workload 7	1.06	1.12	1.02
Workload 8	1.12	1.07	1.13
Workload 9	1.67	1.52	1.47
Workload 10	2.71	2.74	2.69
Average	1.421	1.37	1.343

Table G.18: SLAV metric of CPBFD_70,30,45, CPBFD_70,30,50 and CPBFD_70,30,55 algorithms for each workload (× 0.00001)

Workload	**Algorithms**		
	CPBFD_70,30,45	CPBFD_70,30,50	CPBFD_70,30,55
Workload 1	0.77	0.94	0.74
Workload 2	0.87	1.07	0.91
Workload 3	1.64	1.91	1.52
Workload 4	1.53	1.56	1.45
Workload 5	1.12	1.23	1.09
Workload 6	1.14	0.98	1.13
Workload 7	1	1.06	1.13
Workload 8	1.03	1.11	1.02
Workload 9	1.44	1.28	1.42
Workload 10	2.7	2.38	2.6
Average	1.324	1.352	1.301

Table G.19: SLAV metric of CPBFD_60,40,45, CPBFD_60,40,50 and CPBFD_60,40,55 algorithms for each workload (× 0.00001)

Workload	**Algorithms**		
	CPBFD_60,40,45	CPBFD_60,40,50	CPBFD_60,40,55
Workload 1	0.8	0.81	0.93
Workload 2	0.88	0.81	0.9
Workload 3	1.92	1.9	1.98
Workload 4	1.62	1.63	1.77
Workload 5	1.13	1.25	1.07
Workload 6	1.03	0.97	1.09
Workload 7	0.97	1.1	1.08
Workload 8	1.04	1.09	1.14
Workload 9	1.4	1.44	1.3
Workload 10	2.92	2.71	2.64
Average	1.371	1.371	1.39

Table G.20: SLAV metric of CPBFD_75,25,55, DCPBFD_80,60,40,20,55,1, DCPBFD_80,60,40,20,55,0.75 and DCPBFD_80,60,40,20,55,0.5 algorithms for each workload (× 0.00001)

Workload	**Algorithms**			
	CPBFD_75,25,55	DCPBFD_80,60,40,20,55,1	DCPBFD_80,60, 40,20,55,0.75	DCPBFD_80,60, 40,20,55,0.5

Workload 1	0.9	0.78	0.76	0.88
Workload 2	0.9	0.83	0.8	0.87
Workload 3	1.77	1.69	1.62	1.75
Workload 4	1.31	1.52	1.5	1.45
Workload 5	1.14	1.14	0.99	0.96
Workload 6	0.94	0.97	0.91	0.91
Workload 7	0.93	1.08	1.01	0.99
Workload 8	0.93	0.96	0.95	0.98
Workload 9	1.34	1.31	1.26	1.32
Workload 10	2.68	2.64	2.55	2.38
Average	1.284	1.292	1.235	1.249

Table G.21: The number of migrations of FFD, PABFD and BFD algorithms for each workload (× 1000)

Workload	**Algorithms**		
	FFD	PABFD	BFD
Workload 1	12.241	12.58	10.454
Workload 2	9.556	9.822	8.598
Workload 3	12.328	13.1	11.195
Workload 4	16.99	17.1	14.564
Workload 5	13.054	13.903	11.956
Workload 6	18.641	19.121	15.799
Workload 7	15.189	16.602	13.641
Workload 8	14.612	14.736	13.333
Workload 9	12.888	13.743	11.638
Workload 10	12.801	13.612	11.505
Average	13.8300	14.4319	12.2683

Table G.22: The number of migrations of CPBFD_90,10,45, CPBFD_90,10,50 and CPBFD_90,10,55 algorithms for each workload (× 1000)

Workload	**Algorithms**		
	CPBFD_90,10,45	CPBFD_90,10,50	CPBFD_90,10,55
Workload 1	10.651	10.893	10.791
Workload 2	8.3	8.309	8.309
Workload 3	11.164	11.055	10.952
Workload 4	14.387	14.471	14.582
Workload 5	11.431	11.618	11.245
Workload 6	15.651	16.092	15.72
Workload 7	13.546	13.491	13.472
Workload 8	13.255	13.25	13.358
Workload 9	11.462	11.317	11.655
Workload 10	11.112	11.12	11.237
Average	12.0959	12.1616	12.1321

Table G.23: The number of migrations of CPBFD_80,20,45, CPBFD_80,20,50 and CPBFD_80,20,55 algorithms for each workload (× 1000)

Workload	**Algorithms**		
	CPBFD_80,20,45	CPBFD_80,20,50	CPBFD_80,20,55
Workload 1	10.031	9.855	10.288
Workload 2	7.522	7.847	7.808
Workload 3	10.382	10.134	10.363
Workload 4	13.791	14.061	13.863
Workload 5	10.806	10.803	10.901
Workload 6	14.591	14.576	14.45
Workload 7	12.633	12.532	12.589
Workload 8	12.19	12.504	12.663
Workload 9	10.921	10.602	11.147
Workload 10	10.268	10.462	10.487
Average	11.3135	11.3376	11.4559

Table G.24: The number of migrations of CPBFD_70,30,45, CPBFD_70,30,50 and CPBFD_70,30,55 algorithms for each workload (× 1000)

Workload	**Algorithms**		
	CPBFD_70,30,45	CPBFD_70,30,50	CPBFD_70,30,55
Workload 1	9.229	9.544	8.955
Workload 2	6.877	7.11	6.984
Workload 3	9.603	9.763	9.588
Workload 4	13.504	13.858	13.713
Workload 5	10.512	10.351	10.471
Workload 6	14.34	13.953	14.321
Workload 7	11.816	11.939	11.808
Workload 8	11.335	11.273	11.654
Workload 9	10.717	10.053	10.382
Workload 10	9.96	9.754	10.386
Average	10.7893	10.7598	10.8262

Table G.25: The number of migrations of CPBFD_60,40,45, CPBFD_60,40,50 and CPBFD_60,40,55 algorithms for each workload (× 1000)

Workload	**Algorithms**		
	CPBFD_60,40,45	CPBFD_60,40,50	CPBFD_60,40,55
Workload 1	8.982	9.398	9.297
Workload 2	7.221	7.057	6.869
Workload 3	10.205	10.118	9.884
Workload 4	14.005	14.078	14.554
Workload 5	10.264	10.605	10.335
Workload 6	14.137	13.834	14.223
Workload 7	11.652	12.206	12.355
Workload 8	11.828	12.062	12.087
Workload 9	10.794	10.822	10.354
Workload 10	10.784	10.41	10.414

Average	10.9872	11.0590	11.0372

Table G.26: The number of migrations of CPBFD_75,25,55, DCPBFD_80,60,40,20,55,1, DCPBFD_80,60,40,20,55,0.75 and DCPBFD_80,60,40,20,55,0.5 algorithms for each workload (× 1000)

Workload	**Algorithms**			
	CPBFD_75,25,55	DCPBFD_80,60,40,20,55,1	DCPBFD_80,60, 40,20,55,0.75	DCPBFD_80,60, 40,20,55,0.5
Workload 1	9.551	9.05	9.275	9.237
Workload 2	7.438	7.171	7.056	7.309
Workload 3	9.734	9.668	9.43	9.553
Workload 4	13.265	13.734	13.26	13.361
Workload 5	10.369	10.553	10.102	10.248
Workload 6	14.057	13.934	13.949	14.308
Workload 7	11.389	11.688	12.031	11.68
Workload 8	11.56	11.817	11.892	11.627
Workload 9	10.253	10.396	10.244	10.149
Workload 10	10.387	10.016	10.026	9.93
Average	10.8003	10.8027	10.7265	10.7402

Table G.27: PDM metric of FFD, PABFD and BFD algorithms for each workload

Workload	**Algorithms**		
	FFD	PABFD	BFD
Workload 1	0.03%	0.04%	0.03%
Workload 2	0.04%	0.04%	0.03%
Workload 3	0.05%	0.05%	0.04%
Workload 4	0.04%	0.05%	0.04%
Workload 5	0.04%	0.04%	0.03%
Workload 6	0.04%	0.05%	0.03%
Workload 7	0.04%	0.04%	0.03%
Workload 8	0.04%	0.04%	0.03%
Workload 9	0.04%	0.05%	0.03%
Workload 10	0.05%	0.06%	0.05%
Average	0.041%	0.046%	0.034%

Table G.28: PDM metric of CPBFD_90,10,45, CPBFD_90,10,50 and CPBFD_90,10,55 algorithms for each workload

Workload	**Algorithms**		
	CPBFD_90,10,45	CPBFD_90,10,50	CPBFD_90,10,55
Workload 1	0.03%	0.03%	0.03%
Workload 2	0.03%	0.03%	0.03%
Workload 3	0.04%	0.04%	0.04%
Workload 4	0.03%	0.03%	0.03%
Workload 5	0.03%	0.03%	0.03%
Workload 6	0.03%	0.03%	0.03%
Workload 7	0.03%	0.03%	0.03%

Workload 8	0.03%	0.03%	0.03%
Workload 9	0.03%	0.03%	0.03%
Workload 10	0.05%	0.04%	0.04%
Average	0.033%	0.032%	0.032%

Table G.29: PDM metric of CPBFD_80,20,45, CPBFD_80,20,50 and CPBFD_80,20,55 algorithms for each workload

Workload	**Algorithms**		
	CPBFD_80,20,45	CPBFD_80,20,50	CPBFD_80,20,55
Workload 1	0.02%	0.03%	0.03%
Workload 2	0.02%	0.02%	0.02%
Workload 3	0.04%	0.04%	0.04%
Workload 4	0.03%	0.03%	0.03%
Workload 5	0.03%	0.03%	0.03%
Workload 6	0.03%	0.03%	0.03%
Workload 7	0.03%	0.03%	0.03%
Workload 8	0.03%	0.03%	0.03%
Workload 9	0.03%	0.03%	0.03%
Workload 10	0.04%	0.04%	0.04%
Average	0.030%	0.031%	0.031%

Table G.30: PDM metric of CPBFD_70,30,45, CPBFD_70,30,50 and CPBFD_70,30,55 algorithms for each workload

Workload	**Algorithms**		
	CPBFD_70,30,45	CPBFD_70,30,50	CPBFD_70,30,55
Workload 1	0.02%	0.02%	0.02%
Workload 2	0.02%	0.02%	0.02%
Workload 3	0.04%	0.04%	0.03%
Workload 4	0.03%	0.03%	0.03%
Workload 5	0.03%	0.03%	0.03%
Workload 6	0.03%	0.03%	0.03%
Workload 7	0.03%	0.03%	0.03%
Workload 8	0.03%	0.03%	0.03%
Workload 9	0.03%	0.03%	0.03%
Workload 10	0.04%	0.04%	0.04%
Average	0.030%	0.030%	0.029%

Table G.31: PDM metric of CPBFD_60,40,45, CPBFD_60,40,50 and CPBFD_60,40,55 algorithms for each workload

Workload	**Algorithms**		
	CPBFD_60,40,45	CPBFD_60,40,50	CPBFD_60,40,55
Workload 1	0.02%	0.03%	0.02%
Workload 2	0.03%	0.02%	0.02%
Workload 3	0.04%	0.03%	0.03%
Workload 4	0.03%	0.03%	0.03%

Workload 5	0.03%	0.03%	0.03%
Workload 6	0.03%	0.03%	0.03%
Workload 7	0.03%	0.03%	0.03%
Workload 8	0.03%	0.03%	0.03%
Workload 9	0.03%	0.03%	0.03%
Workload 10	0.04%	0.04%	0.04%
Average	0.031%	0.030%	0.029%

Table G.32: PDM metric of CPBFD_75,25,55, DCPBFD_80,60,40,20,55,1, DCPBFD_80,60,40,20,55,0.75 and DCPBFD_80,60,40,20,55,0.5 algorithms for each workload

Workload	**Algorithms**			
	CPBFD_75,25,55	DCPBFD_80,60,40,20,55,1	DCPBFD_80,60, 40,20,55,0.75	DCPBFD_80,60, 40,20,55,0.5
Workload 1	0.02%	0.02%	0.02%	0.02%
Workload 2	0.02%	0.03%	0.02%	0.02%
Workload 3	0.04%	0.04%	0.03%	0.03%
Workload 4	0.03%	0.03%	0.03%	0.03%
Workload 5	0.03%	0.03%	0.03%	0.03%
Workload 6	0.03%	0.03%	0.03%	0.03%
Workload 7	0.03%	0.03%	0.03%	0.03%
Workload 8	0.03%	0.02%	0.03%	0.03%
Workload 9	0.03%	0.03%	0.03%	0.03%
Workload 10	0.04%	0.04%	0.04%	0.04%
Average	0.030%	0.030%	0.029%	0.029%

Table G.33: SLATAH metric of FFD, PABFD and BFD algorithms for each workload

Workload	**Algorithms**		
	FFD	PABFD	BFD
Workload 1	4.22%	3.88%	3.86%
Workload 2	4.97%	3.44%	4.42%
Workload 3	6.05%	5.27%	5.93%
Workload 4	5.77%	5.09%	4.82%
Workload 5	4.57%	4.12%	5.04%
Workload 6	4.98%	4.41%	4.41%
Workload 7	4.44%	4.43%	4.31%
Workload 8	4.52%	4.35%	4.16%
Workload 9	4.92%	4.76%	4.92%
Workload 10	7.44%	7.20%	7.31%
Average	5.188%	4.695%	4.918%

Table G.34: SLATAH metric of CPBFD_90,10,45, CPBFD_90,10,50 and CPBFD_90,10,55 algorithms for each workload

Workload	**Algorithms**		
	CPBFD_90,10,45	CPBFD_90,10,50	CPBFD_90,10,55

Workload 1	3.65%	3.61%	4.07%
Workload 2	4.11%	4.69%	4.69%
Workload 3	5.93%	4.68%	5.08%
Workload 4	5.19%	4.89%	5.12%
Workload 5	3.93%	4.32%	4.06%
Workload 6	4.14%	4.01%	3.91%
Workload 7	4.14%	3.98%	3.86%
Workload 8	4.20%	4.13%	4.11%
Workload 9	4.58%	4.49%	4.53%
Workload 10	7.11%	6.76%	6.64%
Average	4.698%	4.556%	4.607%

Table G.35: SLATAH metric of CPBFD_80,20,45, CPBFD_80,20,50 and CPBFD_80,20,55 algorithms for each workload

Workload	**Algorithms**		
	CPBFD_80,20,45	CPBFD_80,20,50	CPBFD_80,20,55
Workload 1	3.93%	4.21%	4.03%
Workload 2	3.79%	3.27%	3.40%
Workload 3	5.28%	4.71%	4.59%
Workload 4	5.02%	4.70%	4.58%
Workload 5	3.88%	3.88%	4.03%
Workload 6	3.87%	3.71%	3.74%
Workload 7	3.72%	3.92%	3.61%
Workload 8	4.05%	3.90%	4.04%
Workload 9	4.91%	4.66%	4.53%
Workload 10	6.61%	6.38%	6.34%
Average	4.506%	4.334%	4.289%

Table G.36: SLATAH metric of CPBFD_70,30,45, CPBFD_70,30,50 and CPBFD_70,30,55 algorithms for each workload

Workload	**Algorithms**		
	CPBFD_70,30,45	CPBFD_70,30,50	CPBFD_70,30,55
Workload 1	3.21%	3.79%	3.24%
Workload 2	3.53%	4.46%	3.68%
Workload 3	4.62%	5.19%	4.45%
Workload 4	4.75%	4.74%	4.55%
Workload 5	3.73%	4.23%	3.66%
Workload 6	4.26%	3.86%	4.29%
Workload 7	3.51%	3.65%	4.05%
Workload 8	3.78%	4.10%	3.97%
Workload 9	4.42%	4.16%	4.51%
Workload 10	6.44%	6.30%	6.44%
Average	4.225%	4.448%	4.284%

Table G.37: SLATAH metric of CPBFD_60,40,45, CPBFD_60,40,50 and CPBFD_60,40,55 algorithms for each workload

Workload	**Algorithms**		
	CPBFD_60,40,45	CPBFD_60,40,50	CPBFD_60,40,55
Workload 1	3.35%	3.23%	3.89%
Workload 2	3.50%	3.38%	3.67%
Workload 3	5.30%	5.62%	5.80%
Workload 4	4.77%	4.78%	5.27%
Workload 5	3.84%	4.13%	3.81%
Workload 6	3.89%	3.82%	4.23%
Workload 7	3.54%	3.93%	3.79%
Workload 8	3.85%	3.86%	4.25%
Workload 9	4.45%	4.47%	4.22%
Workload 10	6.80%	6.80%	6.59%
Average	4.329%	4.402%	4.552%

Table G.38: SLATAH metric of CPBFD_75,25,55, DCPBFD_80,60,40,20,55,1, DCPBFD_80,60,40,20,55,0.75 and DCPBFD_80,60,40,20,55,0.5 algorithms for each workload

Workload	**Algorithms**			
	CPBFD_75,25,55	DCPBFD_80,60,40,20,55,1	DCPBFD_80,60,40,20,55,0.75	DCPBFD_80,60,40,20,55,0.5
Workload 1	3.78%	3.37%	3.28%	3.77%
Workload 2	3.77%	3.29%	3.35%	3.64%
Workload 3	4.99%	4.72%	4.72%	5.07%
Workload 4	4.24%	4.63%	4.72%	4.53%
Workload 5	3.99%	3.90%	3.61%	3.60%
Workload 6	3.73%	3.87%	3.60%	3.64%
Workload 7	3.38%	3.80%	3.45%	3.51%
Workload 8	3.67%	3.68%	3.75%	3.92%
Workload 9	4.35%	4.28%	4.22%	4.21%
Workload 10	6.37%	6.42%	6.22%	6.09%
Average	4.227%	4.20%	4.09%	4.20%

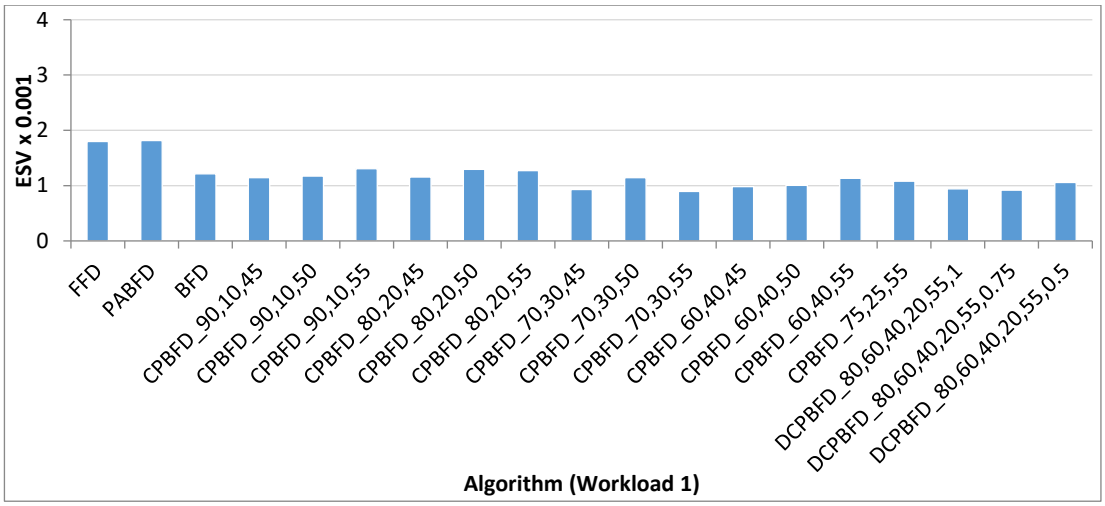

Figure G.1: ESV metric of VM placement algorithms for wokload 1

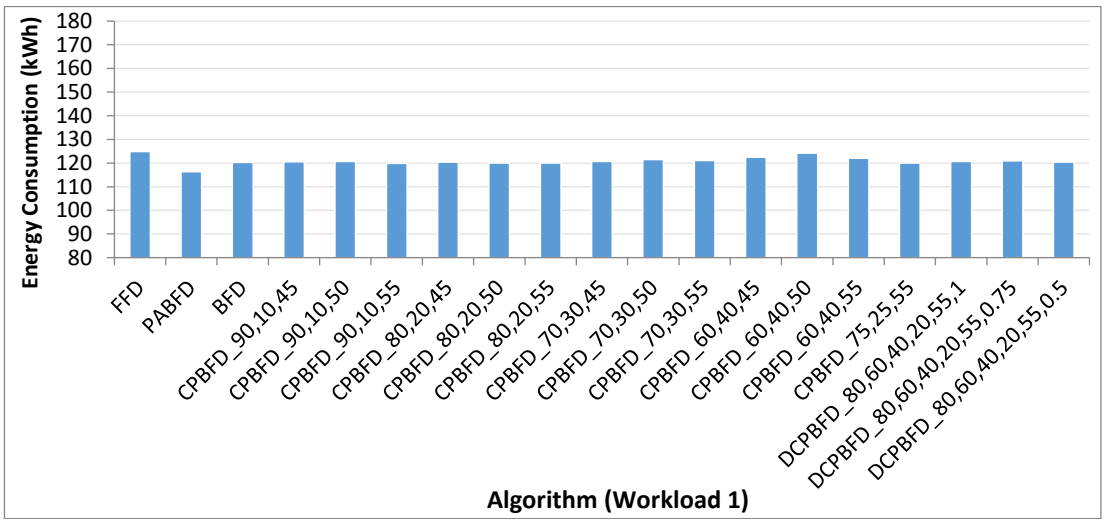

Figure G.2: The energy consumption of VM placement algorithms for wokload 1

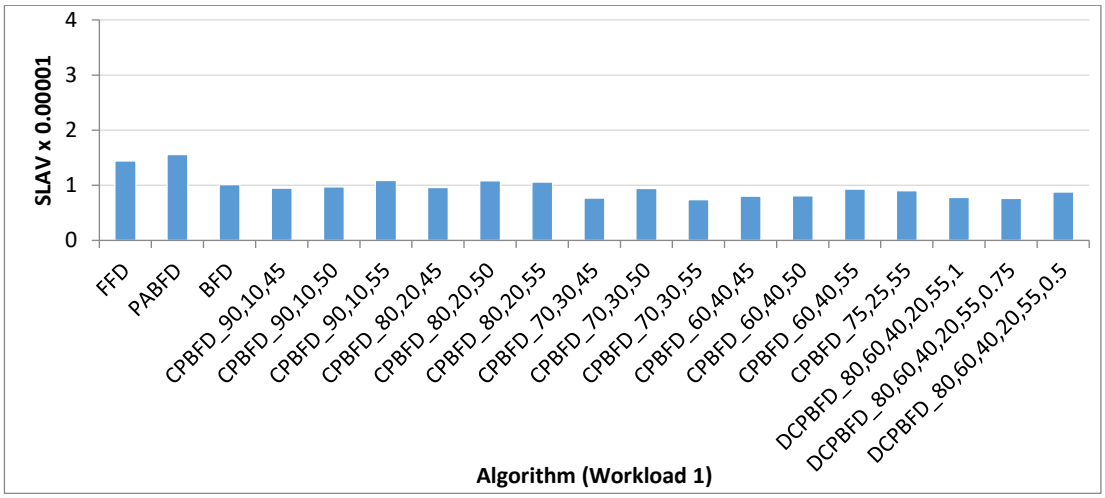

Figure G.3: SLAV metric of VM placement algorithms for wokload 1

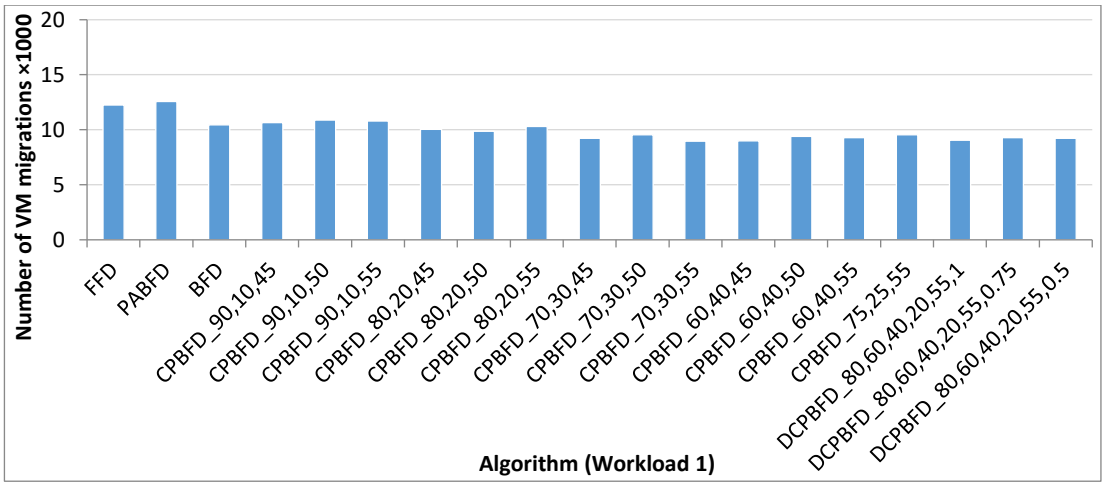

Figure G.4: Number of VM migrations of VM placement algorithms for wokload 1

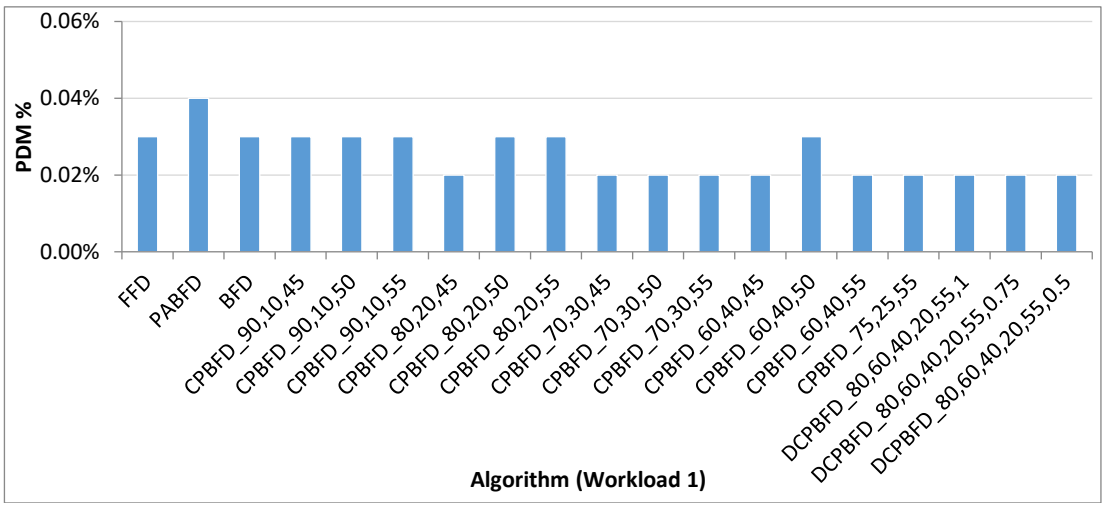

Figure G.5: PDM metric of VM placement algorithms for wokload 1

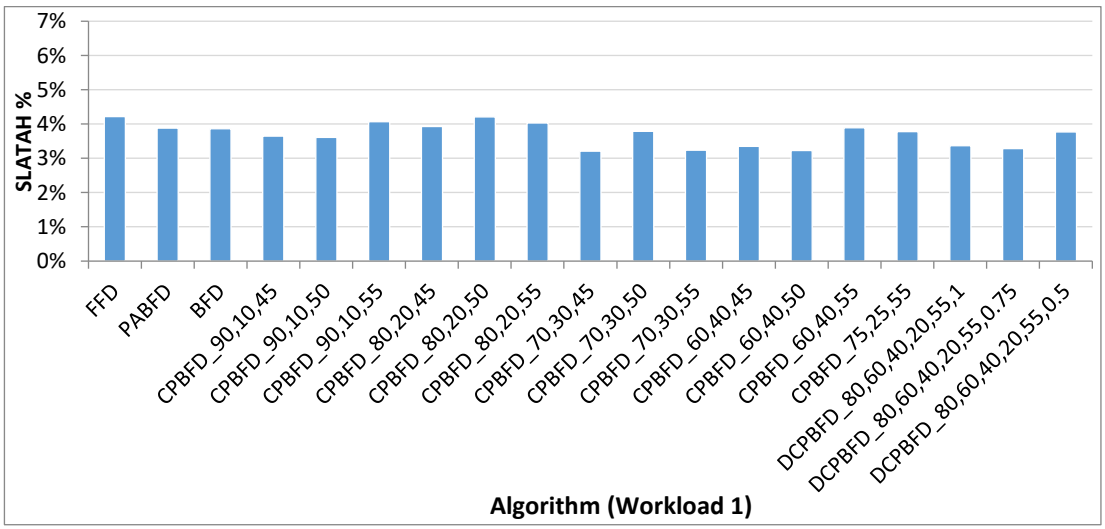

Figure G.6: SLATAH metric of VM placement algorithms for wokload 1

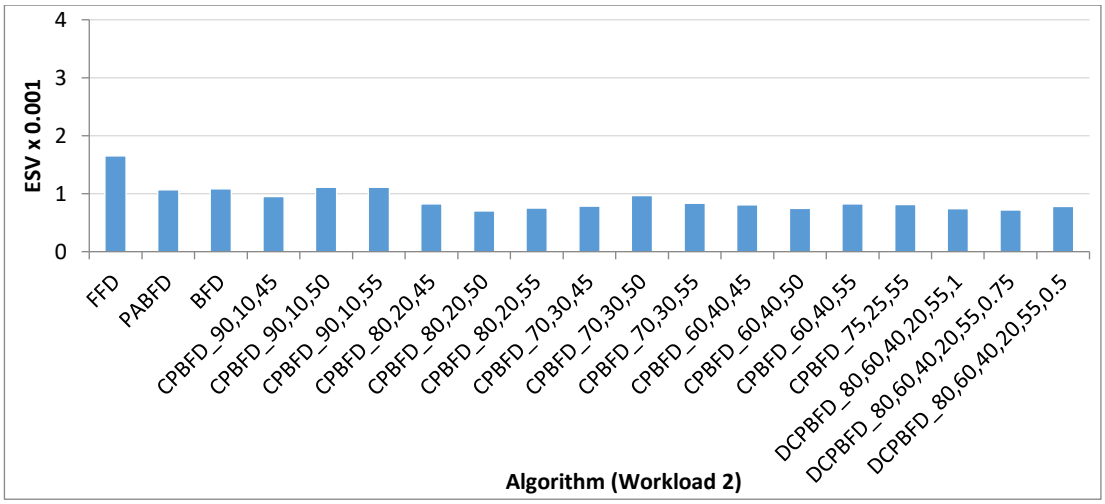

Figure G.7: ESV metric of VM placement algorithms for wokload 2

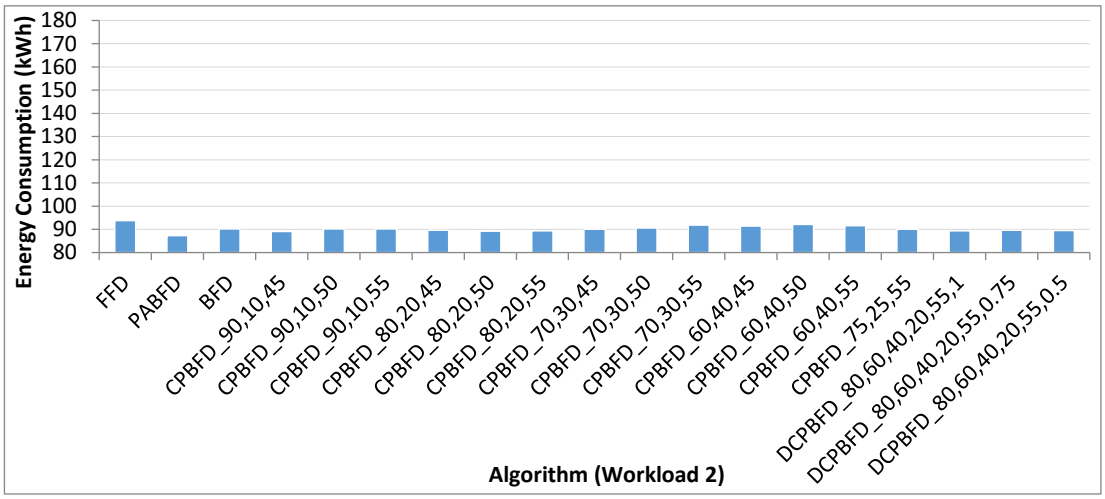

Figure G.8: The energy consumption of VM placement algorithms for wokload 2

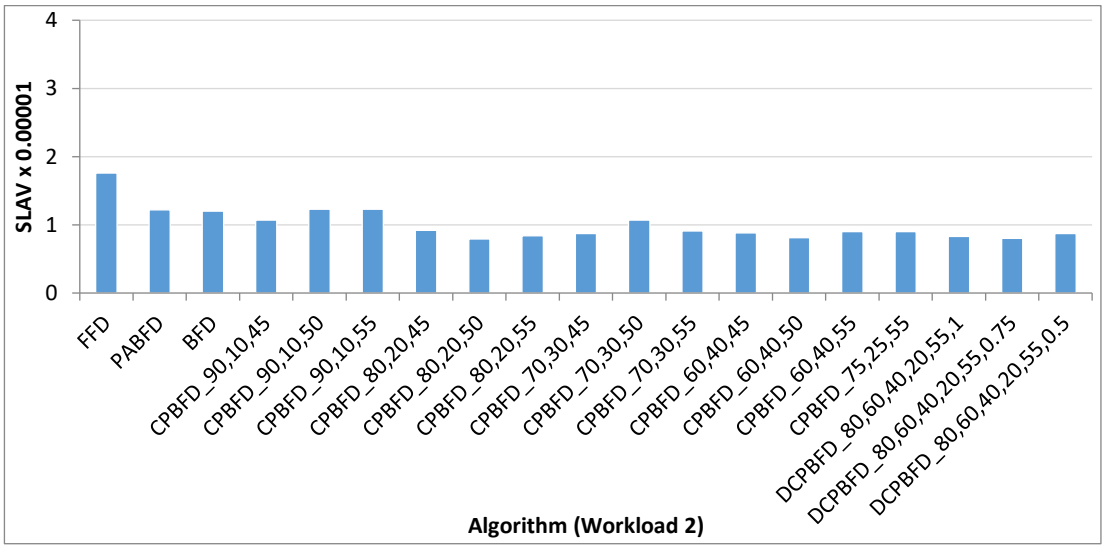

Figure G.9: SLAV metric of VM placement algorithms for wokload 2

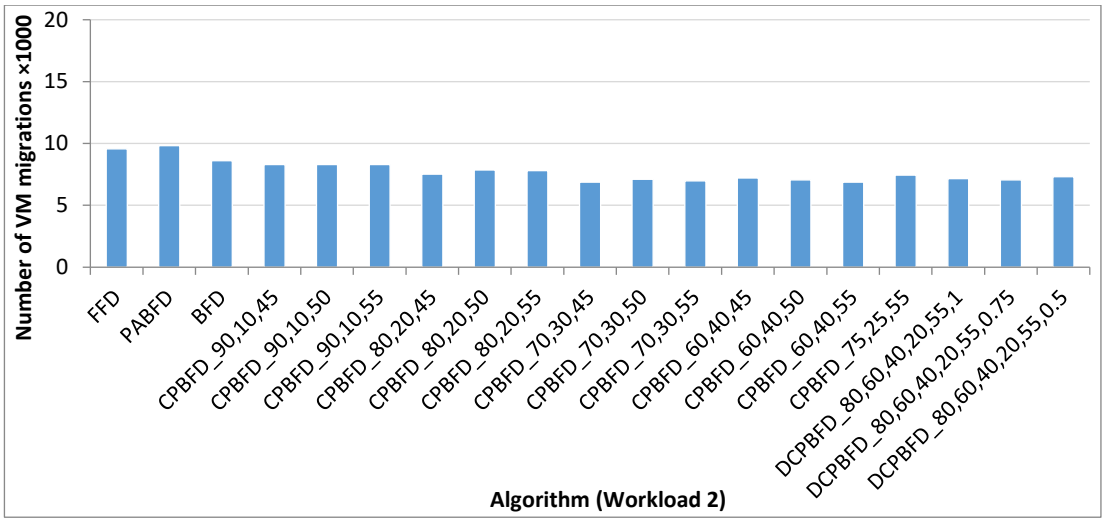

Figure G.10: Number of VM migrations of VM placement algorithms for wokload 2

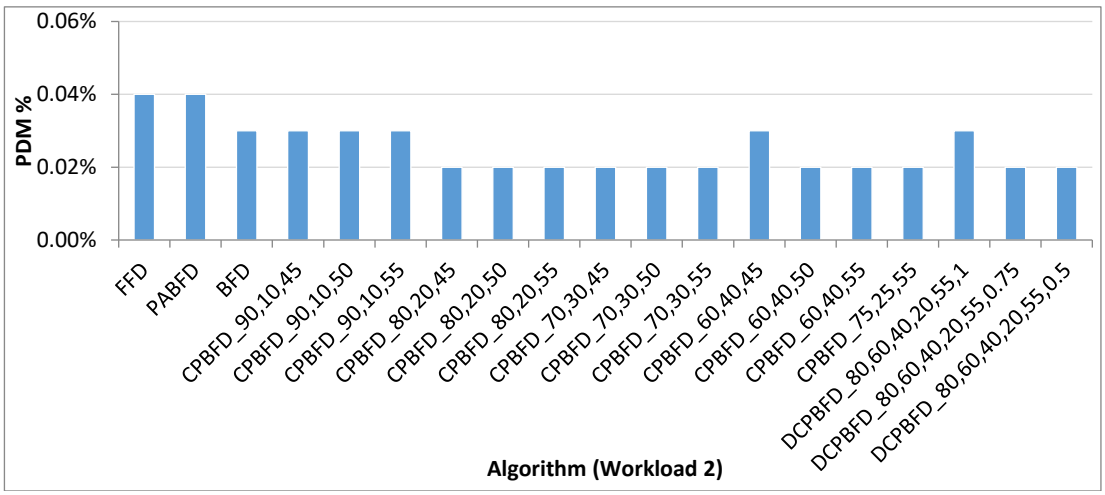

Figure G.11: PDM metric of VM placement algorithms for wokload 2

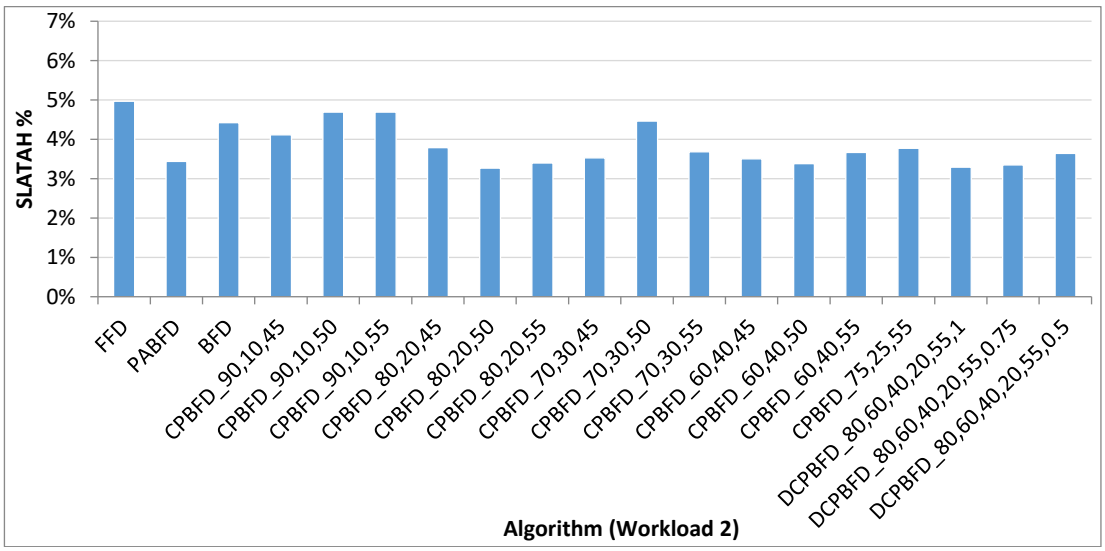

Figure G.12: SLATAH metric of VM placement algorithms for wokload 2

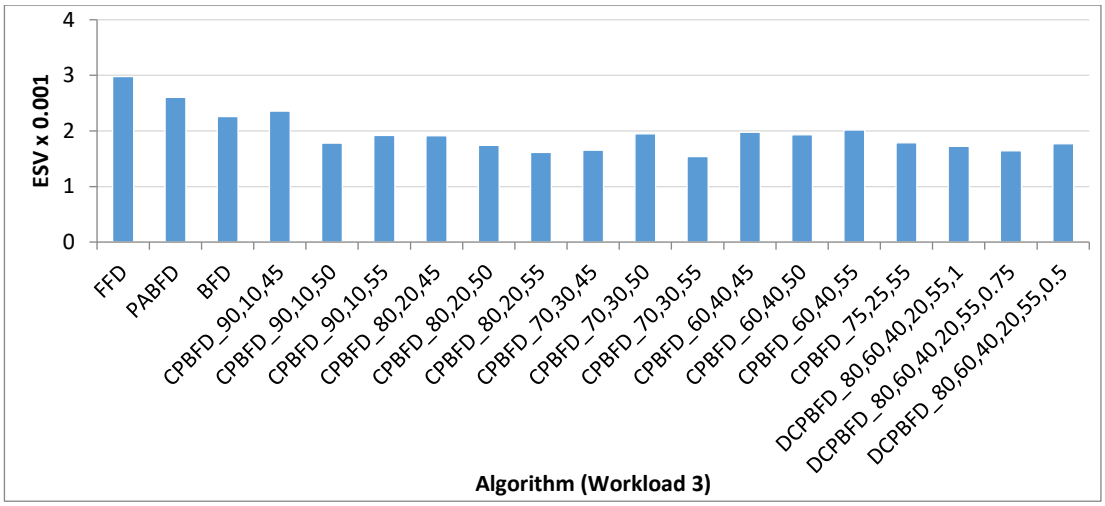

Figure G.13: ESV metric of VM placement algorithms for wokload 3

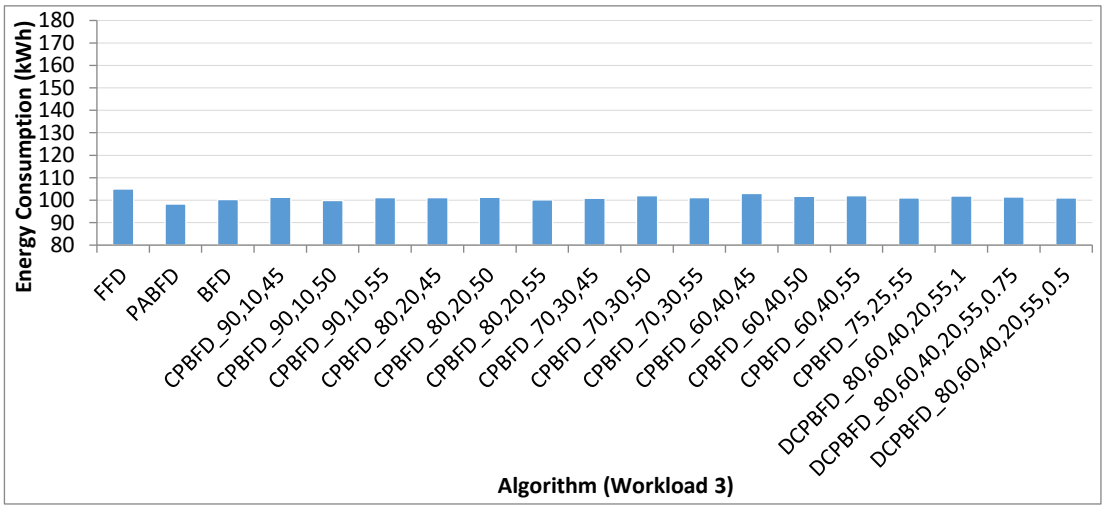

Figure G.14 The energy consumption of VM placement algorithms for wokload 3

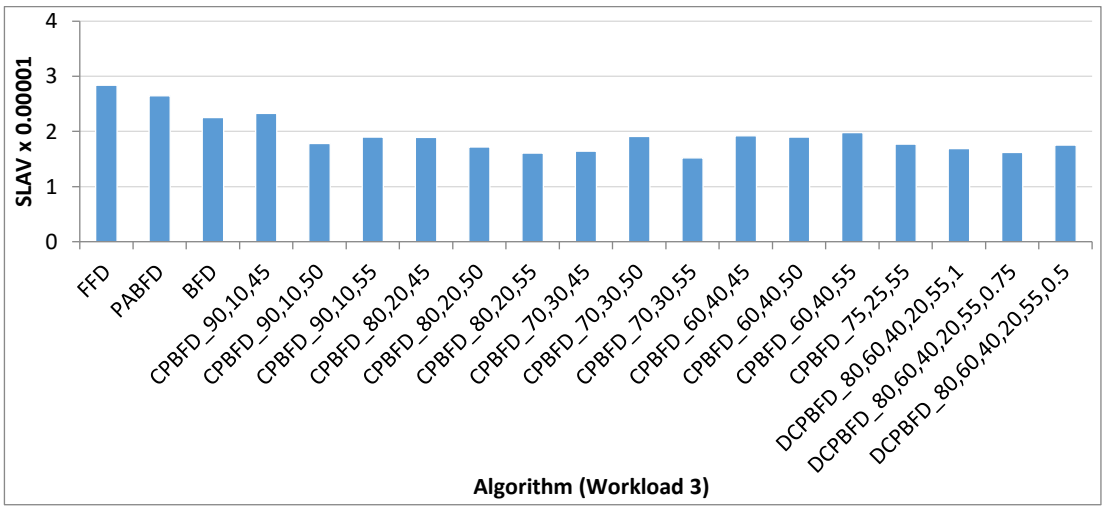

Figure G.15: SLAV metric of VM placement algorithms for wokload 3

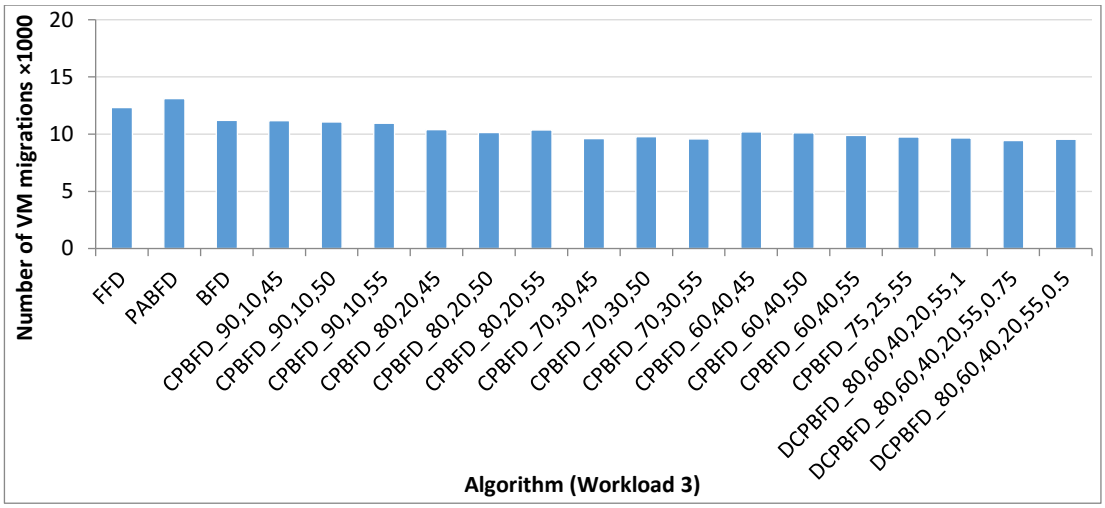

Figure G.16: Number of VM migrations of VM placement algorithms for wokload 3

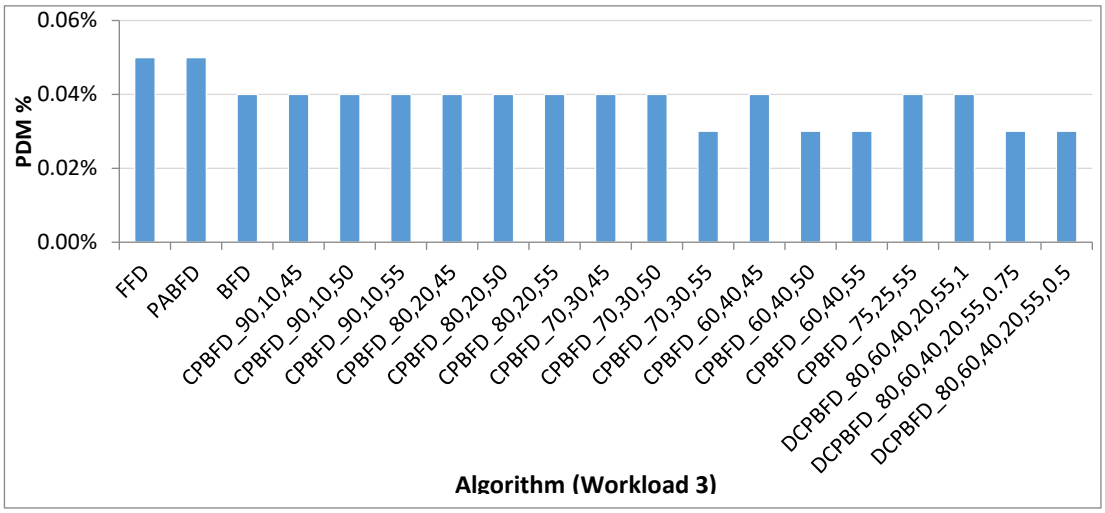

Figure G.17: PDM metric of VM placement algorithms for wokload 3

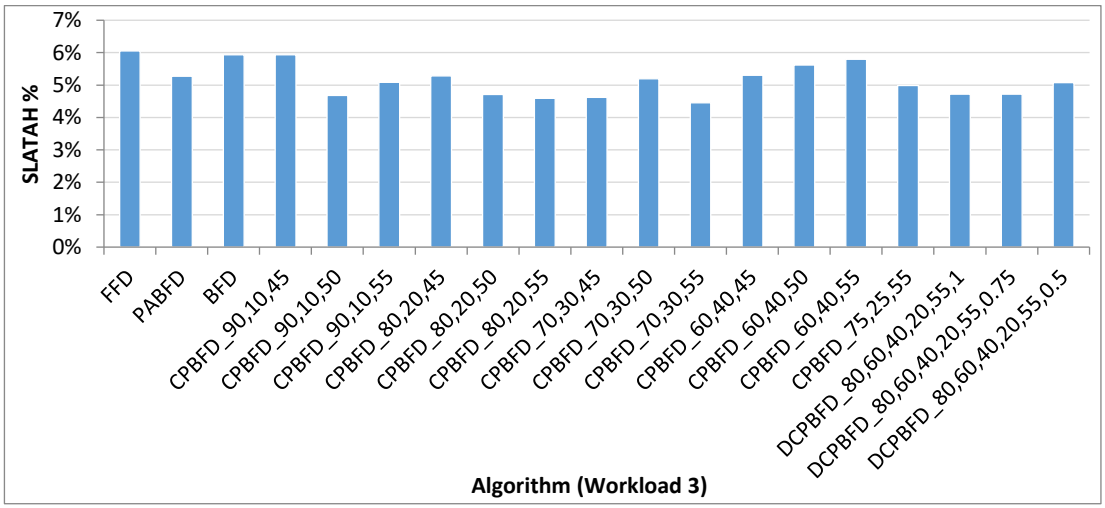

Figure G.18: SLATAH metric of VM placement algorithms for wokload 3

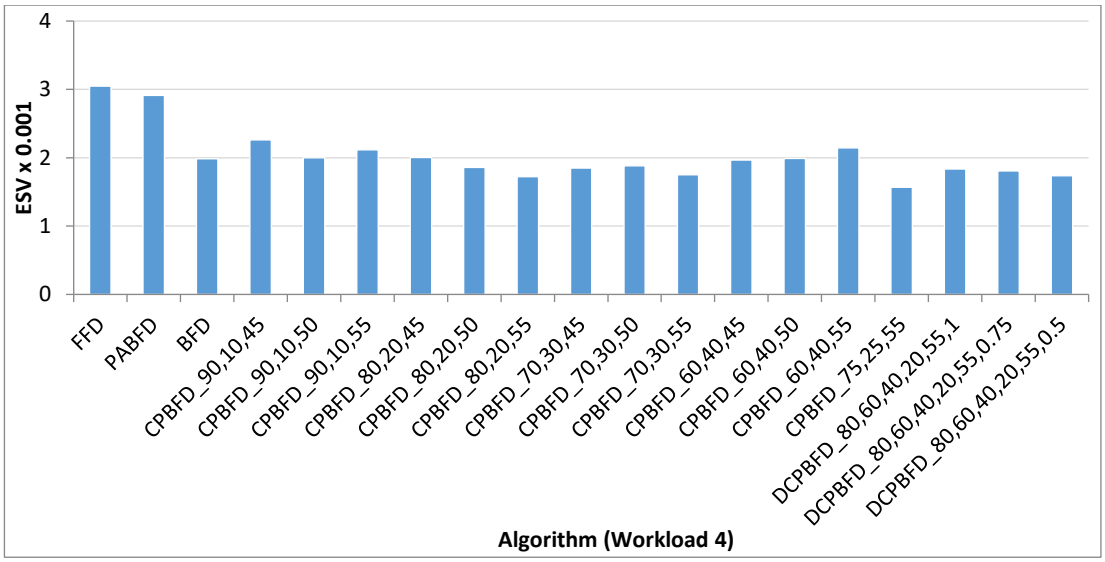

Figure G.19: ESV metric of VM placement algorithms for wokload 4

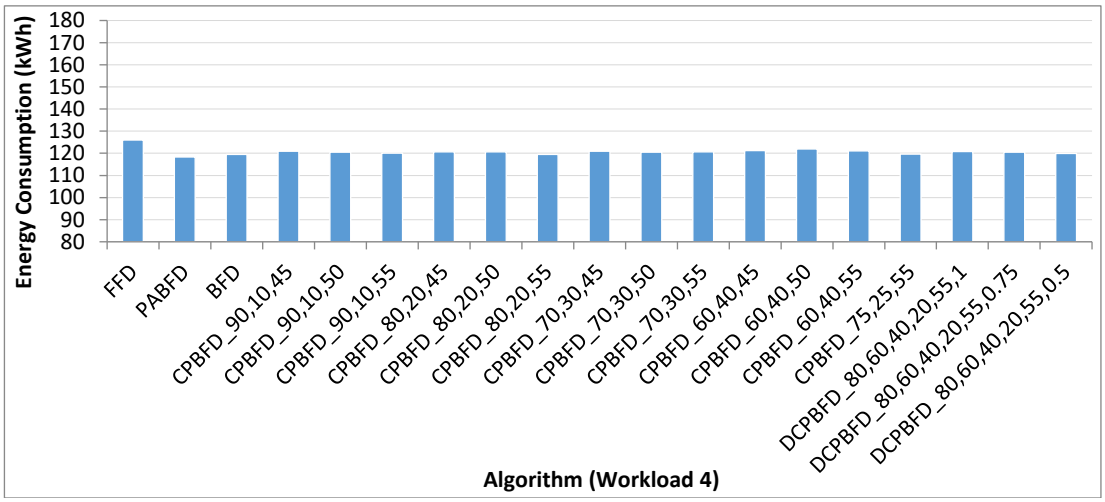

Figure G.20: The energy consumption of VM placement algorithms for wokload 4

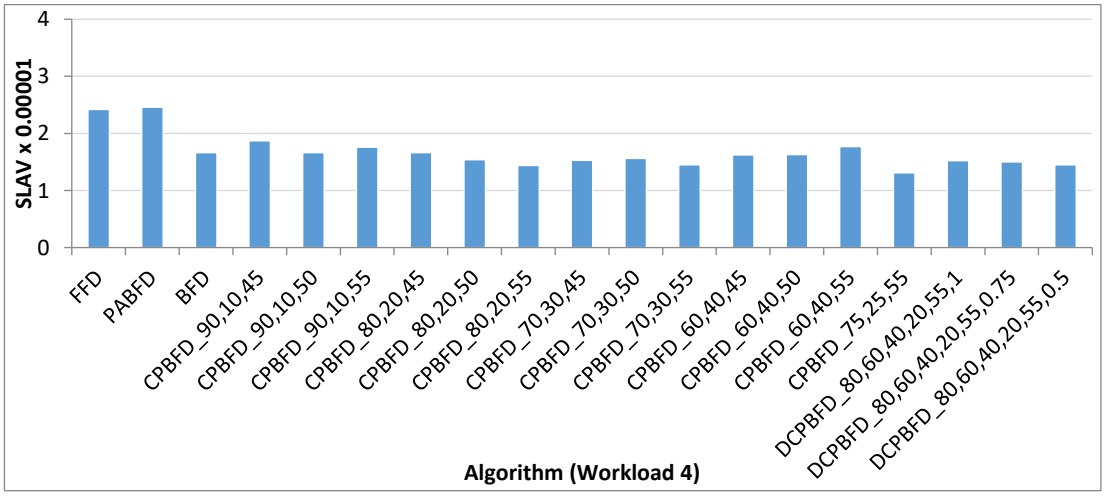

Figure G.21: SLAV metric of VM placement algorithms for wokload 4

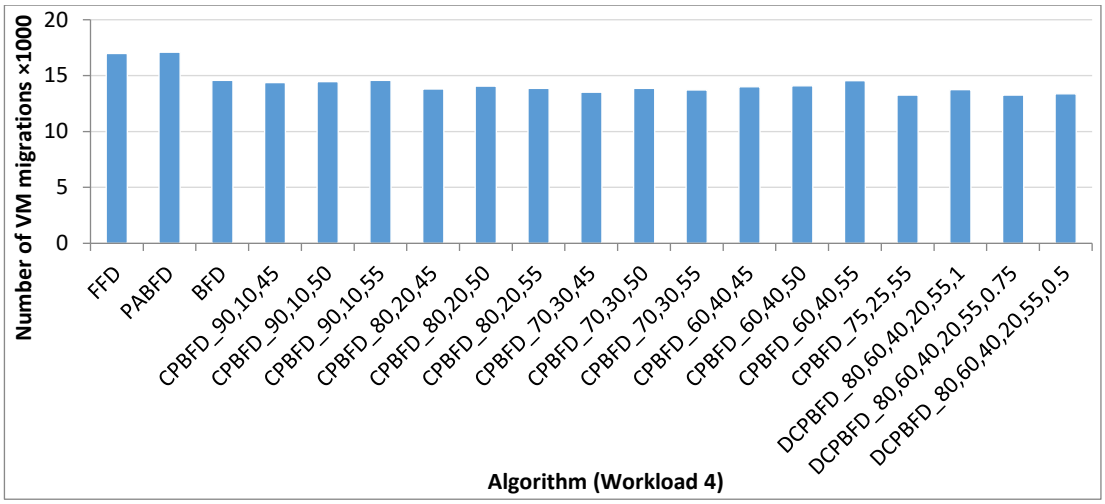

Figure G.22: Number of VM migrations of VM placement algorithms for wokload 4

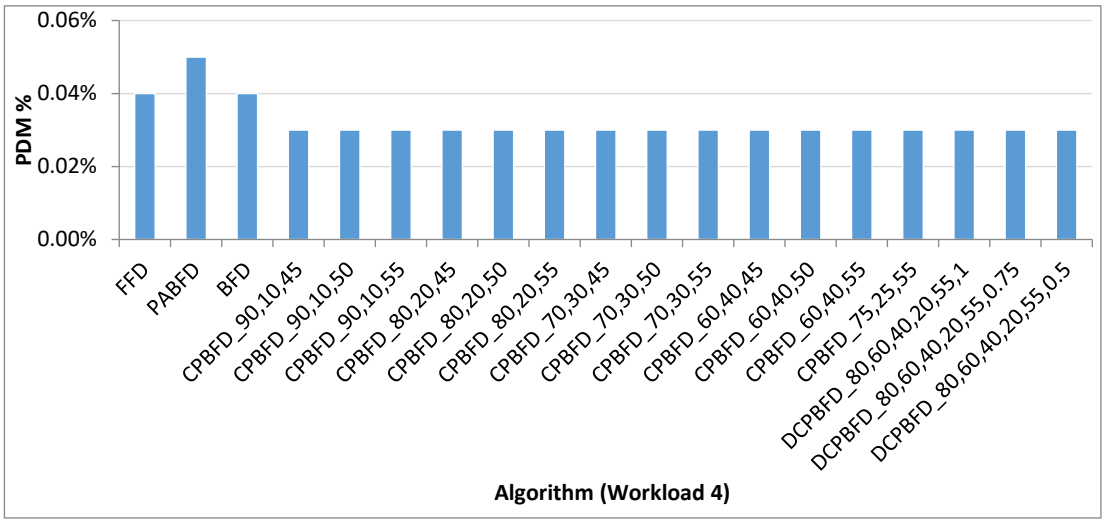

Figure G.23: PDM metric of VM placement algorithms for wokload 4

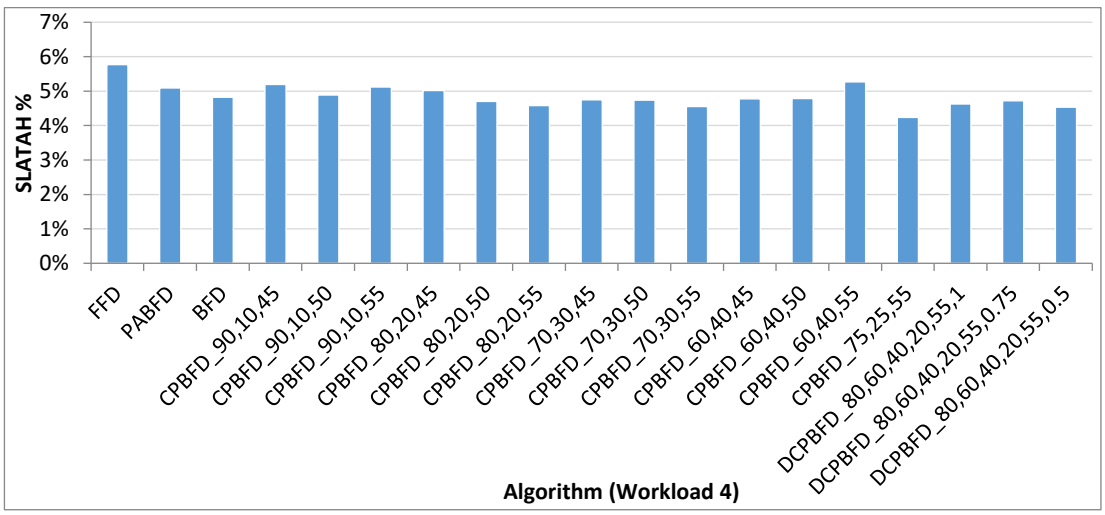

Figure G.24: SLATAH metric of VM placement algorithms for wokload 4

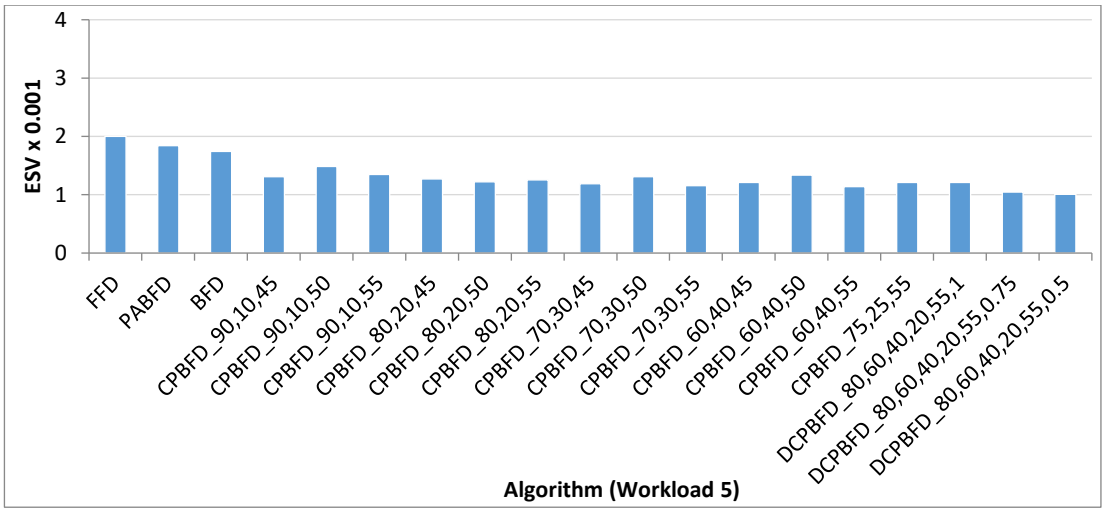

Figure G.25: ESV metric of VM placement algorithms for wokload 5

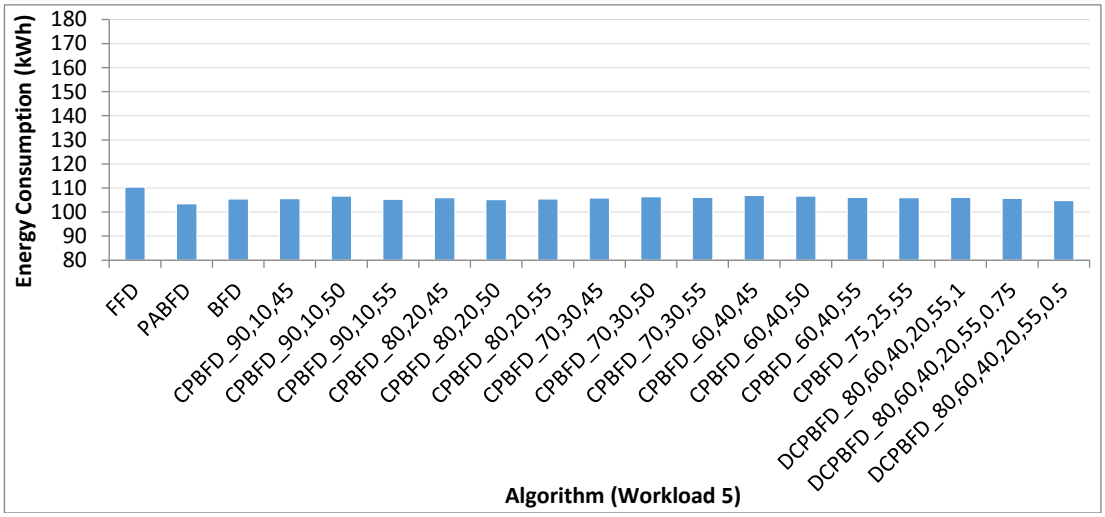

Figure G.26: The energy consumption of VM placement algorithms for wokload 5

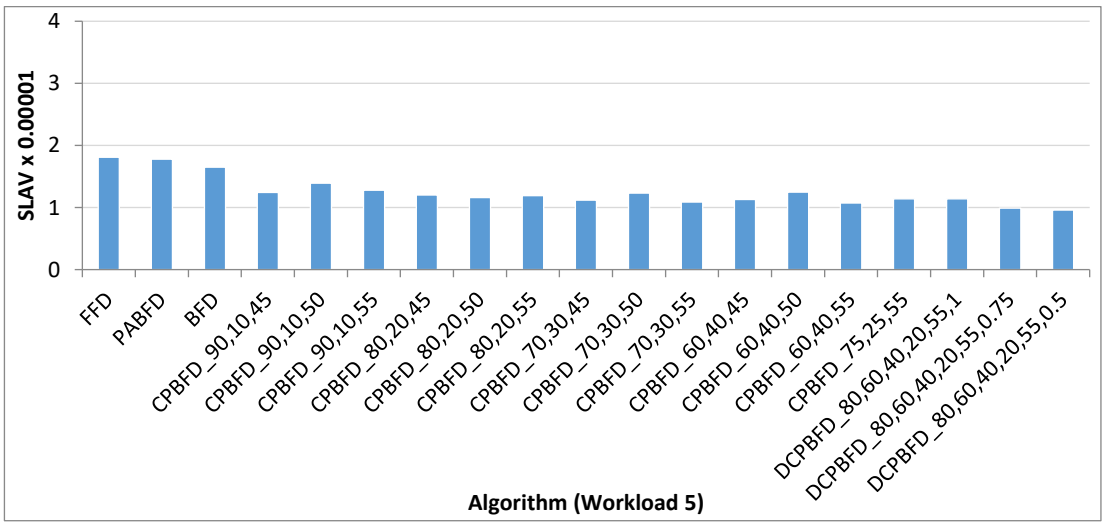

Figure G.27: SLAV metric of VM placement algorithms for wokload 5

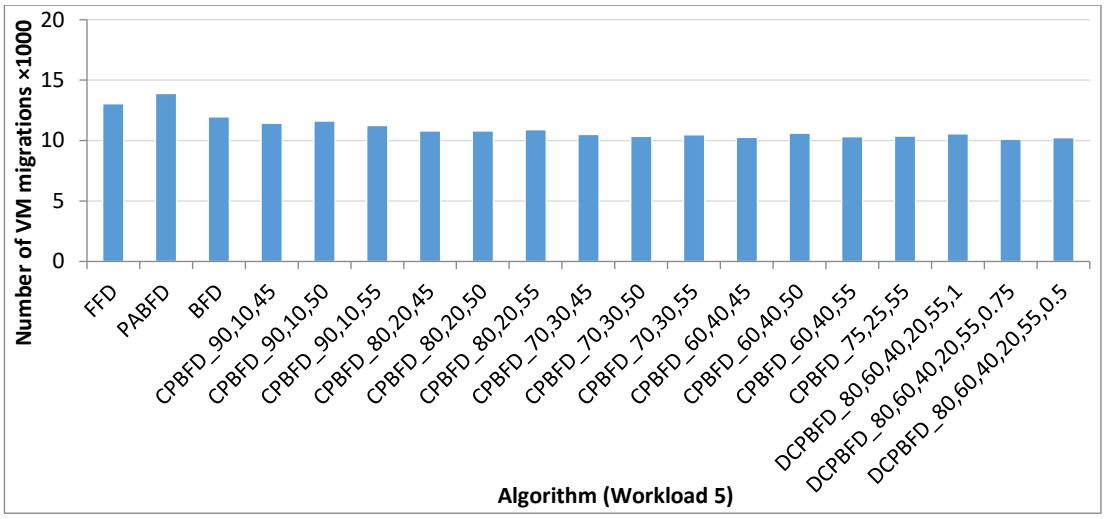

Figure G.28: Number of VM migrations of VM placement algorithms for wokload 5

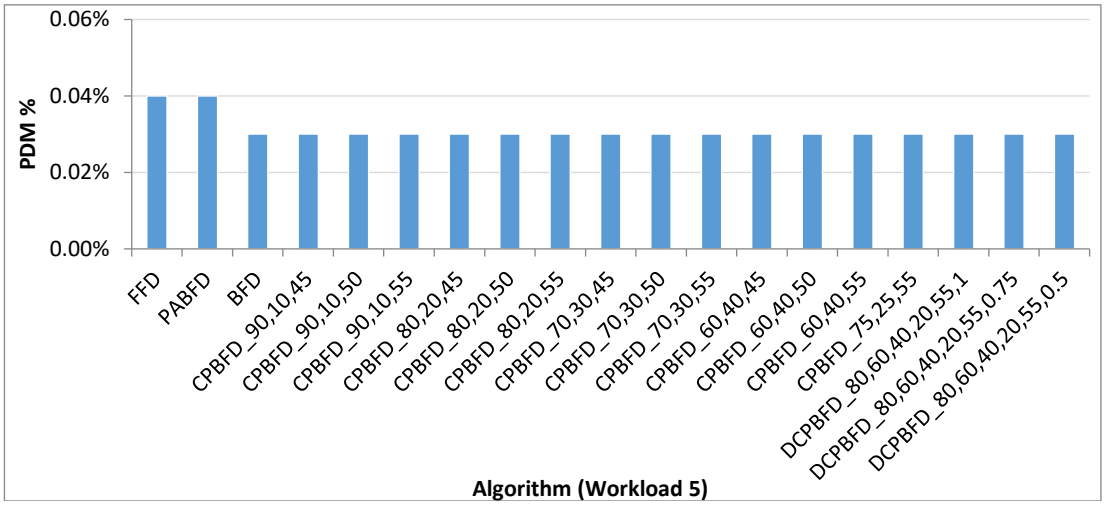

Figure G.29: PDM metric of VM placement algorithms for wokload 5

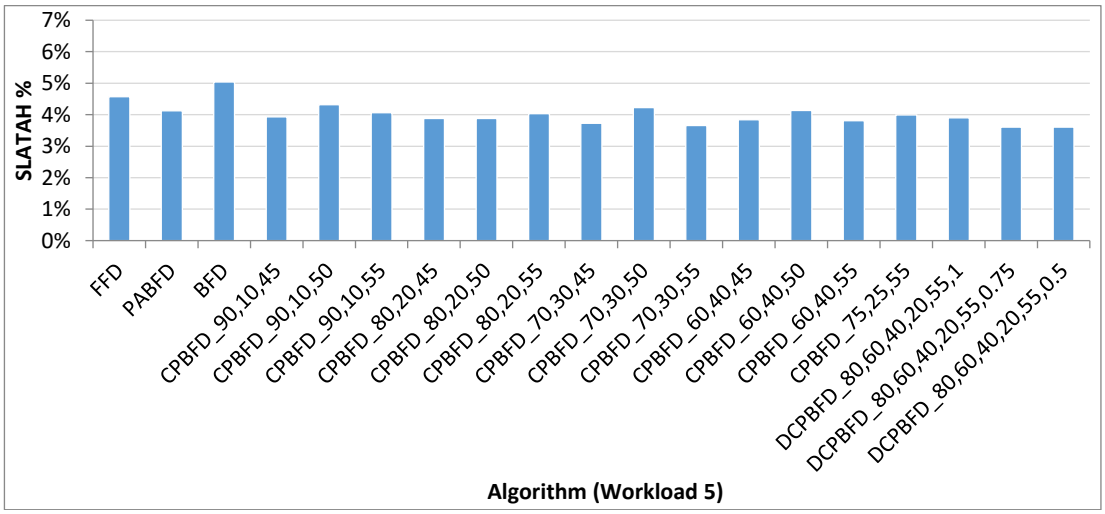

Figure G.30: SLATAH metric of VM placement algorithms for wokload 5

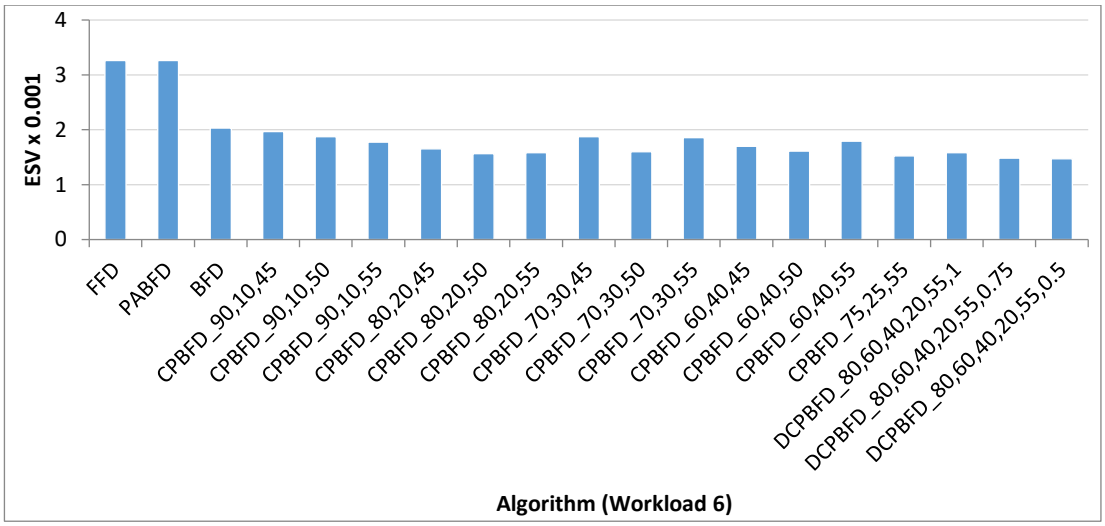

Figure G.31: ESV metric of VM placement algorithms for wokload 6

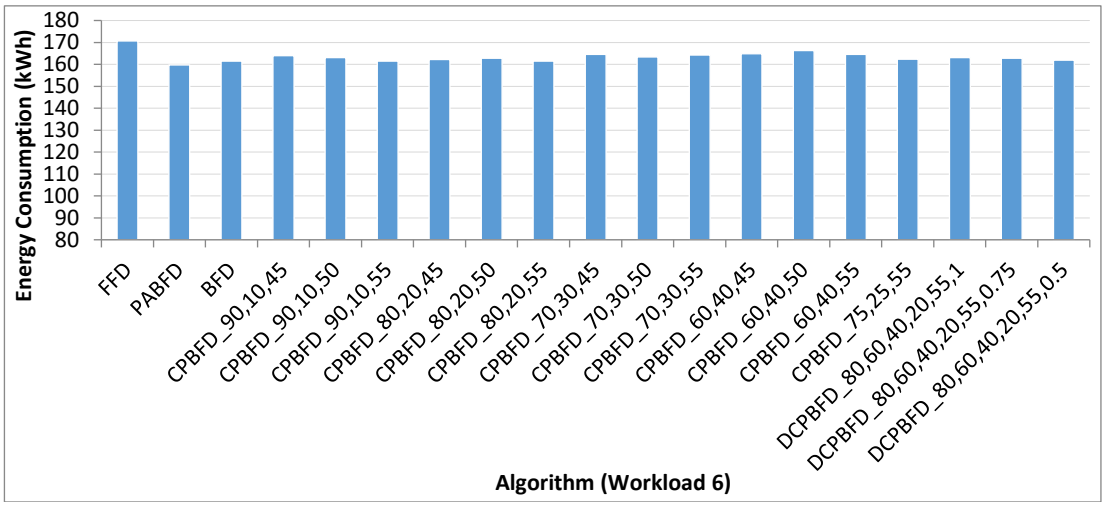

Figure G.32: The energy consumption of VM placement algorithms for wokload 6

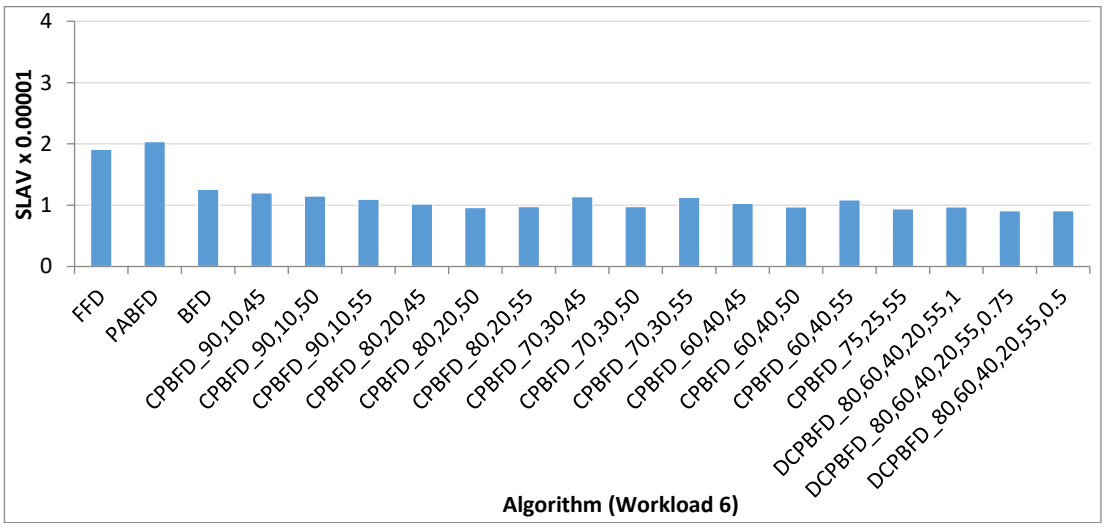

Figure G.33: SLAV metric of VM placement algorithms for wokload 6

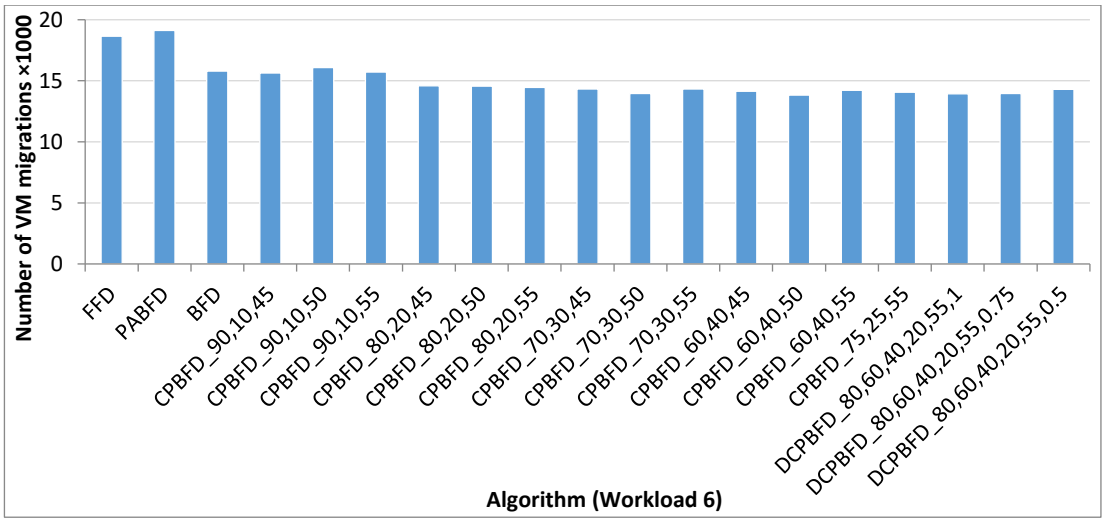

Figure G.34: Number of VM migrations of VM placement algorithms for wokload 6

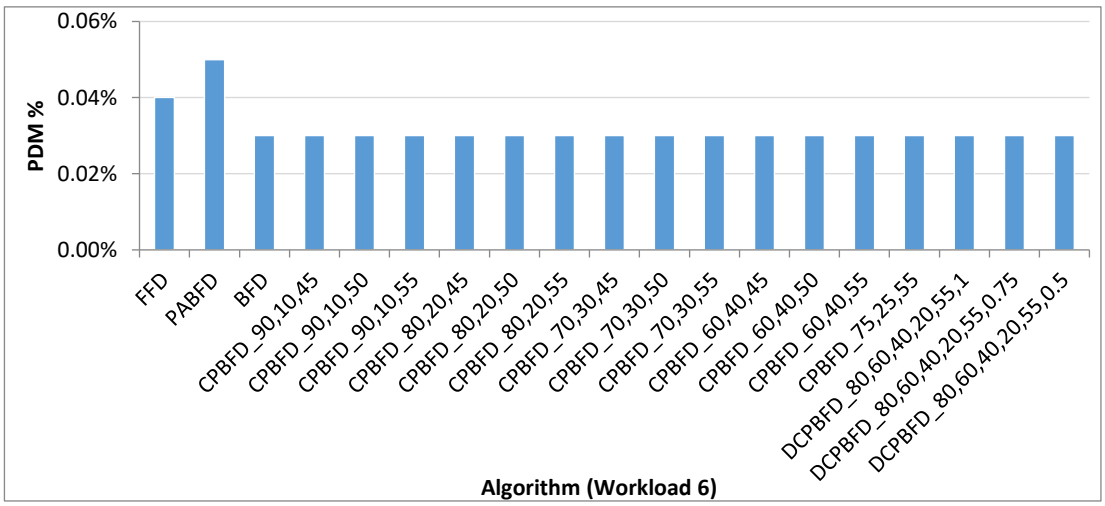

Figure G.35: PDM metric of VM placement algorithms for wokload 6

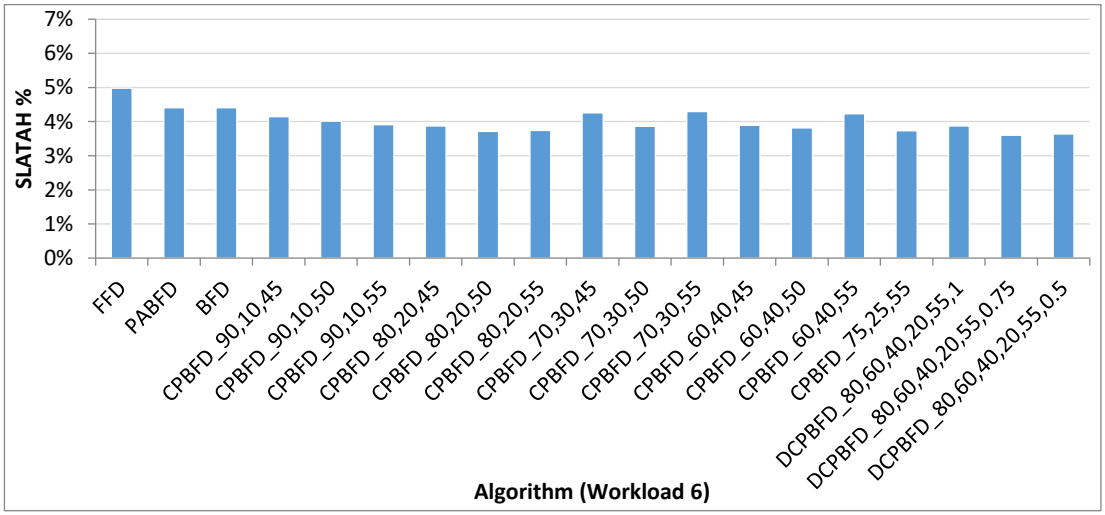

Figure G.36: SLATAH metric of VM placement algorithms for wokload 6

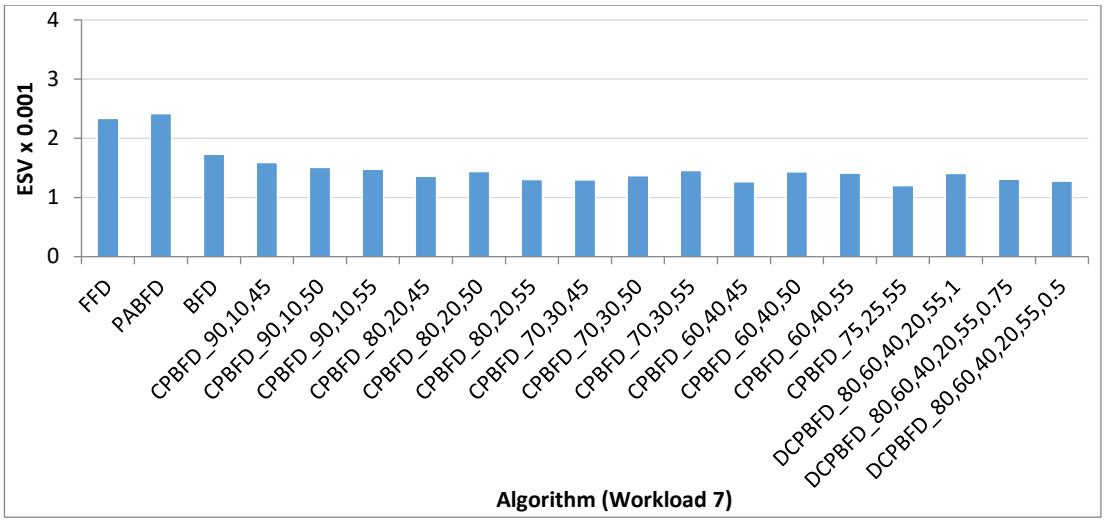

Figure G.37: ESV metric of VM placement algorithms for wokload 7

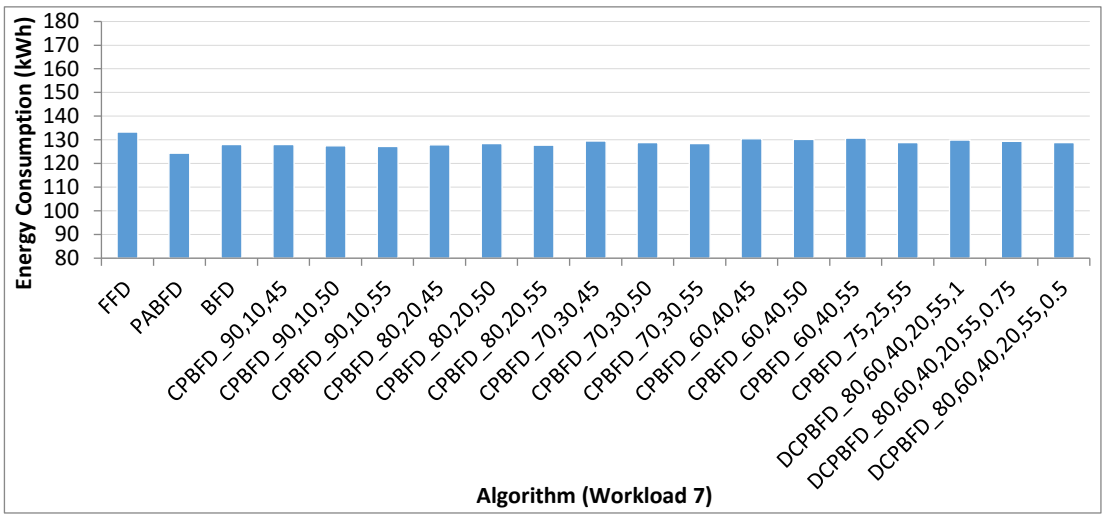

Figure G.38: The energy consumption of VM placement algorithms for wokload 7

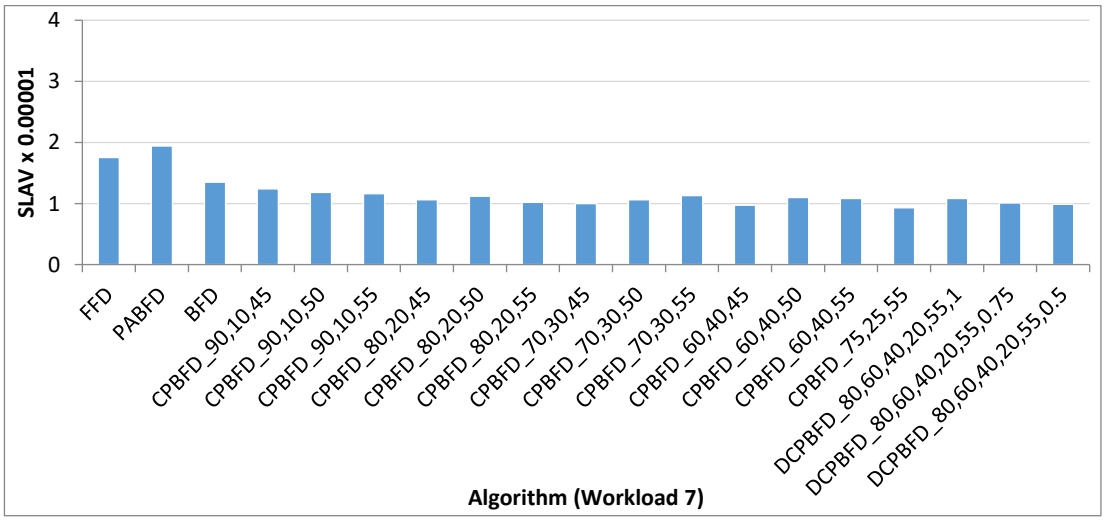

Figure G.39: SLAV metric of VM placement algorithms for wokload 7

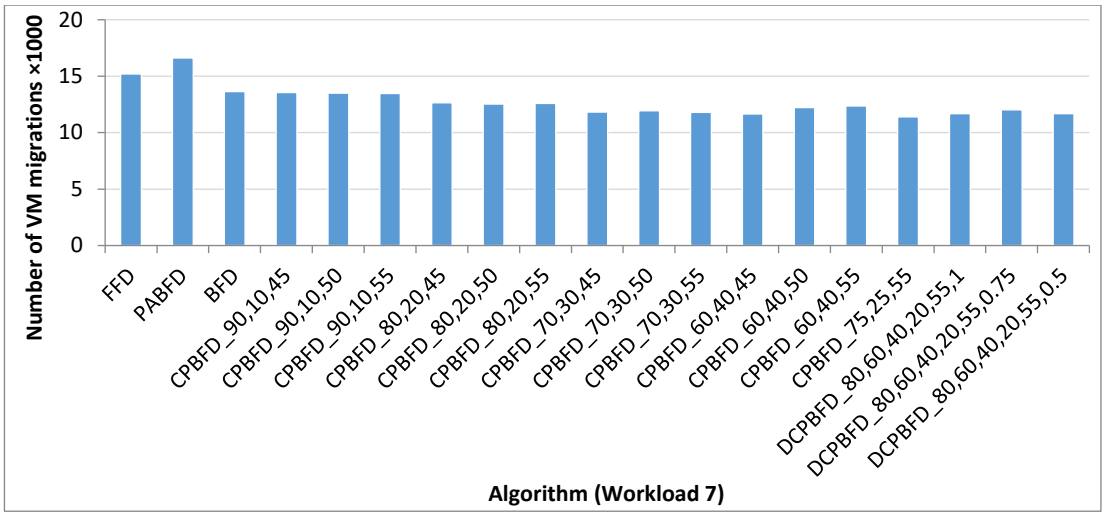

Figure G.40: Number of VM migrations of VM placement algorithms for wokload 7

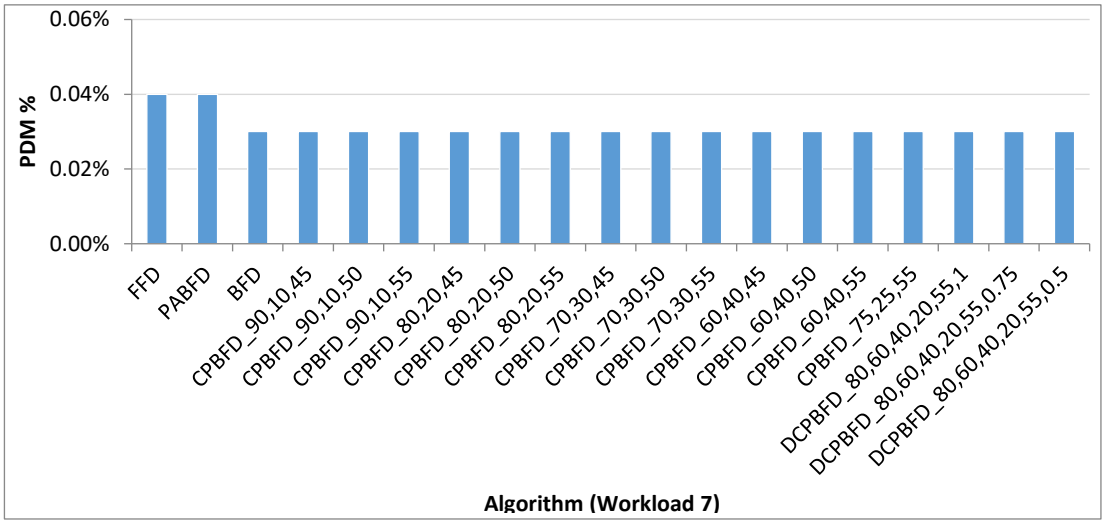

Figure G.41: PDM metric of VM placement algorithms for wokload 7

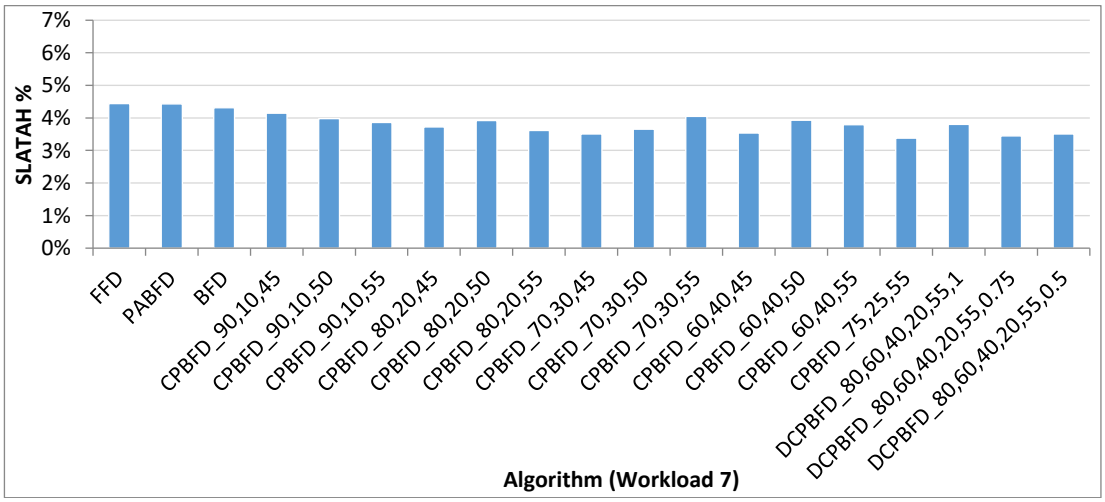

Figure G.42: SLATAH metric of VM placement algorithms for wokload 7

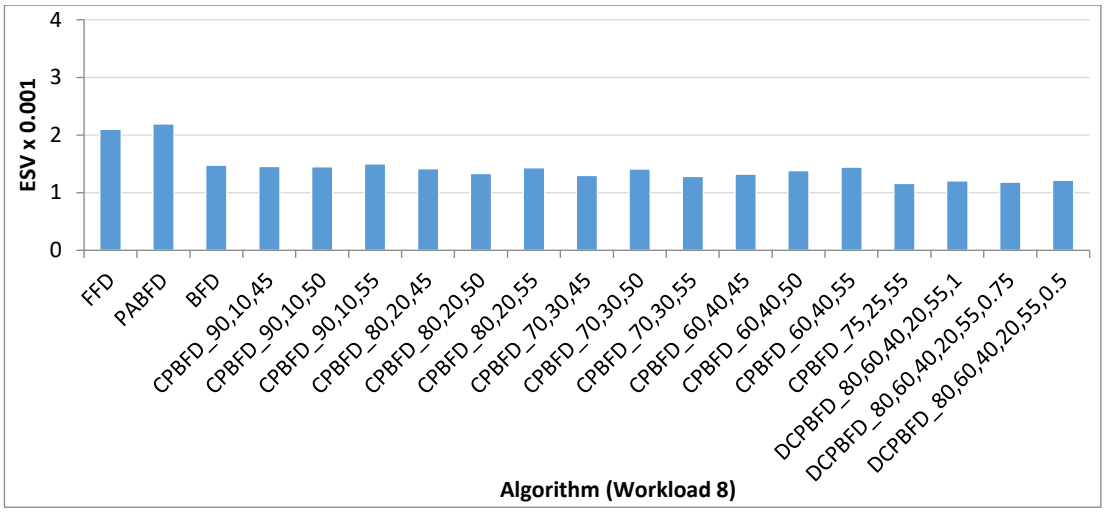

Figure G.43: ESV metric of VM placement algorithms for wokload 8

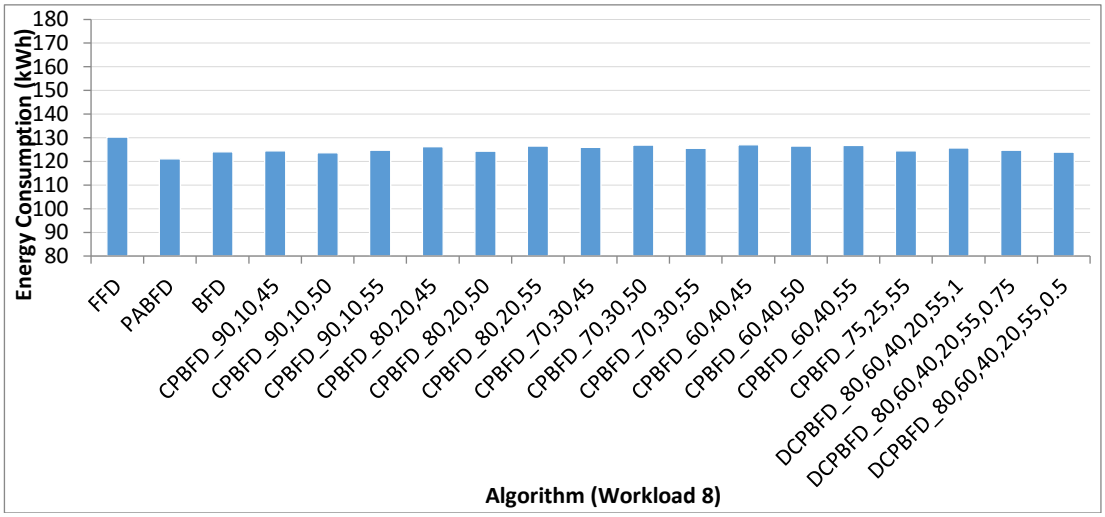

Figure G.44: The energy consumption of VM placement algorithms for wokload 8

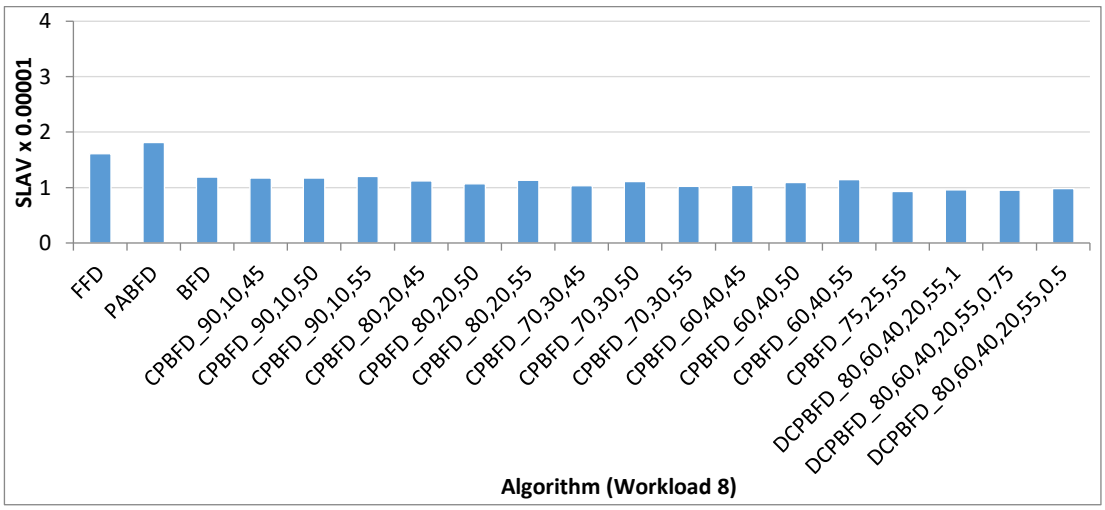

Figure G.45: SLAV metric of VM placement algorithms for wokload 8

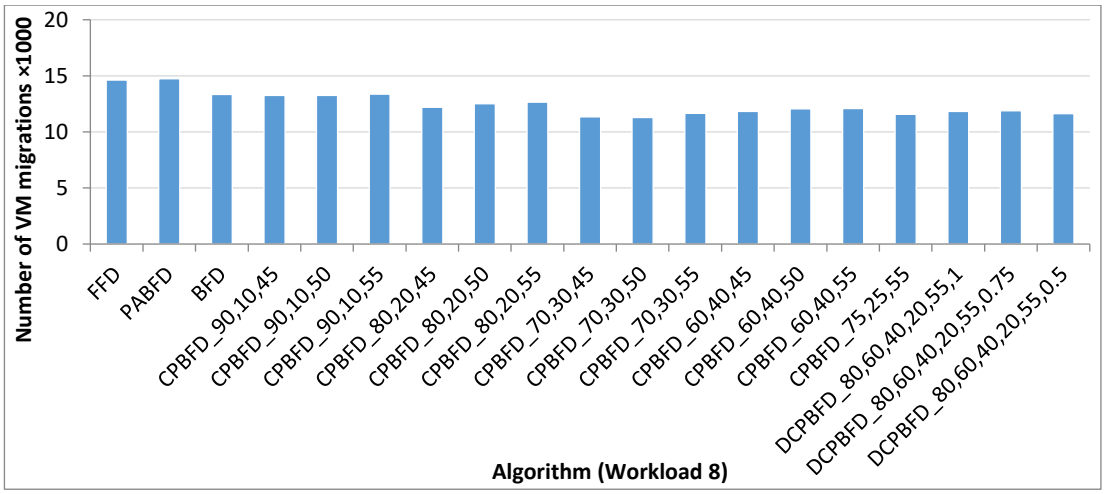

Figure G.46: Number of VM migrations of VM placement algorithms for wokload 8

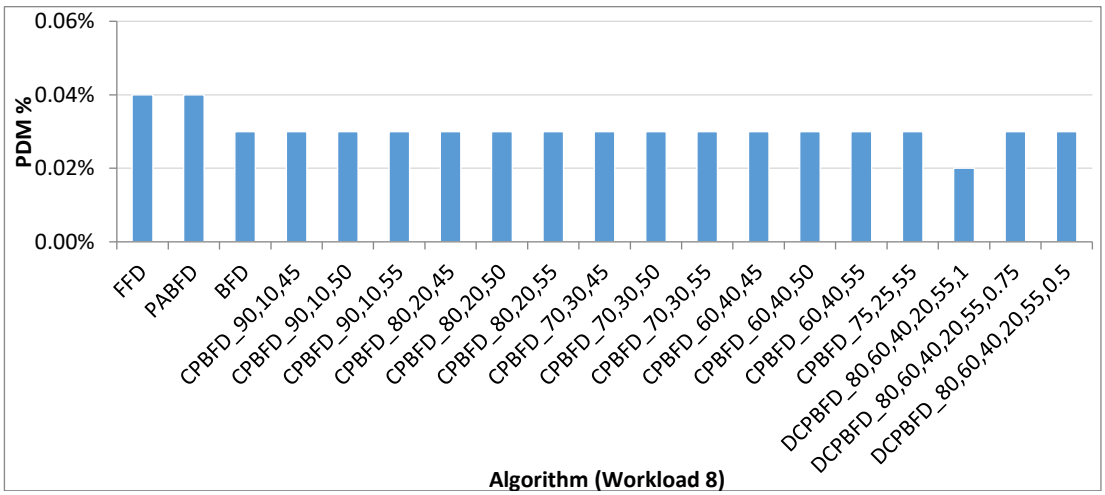

Figure G.47: PDM metric of VM placement algorithms for wokload 8

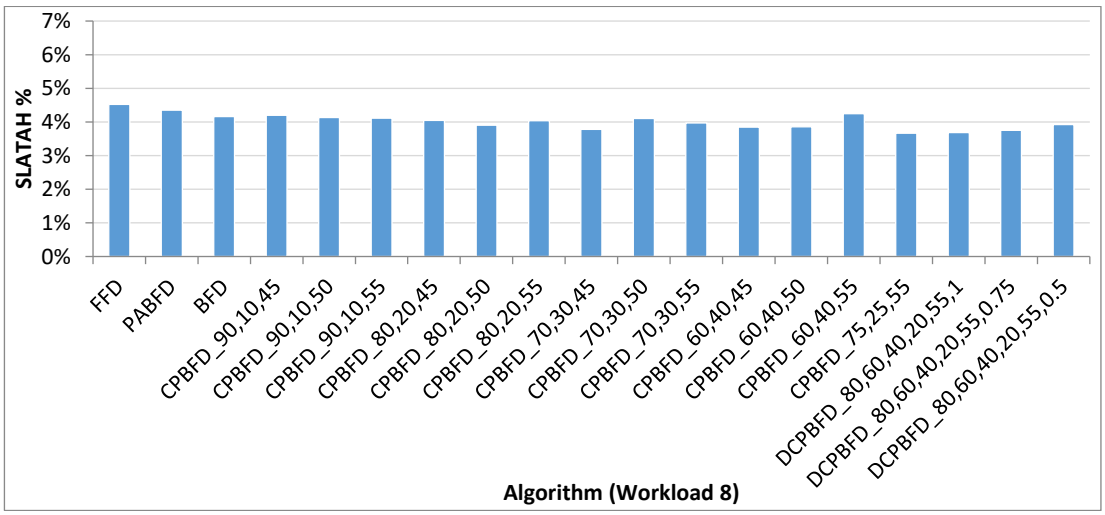

Figure G.48: SLATAH metric of VM placement algorithms for wokload 8

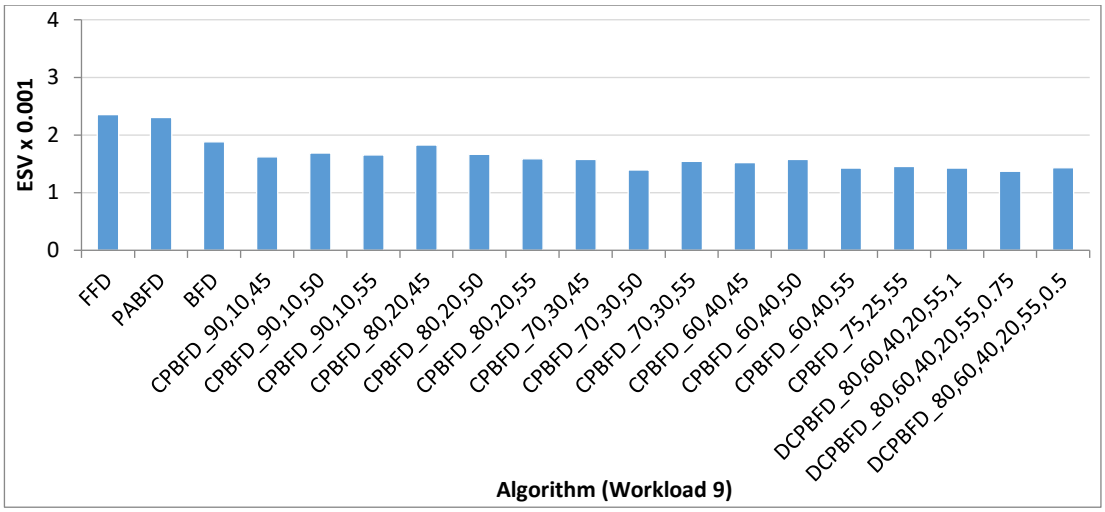

Figure G.49: ESV metric of VM placement algorithms for wokload 9

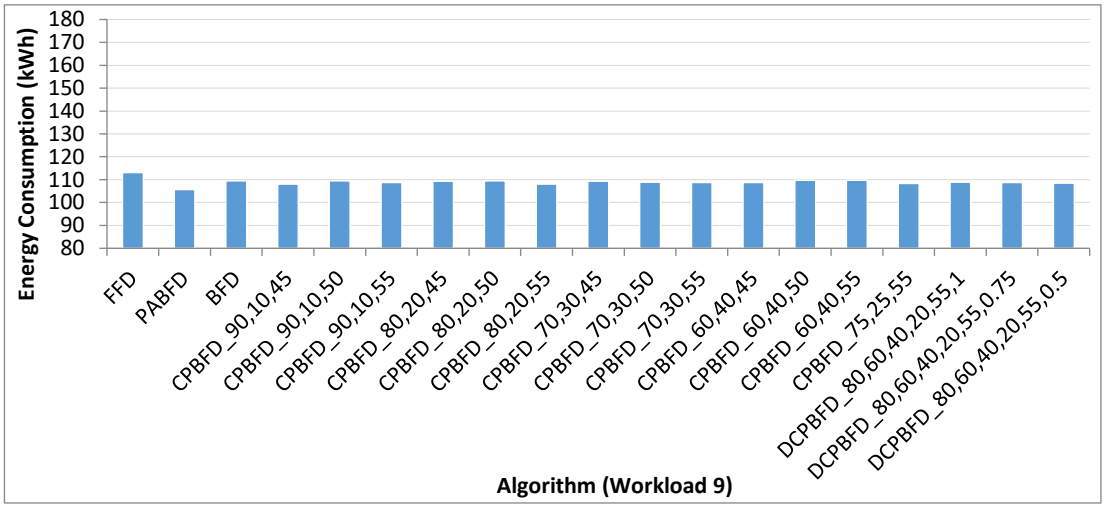

Figure G.50: The energy consumption of VM placement algorithms for wokload 9

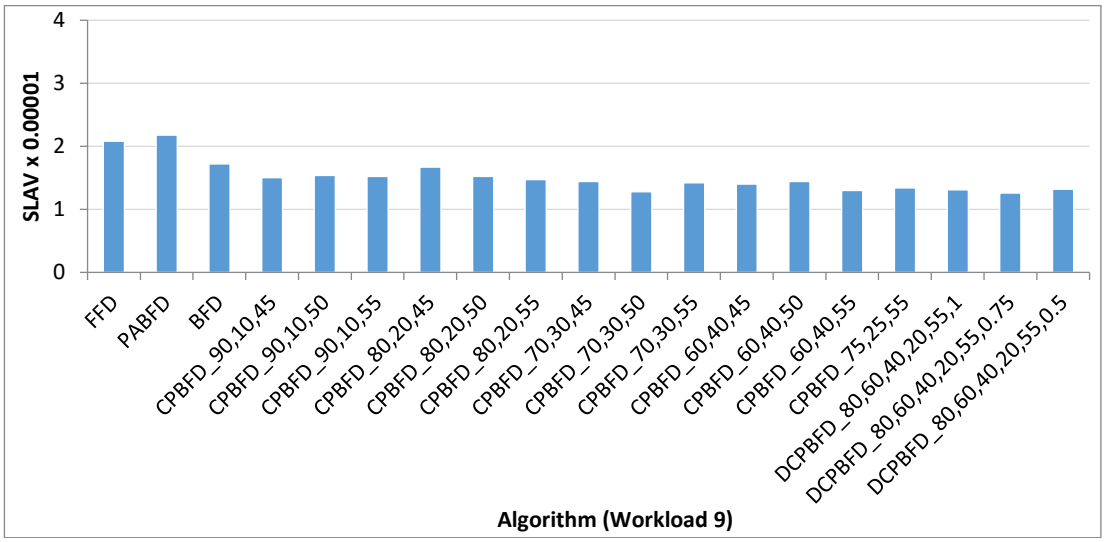

Figure G.51: SLAV metric of VM placement algorithms for wokload 9

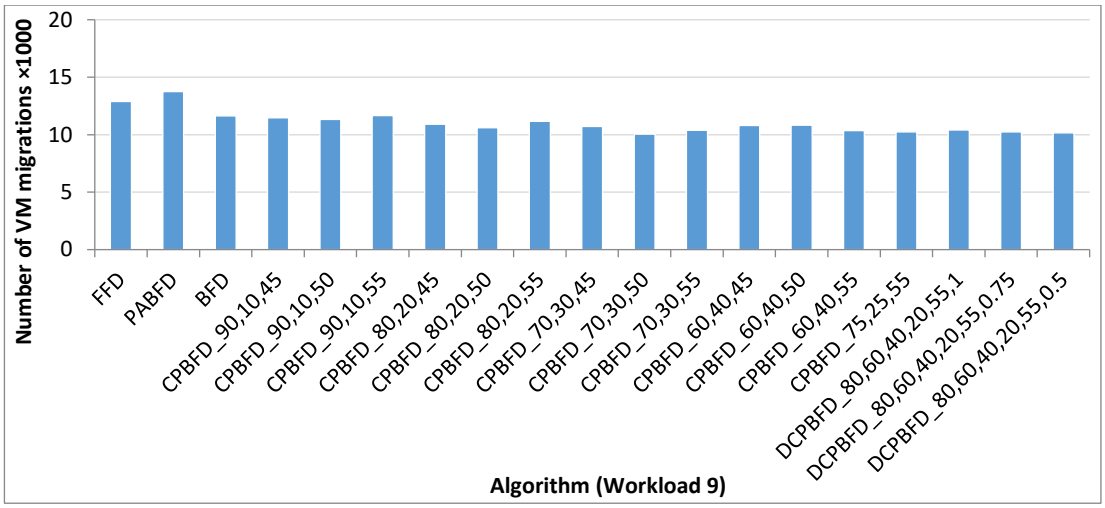

Figure G.52: Number of VM migrations of VM placement algorithms for wokload 9

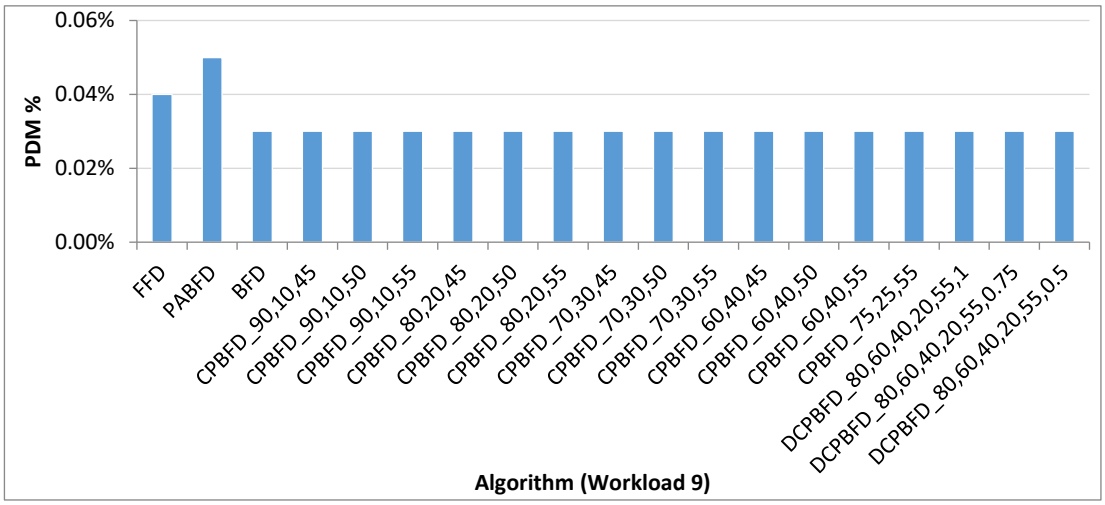

Figure G.53: PDM metric of VM placement algorithms for wokload 9

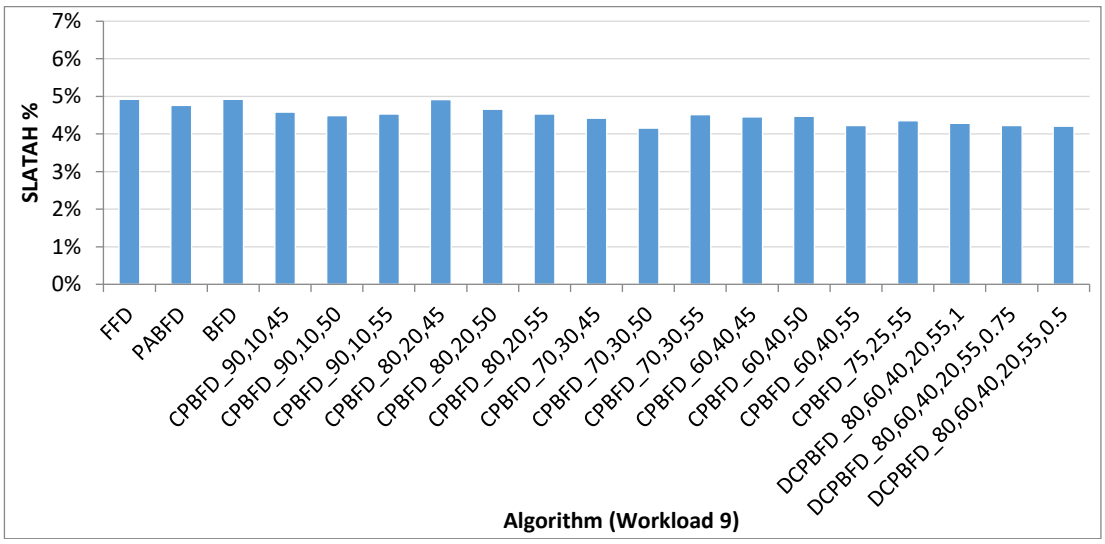

Figure G.54: SLATAH metric of VM placement algorithms for wokload 9

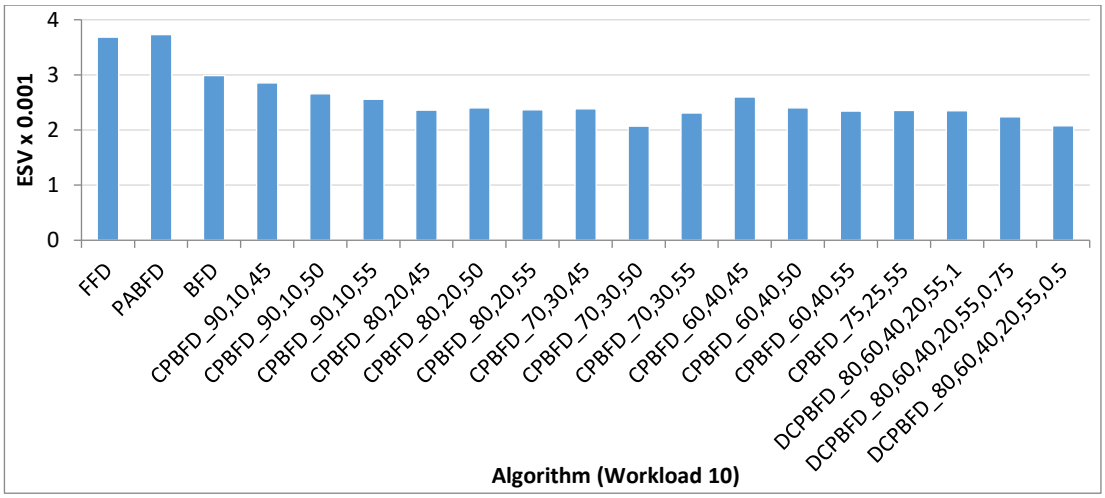

Figure G.55: ESV metric of VM placement algorithms for wokload 10

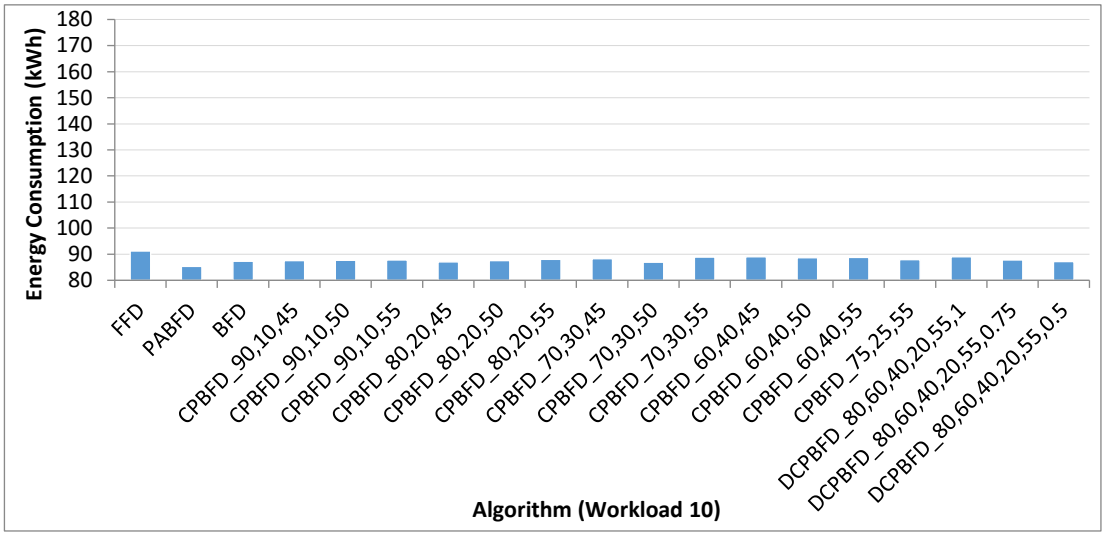

Figure G.56: The energy consumption of VM placement algorithms for wokload 10

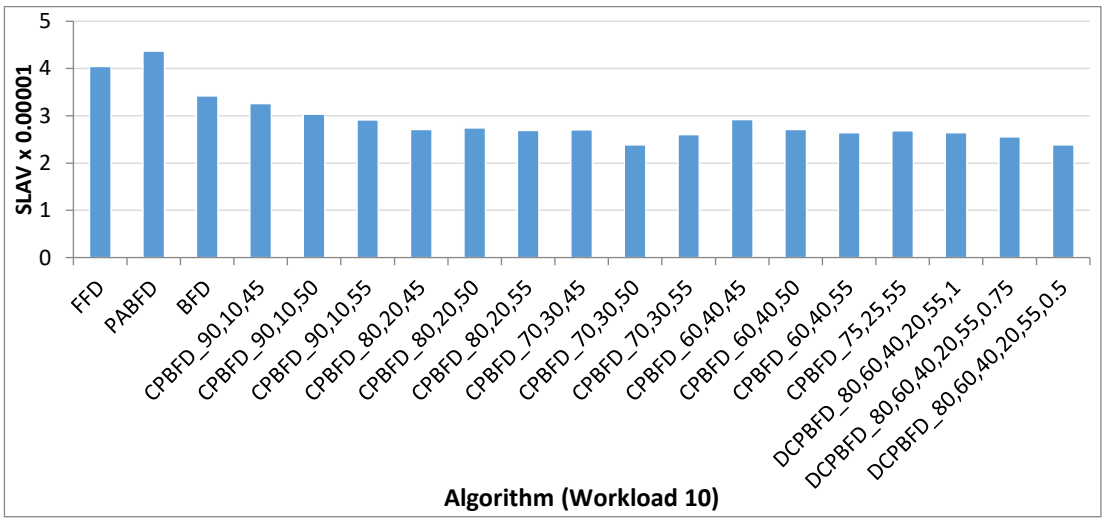

Figure G.57: SLAV metric of VM placement algorithms for wokload 10

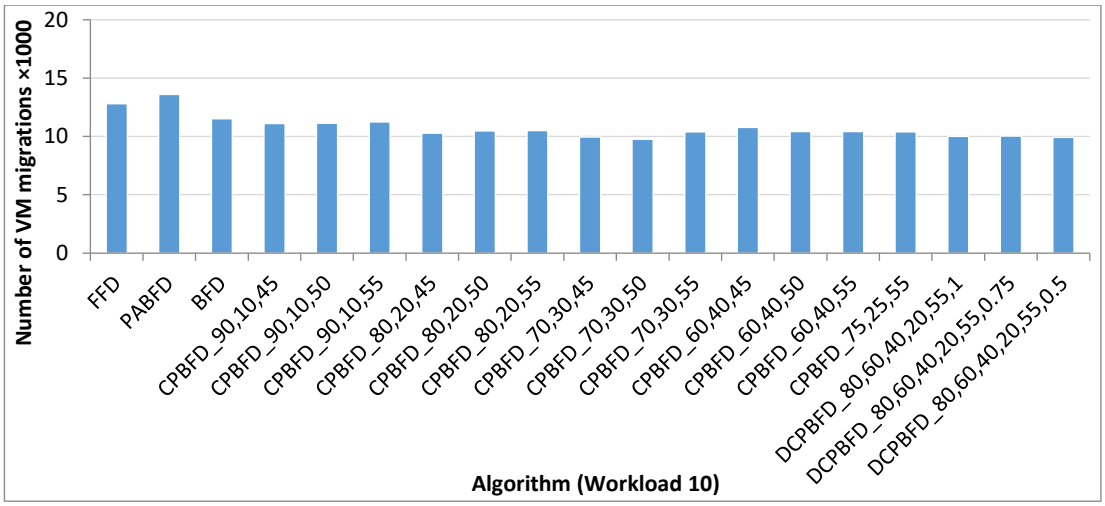

Figure G.58: Number of VM migrations of VM placement algorithms for wokload 10

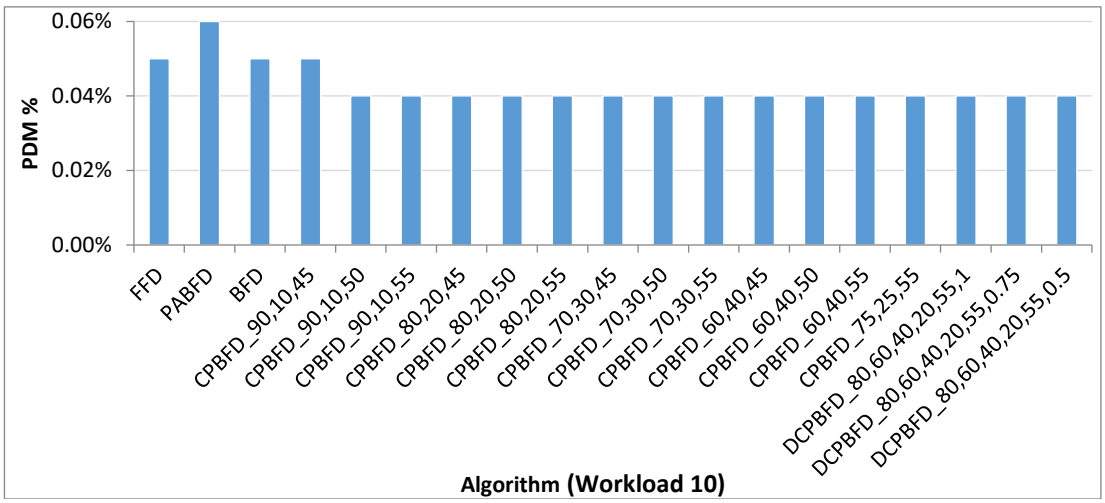

Figure G.59: PDM metric of VM placement algorithms for wokload 10

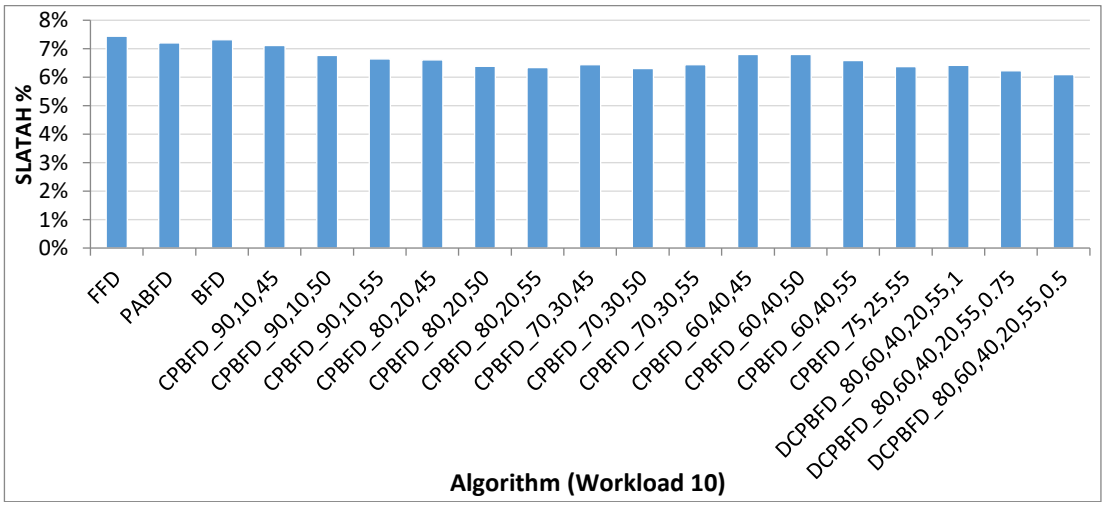

Figure G.60: SLATAH metric of VM placement algorithms for wokload 10

We calculated the mean and standerd deviation for results of 10 workloads of each algoiritm with specific parameter. Then, the 95% confidence interval of the results is calculated for each algorithm using mean and standerd deviation as shown in Table G.1 and F.2 and Figures 3.4 to 3.15.

Appendix H: Installation and Running the Clojure

Download simulation from https://github.com/beloglazov/tpds-2013-simulation.

First, run the script to install Michael Thomas Flanagan's Java Scientific Library (http://www.ee.ucl.ac.uk/~mflanaga/java/) to the local Maven repository:

/. <install_flanagan

To download the dependencies and compile the source code:

<lein compile

Appendix I: Code of Calculation CPU Utilization of PM

```
(defn host-utilization-history [host vms]
  {:pre [(map? host)
         (coll? vms)]
   :post [(coll? %)]}
  (apply map (fn [& xs] (/ (reduce + xs) (host :mips)))
    (map (fn [vm]
           (map #(* % (vm :mips))
                (vm :utilization)))
         vms)))
```

Appendix J: Code of POTFT Algorithm

```
//1: PM’s status = normal loaded
(defn otf [param time-step migration-time host vms]
  {:pre [(not-negnum? param)
         (not-negnum? time-step)
         (not-negnum? migration-time)
         (map? host)
         (coll? vms)]
   :post [(boolean? %)]}
  (let [utilization-history (host-utilization-history host vms)
        cnt (count utilization-history)
        overloading-steps (count (filter #(>= % 1) utilization-history))]
    (and
      (>= cnt 30)
      (> (/ overloading-steps cnt) param))))//2: if pm.OTF>(MAOTF×p) 3:PM’s status
= overloaded)4:end if 5: return PM’s status
```

Appendix K: Code of POTFT Algorithm with VM Quiescing

```
 (defn otf-q [param time-step migration-time host vms turnedoff-vms param-time]
//1: a decision on whether to migrate a VM =false
  {:pre [(not-negnum? param)
         (not-negnum? time-step)
         (not-negnum? migration-time)
         (not-negnum? Param-time)
         (map? host)
         (coll? vms)
         (coll? turnedoff-vms)]
   :post [(boolean? %)]}
  (loop [temp-vm []]
  (let [utilization-history (host-utilization-history host vms)
        cnt (count utilization-history)
        overloading-steps (count (filter #(>= % 1) utilization-history))]
if (and
      (>= cnt 30)
      (> (/ overloading-steps cnt) param)) // 2:if pm.OTF>(MAOTF×p)then
    (do
      (if (> time-step param-time) //3:if current simulation time>T then
        (do
          (> time-step param-time)) //4: a decision on whether to migrate a VM =true
        else do( //5: else
            (conj temp-vm (rand-nth vms)) //6: Select VM randomly  from a set of
generated VMs
            (disj vms (temp-vm)) //7:Remove a CPU utilization trace of a selected VM
            (conj turnedoff-vms (temp-vm))))) 8: Add a VM to the set of turned off VMs
9:end if
else do ((//10: else
        (if (and
             (< (/ overloading-steps cnt) (valueof(MAOTF×(p-0.1))
             (not= turnedoff-vms nil)) //11: if the set of turned off VMs≠ Null and
pm.OTF < (MAOTF×(p-0.1)) // valueof(MAOTF×(p-0.1) is entered manually
     (do
       ((conj temp-vm (rand-nth turnedoff-vms))// 12: Select a VM randomly from
turned off VMs
       (conj vms (temp-vm)) 13:  Assign a CPU utilization trace for the selected VM
       (disj turnedoff-vms (temp-vm))))))) 14: Remove a VM from the set of turned
off VMs 15: end if 16: end if 17: return (a set of turned off VMs, a decision on
whether to migrate a VM)
```

Appendix L: Some CPU Utilization Traces of Input File for Clojure

Table L.1: Some random CPU utilization traces of input file for Clojure

Simulation ID	VM ID	Time Frame	CPU Frequency	Utilization
0	0	0	2400	0.04
1	6	0	3000	0.19
6	17	250	1700	0.13
40	3	41	2000	0.29
80	0	0	2000	0.43

The workload data has been obtained from the CoMon project, a monitoring infrastructure for PlanetLab (http://comon.cs.princeton.edu/). The data used in the simulations in CSV format are available at https://github.com/beloglazov/tpds-2013-workload. The workload data has been transformed in the required internal data format and is located in the workload/ directory.

Appendix M: Code of Workload Generator

```
(ns simulation.runners.workload-generator
  (:use simulation.io)
  (:gen-class))
; Amazon EC2 instance types: 1700, 2000, 2400, 3000
; Amazon EC2 server types: 2x2400, 4x3000, 8x3000, 16x3000
(def vm-mips [1700, 2000, 2400, 3000])
(def host {:mips 12000}) ;4x3000
(def thresholds [0.8 0.85 0.9 0.95 1.0])
(defn -main [& args]
  (let [otf-min (Double/valueOf (nth args 0))
        otf-max (Double/valueOf (nth args 1))
        otf-max-at-30 (Double/valueOf (nth args 2))
        n (Double/valueOf (nth args 3))
        input-path (nth args 4)
        output-path (nth args 5)
        ]
    (do
      (pregenerate-workload-filter-otf vm-mips host thresholds otf-min otf-max otf-
max-at-30 n input-path output-path)
      (println "Done"))))
; lein run -m simulation.runners.workload-generator 0.2 1.0 0.1 100 "../data"
"workload/planetlab_20_100_10_100"
simulation.io:
(defn pregenerate-workload-filter-otf [mips host thresholds otf-min otf-max otf-max-
at-30 n input-path output-path]
  {:pre [(coll? mips)
         (map? host)
         (coll? thresholds)
         (posnum? otf-min)
         (posnum? otf-max)
         (posnum? otf-max-at-30)
         (posnum? n)
         (string? input-path)
         (string? output-path)]
   :post [(coll? %)]}
  (loop [workload []]
    (if (== n (count workload))
      (do
        (spit output-path (json/json-str workload))
        workload)
      (do
        (prn (count workload))
        (let [vms (load-vms mips host (rand-nth thresholds) input-path)
            host-utilization (host-utilization-history host vms)
            host-utilization-30 (take 30 host-utilization)
            otf (double (/ (count (filter #(>= % 1) host-utilization)) (count host-
utilization)))
```

```
          otf-at-30 (double (/ (count (filter #(>= % 1) host-utilization-30)) (count host-
utilization-30)))]
        (if (and
              (>= otf otf-min)
              (<= otf otf-max)
              (<= otf-at-30 otf-max-at-30))
          (do
            (prn "otf" otf)
            (prn "otf-at-30" otf-at-30)
            (recur (conj workload vms)))
          (recur workload)))))))
```

Appendix N: Clojure Simulation Process Steps

The source code and the implementation of all benchmark algorithms created by [4] are available online in (https://github.com/beloglazov/tpds-2013-simulation/). The maximum OTF value after the first 30 time frames of each CPU utilization trace is modified in simulation from 09 to 10 to compare the algorithms with the same parameters in [4]. The authors in [4] constrained the maximum allowed OTF after the first 30 time steps to 10%, and as we mentioned in section 4.4.1, the maximum allowed OTF is limited after the first 30 time steps to 10%. So the file name format is modified in simulation from planetlab_20_100_09_100 to planetlab_20_100_10_100.

where:

20 - The minimum overall OTF value after the first 30 time frames
100 - The maximum overall OTF value
10 - The maximum OTF value after the first 30 time frames of each CPU utilization trace
100 - The number of different simulation runs

To create Eclipse project files:
<lein eclipse

To run the unit tests:
<lein midje

Examples to run benchmark algorithms (MHOD (markov-multisize), THR, MAD, IQR, LR, LRR, OTFT (otf-limit)):

Run following code in run-planetlab.sh that is available in (https://github.com/beloglazov/tpds-2013-simulation/ blob/master/run-planetlab.sh):

```
#!/bin/sh
rm results-planetlab-20-100-10-100.json
rm results-planetlab-20-100-10-100.csv
echo "[" > results-planetlab-20-100-10-100.json
lein run -m simulation.runners.universal workload/planetlab_20_100_10_100
markov-multisize 0.1 results-planetlab-20-100-10-100.json "[1.0]" 0.1 "[30 40 50 60
70 80 90 100]"
```

lein run -m simulation.runners.universal workload/planetlab_20_100_10_100 markov-multisize 0.2 results-planetlab-20-100-10-100.json "[1.0]" 0.1 "[30 40 50 60 70 80 90 100]"

lein run -m simulation.runners.universal workload/planetlab_20_100_10_100 markov-multisize 0.3 results-planetlab-20-100-10-100.json "[1.0]" 0.1 "[30 40 50 60 70 80 90 100]"

lein run -m simulation.runners.universal workload/planetlab_20_100_10_100 thr 0.8 results-planetlab-20-100-10-100.json

lein run -m simulation.runners.universal workload/planetlab_20_100_10_100 thr 0.9 results-planetlab-20-100-10-100.json

lein run -m simulation.runners.universal workload/planetlab_20_100_10_100 thr 1.0 results-planetlab-20-100-10-100.json

lein run -m simulation.runners.universal workload/planetlab_20_100_10_100 mad 2.0 results-planetlab-20-100-10-100.json

lein run -m simulation.runners.universal workload/planetlab_20_100_10_100 mad 3.0 results-planetlab-20-100-10-100.json

lein run -m simulation.runners.universal workload/planetlab_20_100_10_100 iqr 1.0 results-planetlab-20-100-10-100.json

lein run -m simulation.runners.universal workload/planetlab_20_100_10_100 iqr 2.0 results-planetlab-20-100-10-100.json

lein run -m simulation.runners.universal workload/planetlab_20_100_10_100 lr 1.0 results-planetlab-20-100-10-100.json

lein run -m simulation.runners.universal workload/planetlab_20_100_10_100 lr 1.1 results-planetlab-20-100-10-100.json

lein run -m simulation.runners.universal workload/planetlab_20_100_10_100 lr 1.2 results-planetlab-20-100-10-100.json

lein run -m simulation.runners.universal workload/planetlab_20_100_10_100 lrr 1.0 results-planetlab-20-100-10-100.json

lein run -m simulation.runners.universal workload/planetlab_20_100_10_100 lrr 1.1 results-planetlab-20-100-10-100.json

lein run -m simulation.runners.universal workload/planetlab_20_100_10_100 lrr 1.2 results-planetlab-20-100-10-100.json

lein run -m simulation.runners.universal workload/planetlab_20_100_10_100 otf-limit 0.1 results-planetlab-20-100-10-100.json

lein run -m simulation.runners.universal workload/planetlab_20_100_10_100 otf-limit 0.2 results-planetlab-20-100-10-100.json

lein run -m simulation.runners.universal workload/planetlab_20_100_10_100 otf-limit 0.3 results-planetlab-20-100-10-100.json

Examples to run POTFT (otf):

#!/bin/sh

```
rm results-planetlab-20-100-10-100.json
rm results-planetlab-20-100-10-100.csv

echo "[" > results-planetlab-20-100-10-100.json

lein run -m simulation.runners.universal workload/planetlab_20_100_10_100 otf
0.08 results-planetlab-20-100-10-100.json

lein run -m simulation.runners.universal workload/planetlab_20_100_10_100 otf
0.16 results-planetlab-20-100-10-100.json

lein run -m simulation.runners.universal workload/planetlab_20_100_10_100 otf
0.24 results-planetlab-20-100-10-100.json

lein run -m simulation.runners.universal workload/planetlab_20_100_10_100 otf
0.085 results-planetlab-20-100-10-100.json

lein run -m simulation.runners.universal workload/planetlab_20_100_10_100 otf
0.17 results-planetlab-20-100-10-100.json

lein run -m simulation.runners.universal workload/planetlab_20_100_10_100 otf
0.255 results-planetlab-20-100-10-100.json

lein run -m simulation.runners.universal workload/planetlab_20_100_10_100 otf
0.09 results-planetlab-20-100-10-100.json

lein run -m simulation.runners.universal workload/planetlab_20_100_10_100 otf
0.18 results-planetlab-20-100-10-100.json

lein run -m simulation.runners.universal workload/planetlab_20_100_10_100 otf
0.27 results-planetlab-20-100-10-100.json
```

Examples to run POTFT with VM quiescing (otf-q):

```
lein run -m simulation.runners.universal workload/planetlab_20_100_10_100 otf-q
0.08 results-planetlab-20-100-10-100.json

lein run -m simulation.runners.universal workload/planetlab_20_100_10_100 otf-q
0.16 results-planetlab-20-100-10-100.json

lein run -m simulation.runners.universal workload/planetlab_20_100_10_100 otf-q
0.24 results-planetlab-20-100-10-100.json

lein run -m simulation.runners.universal workload/planetlab_20_100_10_100 otf-q
0.085 results-planetlab-20-100-10-100.json

lein run -m simulation.runners.universal workload/planetlab_20_100_10_100 otf-q
0.17 results-planetlab-20-100-10-100.json

lein run -m simulation.runners.universal workload/planetlab_20_100_10_100 otf-q
0.255 results-planetlab-20-100-10-100.json

lein run -m simulation.runners.universal workload/planetlab_20_100_10_100 otf-q
0.09 results-planetlab-20-100-10-100.json

lein run -m simulation.runners.universal workload/planetlab_20_100_10_100 otf-q
0.18 results-planetlab-20-100-10-100.json

lein run -m simulation.runners.universal workload/planetlab_20_100_10_100 otf-q
0.27 results-planetlab-20-100-10-100.json
```

Appendix O: Result of PM Overloading Detection Algorithms

Table O.1 presents mean, standard deviation and 95% confidence interval of results for all PM overload detection algorithms. Mean, standard deviation and 95% confidence interval have been calculated using exel formulas, which are AVERAGE(data range), STDEV(data range), [AVERAGE-CONFIDENCE(0.05,stdev,100), AVERAGE+CONFIDENCE(0.05, stdev, 100)], respectively. 95% confidence interval is used instead of using paired t-test with 95% confidence interval that used in [4], because 95% confidence interval is commonly used in related research [66-68]. Moreover, mean, standard deviation and 95% confidence interval meet the target that is comparing between means and checking how much the results of algorithms differ from the mean value.

Table O.1: Mean, standard deviation and 95% confidence interval of results for all PM overloading detection algorithms results

Algorithm	Parameter	Mean	Standard Deviation	95% confidence interval
THR_80	Time Until migration	408	168.523	[374.97, 441.03]
	OTF	0.087781	0.042443	[0.07946252, 0.09609948]
THR_90	Time Until migration	1380	1522.259	[1081.64, 1678.36]
	OTF	0.081529	0.105349	[0.06088117, 0.10217683]
THR_100	Time Until migration	4896	2860.254	[4335.4, 5456.6]
	OTF	0.116181	0.114632	[0.0937127, 0.1386473]
MAD_2	Time Until migration	9627	886.7663	[9453.2, 9800.8]
	OTF	0.054761	0.021318	[0.0505817, 0.0589383]
MAD_3	Time Until migration	9252	378.0813	[9177.9, 9326.1]
	OTF	0.053946	0.021987	[0.04963582, 0.05825418]
IQR_1	Time Until migration	9552	651.7389	[9424.26, 9679.74]
	OTF	0.054763	0.021326	[0.05058238, 0.05894162]

IQR_2	Time Until migration	9075	130.5582	[9049.41, 9100.59]
	OTF	0.053282	0.021412	[0.04908552, 0.05747848]
LR_1.2	Time Until migration	7551	9512.982	[5686.49, 9415.51]
	OTF	0.067663	0.059515	[0.05599827, 0.07932773]
LR_1.1	Time Until migration	26712	22396.95	[22322.28, 31101.72]
	OTF	0.147006	0.087209	[0.12991235, 0.16409765]
LR_1.0	Time Until migration	68328	20273.5	[64354.47, 72301.53]
	OTF	0.278749	0.083054	[0.26247091, 0.29502709]
LRR_1.2	Time Until migration	8109	10296.89	[6090.85, 10127.15]
	OTF	0.071861	0.062847	[0.05954321, 0.08417879]
LRR_1.1	Time Until migration	26121	22351.43	[21740.2, 30501.8]
	OTF	0.141437	0.088527	[0.12408503, 0.15878697]
LRR_1.0	Time Until migration	66306	22161.34	[61962.46, 70649.54]
	OTF	0.27593	0.089958	[0.25829775, 0.29356025]
OTF_10	Time Until migration	18969	8129.593	[17375.63, 20562.37]
	OTF	0.109056	0.005248	[0.10802661, 0.11008339]
OTF_20	Time Until migration	40254	15425.17	[37230.72, 43277.28]
	OTF	0.204643	0.00275	[0.20410401, 0.20518199]
OTF_30	Time Until migration	69630	21613.38	[65393.86, 73866.14]
	OTF	0.26347	0.03927	[0.2557732, 0.2711668]
MHOD_10	Time Until migration	15888	7553.81	[14407.48, 17368.51]
	OTF	0.090635	0.005937	[0.08947161, 0.09179901]
MHOD_20	Time Until migration	38352	15522.98	[35309.55, 41394.44]
	OTF	0.195637	0.003468	[0.19495731, 0.19631671]
MHOD_30	Time Until migration	69144	21873.51	[64856.87, 73431.12]
	OTF	0.261223	0.036976	[0.25397541, 0.26846981]
POTF_8	Time Until migration	17418	7517.364	[15944.62, 18891.38]
	OTF	0.084445	0.002045	[0.08404319, 0.08484481]
POTF_16	Time Until	37926	14558.34	[35072.62, 40779.38]

	migration			
	OTF	0.164582	0.002682	[0.16405653, 0.16510747]
POTF_24	Time Until migration	63072	22819.13	[58599.53, 67544.47]
	OTF	0.234311	0.014052	[0.2315561, 0.2370639]
POTF_8.5	Time Until migration	17982	7741.942	[16464.61, 19499.39]
	OTF	0.08977	0.002868	[0.08920688, 0.09033112]
POTF_17	Time Until migration	38721	14892.12	[35802.2, 41639.8]
	OTF	0.174508	0.002668	[0.17398428, 0.17502972]
POTF_25.5	Time Until migration	66102	23431.03	[61509.6, 70694.4]
	OTF	0.243496	0.020763	[0.23942672, 0.24756528]
POTF_9	Time Until migration	18135	7826.688	[16601, 19669]
	OTF	0.095007	0.002513	[0.09451446, 0.09549954]
POTF_18	Time Until migration	39321	15090.66	[36363.28, 42278.72]
	OTF	0.187003	0.002972	[0.1864195, 0.1875845]
POTF_27	Time Until migration	67518	22891.28	[63031.39, 72004.61]
	OTF	0.251127	0.027302	[0.24577491, 0.25647709]
POTFT_8_Q_40	Time Until migration	46626	3672.72	[45906.16, 47345.84]
	OTF	0.084581	0.002153	[0.0841582, 0.0850018]
	Number of quiescings	2.93	1.401695	[2.6553, 3.2047]
POTFT_16_Q_40	Time Until migration	51264	7384.207	[49816.72, 52711.28]
	OTF	0.165865	0.003615	[0.16515647, 0.16657353]
	Number of quiescings	0.88	0.902074	[0.7032, 1.0568]
POTFT_24_Q_40	Time Until migration	67755	16995.58	[64423.93, 71086.07]
	OTF	0.235287	0.014818	[0.23238192, 0.23819008]
	Number of quiescings	0.28	0.551948	[0.1718, 0.3882]
POTFT_8.5_Q_40	Time Until migration	47292	5813.53	[46152.57, 48431.43]
	OTF	0.08964	0.002927	[0.0890665, 0.0902135]
	Number of quiescings	2.68	1.286134	[2.4279, 2.9321]

POTFT_17_Q_40	Time Until migration	51288	7952.246	[49729.39, 52846.61]
	OTF	0.17636	0.004084	[0.17555855, 0.17715945]
	Number of quiescings	0.78	0.811284	[0.621, 0.939]
POTFT_25.5_Q_40	Time Until migration	70464	17608.5	[67012.8, 73915.2]
	OTF	0.244231	0.021402	[0.24003629, 0.24842571]
	Number of quiescings	0.2	0.426401	[0.116, 0.284]
POTFT_9_Q_40	Time Until migration	46947	4663.572	[46032.96, 47861.04]
	OTF	0.094911	0.002343	[0.094451, 0.095369]
	Number of quiescings	2.51	1.159023	[2.2828, 2.7372]
POTFT_18_Q_40	Time Until migration	51081	7886.704	[49535.23, 52626.77]
	OTF	0.186989	0.003085	[0.18638355, 0.18759245]
	Number of quiescings	0.74	0.773553	[0.5884, 0.8916]
POTFT_27_Q_40	Time Until migration	71913	16958.02	[68589.29, 75236.71]
	OTF	0.252014	0.028165	[0.24649296, 0.25753304]
	Number of quiescings	0.19	0.394277	[0.1127, 0.2673]
POTFT_8_Q_80	Time Until migration	84954	2111.303	[84540.19, 85367.81]
	OTF	0.084395	0.002183	[0.08396714, 0.08482286]
	Number of quiescings	5.97	2.438972	[5.492, 6.448]
POTFT_16_Q_80	Time Until migration	85476	1732.25	[85136.49, 85815.51]
	OTF	0.166906	0.003352	[0.16624902, 0.16756298]
	Number of quiescings	2.49	1.520865	[2.1919, 2.7881]
POTFT_24_Q_80	Time Until migration	86115	1178.886	[85883.94, 86346.06]
	OTF	0.237248	0.01613	[0.23408658, 0.24040942]
	Number of quiescings	1.28	1.239746	[1.037, 1.523]
POTFT_8.5_Q_80	Time Until migration	84783	1727.055	[84444.5, 85121.5]
	OTF	0.089938	0.002871	[0.08937529, 0.09050071]
	Number of quiescings	5.54	2.244724	[5.1, 5.98]

POTFT_17_Q_80	Time Until migration	85674	1493.915	[85381.2, 85966.8]
	OTF	0.177821	0.002897	[0.17725339, 0.17838861]
	Number of quiescings	2.38	1.502725	[2.0855, 2.6745].
POTFT_25.5_Q_80	Time Until migration	86136	1071.94	[85925.9, 86346.1]
	OTF	0.246338	0.023144	[0.24180086, 0.25087314]
	Number of quiescings	0.9	1.020002	[0.7, 1.1]
POTFT_9_Q_80	Time Until migration	85179	1938.16	[84799.13, 85558.87]
	OTF	0.095201	0.002371	[0.09473629, 0.09566571]
	Number of quiescings	5.13	2.077562	[4.7228, 5.5372]
POTFT_18_Q_80	Time Until migration	85920	1313.219	[85662.61, 86177.39]
	OTF	0.18694	0.003102	[0.1863322, 0.1875478]
	Number of quiescings	2.28	1.239746	[2.037, 2.523]
POTFT_27_Q_80	Time Until migration	86229	827.9822	[86066.72, 86391.28]
	OTF	0.253842	0.029826	[0.24799521, 0.25968679]
	Number of quiescings	0.81	0.906709	[0.6323, 0.9877]

Table O.2 shows time until migration anf OTF results of output file for all PM overload detection algorithms. The output files are MS excel files. Results in Table O.2 are copied manually from MS excel files.

Table O.2: Results of PM Overloading Detection Algorithms

Algorithm	**THR_80**		**THR_90**	
Simulation ID	**Time until migration**	**OTF**	**Time until migration**	**OTF**
1	330	0.090909	2730	0.010989
2	330	0.090909	330	0.090909
3	630	0.047619	4230	0.078014
4	330	0.090909	1230	0.268293
5	630	0.047619	2730	0.010989
6	330	0.090909	630	0.047619
7	330	0.090909	330	0.090909

8	330	0.090909	1530	0.019608
9	330	0.090909	930	0.032258
10	330	0.090909	930	0.032258
11	330	0.090909	330	0.090909
12	330	0.090909	330	0.090909
13	330	0.090909	1830	0.180328
14	630	0.047619	630	0.047619
15	330	0.090909	330	0.090909
16	330	0.090909	1230	0.268293
17	930	0.032258	930	0.032258
18	630	0.047619	630	0.047619
19	330	0.090909	630	0.047619
20	330	0.090909	1530	0.019608
21	930	0.354839	930	0.354839
22	630	0.047619	630	0.047619
23	330	0.090909	630	0.52381
24	330	0.090909	330	0.090909
25	330	0.090909	330	0.090909
26	330	0.090909	1530	0.019608
27	330	0.090909	1230	0.02439
28	330	0.090909	330	0.090909
29	330	0.090909	8730	0.037801
30	330	0.090909	930	0.032258
31	330	0.090909	2130	0.15493
32	330	0.090909	630	0.52381
33	330	0.090909	930	0.032258
34	330	0.090909	630	0.047619
35	330	0.090909	630	0.047619
36	330	0.090909	1530	0.019608
37	330	0.090909	330	0.090909
38	930	0.032258	1530	0.019608
39	630	0.047619	930	0.032258
40	330	0.090909	2430	0.012346
41	330	0.090909	2730	0.120879
42	330	0.090909	330	0.090909
43	330	0.090909	330	0.090909
44	330	0.090909	930	0.032258
45	330	0.090909	630	0.047619
46	330	0.090909	330	0.090909
47	630	0.047619	1230	0.02439
48	930	0.032258	1230	0.02439
49	330	0.090909	1530	0.215686

50	630	0.047619	630	0.047619
51	330	0.090909	330	0.090909
52	630	0.047619	1230	0.02439
53	930	0.354839	930	0.354839
54	330	0.090909	330	0.090909
55	330	0.090909	2430	0.135802
56	330	0.090909	1830	0.016393
57	330	0.090909	1230	0.02439
58	330	0.090909	3330	0.009009
59	330	0.090909	330	0.090909
60	330	0.090909	330	0.090909
61	630	0.047619	1830	0.016393
62	330	0.090909	330	0.090909
63	330	0.090909	630	0.52381
64	330	0.090909	930	0.032258
65	330	0.090909	1230	0.02439
66	330	0.090909	930	0.032258
67	330	0.090909	1830	0.016393
68	330	0.090909	4230	0.007092
69	630	0.047619	930	0.354839
70	330	0.090909	630	0.047619
71	630	0.047619	3930	0.083969
72	330	0.090909	330	0.090909
73	330	0.090909	1830	0.016393
74	330	0.090909	930	0.032258
75	330	0.090909	330	0.090909
76	330	0.090909	1230	0.02439
77	330	0.090909	630	0.047619
78	330	0.090909	330	0.090909
79	930	0.032258	1230	0.02439
80	330	0.090909	630	0.047619
81	330	0.090909	2130	0.014085
82	630	0.047619	930	0.032258
83	330	0.090909	930	0.032258
84	330	0.090909	330	0.090909
85	330	0.090909	1530	0.019608
86	330	0.090909	330	0.090909
87	330	0.090909	3330	0.099099
88	330	0.090909	930	0.032258
89	330	0.090909	630	0.047619
90	330	0.090909	330	0.090909
91	330	0.090909	6630	0.004525

92	330	0.090909	330	0.090909
93	630	0.047619	1830	0.016393
94	330	0.090909	630	0.047619
95	330	0.090909	2130	0.014085
96	330	0.090909	1230	0.02439
97	330	0.090909	630	0.047619
98	330	0.090909	630	0.047619
99	330	0.090909	7830	0.003831
100	330	0.090909	6330	0.004739
Mean	408	0.087781	1380	0.081529
Standard deviation	168.523	0.042443	1522.259	0.105349

Table O.2: Results of PM Overloading Detection Algorithms (continued)

Algorithm	**THR_100**		**MAD_2**	
Simulation ID	**Time until migration**	**OTF**	**Time until migration**	**OTF**
1	4230	0.078014	9030	0.069767
2	930	0.354839	9030	0.069767
3	4230	0.078014	9330	0.067524
4	1230	0.268293	10530	0.031339
5	6930	0.047619	9030	0.036545
6	1530	0.215686	9030	0.069767
7	6930	0.047619	9030	0.036545
8	2730	0.120879	9030	0.069767
9	1830	0.180328	9330	0.067524
10	3930	0.083969	9030	0.069767
11	9030	0.036545	9030	0.036545
12	4830	0.068323	9030	0.069767
13	1830	0.180328	9030	0.069767
14	9330	0.03537	9330	0.03537
15	2430	0.135802	9030	0.069767
16	1230	0.268293	9330	0.067524
17	7230	0.045643	10830	0.058172
18	5430	0.060773	9030	0.069767
19	7830	0.042146	10230	0.061584
20	1830	0.180328	9030	0.069767
21	930	0.354839	9630	0.034268
22	9030	0.036545	9030	0.036545
23	630	0.52381	9030	0.069767
24	5130	0.064327	9030	0.069767
25	7530	0.043825	9630	0.034268

26	2430	0.135802	9330	0.067524
27	6630	0.049774	11130	0.02965
28	9330	0.03537	9330	0.03537
29	8730	0.037801	11130	0.02965
30	2730	0.120879	10230	0.061584
31	2130	0.15493	9930	0.063444
32	630	0.52381	9630	0.065421
33	5430	0.060773	9030	0.069767
34	1830	0.180328	9030	0.069767
35	2130	0.15493	9030	0.069767
36	3930	0.083969	9630	0.096573
37	7830	0.042146	9030	0.036545
38	8130	0.04059	9030	0.036545
39	3330	0.099099	11130	0.056604
40	7230	0.045643	9330	0.03537
41	2730	0.120879	9030	0.069767
42	5730	0.057592	9030	0.036545
43	930	0.354839	9330	0.067524
44	6930	0.047619	9030	0.069767
45	4530	0.072848	14730	0.022403
46	5730	0.057592	9630	0.065421
47	8730	0.037801	9630	0.065421
48	5430	0.060773	9330	0.03537
49	1530	0.215686	9030	0.069767
50	3030	0.108911	9630	0.034268
51	9630	0.034268	9330	0.003215
52	2730	0.120879	9930	0.033233
53	930	0.354839	10830	0.058172
54	5130	0.064327	9630	0.096573
55	2430	0.135802	9030	0.036545
56	3630	0.090909	9030	0.036545
57	7230	0.045643	11430	0.055118
58	3630	0.090909	11430	0.028871
59	630	0.52381	9630	0.096573
60	6630	0.049774	9330	0.067524
61	5430	0.060773	10530	0.031339
62	6930	0.047619	9030	0.069767
63	630	0.52381	9030	0.069767
64	5430	0.060773	9330	0.067524
65	3030	0.108911	9930	0.063444
66	11730	0.028133	10230	0.002933
67	6930	0.047619	9630	0.065421

68	12330	0.026764	9030	0.003322
69	930	0.354839	10530	0.059829
70	3930	0.083969	9930	0.063444
71	3930	0.083969	9030	0.069767
72	2430	0.135802	9030	0.069767
73	4230	0.078014	9030	0.069767
74	5730	0.057592	9330	0.067524
75	4230	0.078014	9630	0.065421
76	6930	0.047619	10530	0.059829
77	3030	0.108911	9030	0.069767
78	2430	0.135802	9030	0.069767
79	8730	0.037801	10230	0.032258
80	8430	0.039146	9930	0.033233
81	6930	0.047619	9030	0.069767
82	1830	0.180328	9330	0.099678
83	1830	0.180328	10830	0.058172
84	5730	0.057592	9030	0.069767
85	10230	0.032258	9030	0.003322
86	5730	0.057592	10230	0.032258
87	3330	0.099099	9930	0.063444
88	4530	0.072848	9330	0.03537
89	5130	0.064327	9630	0.034268
90	3630	0.090909	9630	0.065421
91	10830	0.030471	9930	0.003021
92	4530	0.072848	9030	0.069767
93	7230	0.045643	9030	0.069767
94	6630	0.049774	9630	0.034268
95	4230	0.078014	9030	0.069767
96	3630	0.090909	9030	0.069767
97	6930	0.047619	9030	0.069767
98	2430	0.135802	11730	0.079284
99	8430	0.039146	12330	0.026764
100	9330	0.03537	9330	0.03537
Mean	4896	0.116181	9627	0.054761
Standard deviation	2860.254	0.114632	886.7663	0.021318

Table O.2: Results of PM Overloading Detection Algorithms (continued)

Algorithm	**MAD_3**		**IQR_1**	
Simulation ID	**Time until migration**	**OTF**	**Time until migration**	**OTF**
1	9030	0.069767	9030	0.069767

2	9030	0.069767	9030	0.069767
3	9330	0.067524	9330	0.067524
4	10230	0.032258	10530	0.031339
5	9030	0.036545	9930	0.033233
6	9030	0.069767	9030	0.069767
7	9030	0.036545	9330	0.03537
8	9030	0.069767	9030	0.069767
9	9330	0.067524	9330	0.067524
10	9030	0.069767	9030	0.069767
11	9030	0.036545	9030	0.036545
12	9030	0.069767	9030	0.069767
13	9030	0.069767	9030	0.069767
14	9030	0.003322	9330	0.03537
15	9030	0.069767	9030	0.069767
16	9330	0.067524	9330	0.067524
17	9030	0.069767	10830	0.058172
18	9030	0.069767	9030	0.069767
19	9330	0.067524	10230	0.061584
20	9030	0.069767	9030	0.069767
21	9330	0.03537	9330	0.03537
22	9030	0.036545	9030	0.036545
23	9030	0.069767	9030	0.069767
24	9030	0.069767	9030	0.069767
25	9030	0.036545	9630	0.034268
26	9330	0.067524	9330	0.067524
27	9630	0.034268	11130	0.02965
28	9030	0.003322	9330	0.03537
29	10230	0.032258	11130	0.02965
30	9330	0.067524	10230	0.061584
31	9030	0.069767	9930	0.063444
32	9630	0.065421	9630	0.065421
33	9030	0.069767	9030	0.069767
34	9030	0.069767	9030	0.069767
35	9030	0.069767	9030	0.069767
36	9630	0.096573	9630	0.096573
37	9030	0.036545	9030	0.036545
38	9030	0.036545	9030	0.036545
39	9330	0.067524	10830	0.058172
40	9330	0.03537	9330	0.03537
41	9030	0.069767	9030	0.069767
42	9030	0.036545	9030	0.036545
43	9030	0.069767	9330	0.067524

44	9030	0.069767	9030	0.069767
45	9030	0.036545	10530	0.031339
46	9330	0.03537	9630	0.065421
47	9330	0.03537	9330	0.03537
48	9330	0.03537	9330	0.03537
49	9030	0.069767	9030	0.069767
50	9630	0.034268	9630	0.034268
51	9330	0.003215	9330	0.003215
52	9330	0.03537	9930	0.033233
53	9630	0.065421	9630	0.065421
54	9030	0.069767	9630	0.096573
55	9030	0.036545	9030	0.036545
56	9030	0.036545	9030	0.036545
57	9630	0.034268	11430	0.055118
58	11430	0.028871	11430	0.028871
59	9630	0.096573	9630	0.096573
60	9030	0.069767	9330	0.067524
61	9930	0.033233	10530	0.031339
62	9030	0.069767	9030	0.069767
63	9030	0.069767	9030	0.069767
64	9330	0.067524	9330	0.067524
65	9030	0.036545	9930	0.063444
66	9030	0.003322	10230	0.002933
67	9330	0.067524	9630	0.065421
68	9030	0.003322	10230	0.002933
69	9630	0.065421	9630	0.065421
70	9630	0.065421	9930	0.063444
71	9030	0.069767	9030	0.069767
72	9030	0.069767	9030	0.069767
73	9030	0.069767	9030	0.069767
74	9030	0.069767	9330	0.067524
75	9030	0.069767	9630	0.065421
76	10230	0.061584	10230	0.061584
77	9030	0.069767	9030	0.069767
78	9030	0.069767	9030	0.069767
79	9330	0.03537	10230	0.032258
80	9030	0.036545	9930	0.033233
81	9030	0.069767	9030	0.069767
82	9330	0.099678	9330	0.099678
83	10230	0.061584	10830	0.058172
84	9030	0.069767	9030	0.069767
85	9030	0.003322	9030	0.003322

86	9330	0.03537	10530	0.031339
87	9930	0.063444	9930	0.063444
88	9030	0.036545	9330	0.03537
89	9630	0.034268	9630	0.034268
90	9030	0.069767	9030	0.069767
91	9330	0.003215	9930	0.003021
92	9030	0.069767	9030	0.069767
93	9030	0.069767	9030	0.069767
94	9630	0.034268	9630	0.034268
95	9030	0.069767	9030	0.069767
96	9030	0.069767	9030	0.069767
97	9030	0.069767	9030	0.069767
98	9330	0.067524	11730	0.079284
99	9030	0.036545	10230	0.032258
100	9330	0.03537	9330	0.03537
Mean	9252	0.053946	9552	0.054763
Standard deviation	378.0813	0.021987	651.7389	0.021326

Table O.2: Results of PM Overloading Detection Algorithms (continued)

Algorithm	**IQR_2**		**LR_1.2**	
Simulation ID	**Time until migration**	**OTF**	**Time until migration**	**OTF**
1	9030	0.069767	3330	0.009009
2	9030	0.069767	3330	0.189189
3	9030	0.069767	7230	0.087137
4	9630	0.034268	3930	0.083969
5	9030	0.036545	3330	0.009009
6	9030	0.069767	15330	0.099804
7	9030	0.036545	8730	0.037801
8	9030	0.069767	11730	0.079284
9	9330	0.067524	3330	0.099099
10	9030	0.069767	3930	0.083969
11	9030	0.036545	3330	0.009009
12	9030	0.069767	6630	0.049774
13	9030	0.069767	7230	0.087137
14	9030	0.003322	6630	0.004525
15	9030	0.069767	4230	0.078014
16	9030	0.069767	3330	0.099099
17	9030	0.069767	3330	0.009009
18	9030	0.069767	3330	0.009009
19	9030	0.069767	3630	0.008264

20	9030	0.069767	17430	0.156627
21	9030	0.036545	15930	0.05838
22	9030	0.036545	3330	0.009009
23	9030	0.069767	3330	0.189189
24	9030	0.069767	3330	0.009009
25	9030	0.036545	10830	0.030471
26	9030	0.069767	3330	0.099099
27	9630	0.034268	9630	0.034268
28	9030	0.003322	3330	0.009009
29	9030	0.036545	23430	0.180538
30	9030	0.069767	3330	0.099099
31	9030	0.069767	3930	0.160305
32	9030	0.069767	3330	0.189189
33	9030	0.069767	3330	0.009009
34	9030	0.069767	3630	0.090909
35	9030	0.069767	3330	0.099099
36	9030	0.069767	3330	0.009009
37	9030	0.036545	11730	0.079284
38	9030	0.036545	3330	0.009009
39	9030	0.069767	3330	0.099099
40	9030	0.036545	3330	0.009009
41	9030	0.069767	3330	0.099099
42	9030	0.036545	7830	0.042146
43	9030	0.069767	10530	0.059829
44	9030	0.069767	3330	0.009009
45	9030	0.036545	3330	0.009009
46	9030	0.036545	3330	0.009009
47	9030	0.036545	6930	0.004329
48	9030	0.036545	3330	0.009009
49	9030	0.069767	9930	0.063444
50	9030	0.036545	3330	0.099099
51	9030	0.003322	3330	0.009009
52	9330	0.03537	3330	0.099099
53	9330	0.067524	3930	0.160305
54	9030	0.069767	3330	0.009009
55	9030	0.036545	12630	0.049881
56	9030	0.036545	3630	0.090909
57	9630	0.034268	3330	0.009009
58	9030	0.036545	3330	0.009009
59	9330	0.067524	3930	0.160305
60	9030	0.069767	12930	0.118329
61	9330	0.03537	15030	0.021956

62	9030	0.069767	59430	0.202423
63	9030	0.069767	3330	0.099099
64	9030	0.036545	9030	0.036545
65	9030	0.036545	3330	0.099099
66	9030	0.003322	8130	0.00369
67	9330	0.067524	9030	0.069767
68	9030	0.003322	11130	0.002695
69	9030	0.069767	3930	0.160305
70	9030	0.069767	3930	0.083969
71	9030	0.069767	3330	0.009009
72	9030	0.069767	3330	0.189189
73	9030	0.069767	7530	0.043825
74	9030	0.069767	10830	0.058172
75	9030	0.069767	9330	0.067524
76	9030	0.069767	9330	0.067524
77	9030	0.069767	3330	0.099099
78	9030	0.069767	3330	0.099099
79	9030	0.036545	8430	0.003559
80	9030	0.036545	3330	0.009009
81	9030	0.069767	9630	0.065421
82	9030	0.069767	6630	0.095023
83	9030	0.069767	3330	0.099099
84	9030	0.069767	15330	0.138943
85	9030	0.003322	3330	0.009009
86	9330	0.03537	6930	0.047619
87	9030	0.069767	3330	0.099099
88	9030	0.036545	4530	0.072848
89	9330	0.03537	3630	0.008264
90	9030	0.069767	3330	0.009009
91	9330	0.003215	24630	0.025579
92	9030	0.069767	3330	0.009009
93	9030	0.069767	3330	0.009009
94	9030	0.036545	3630	0.008264
95	9030	0.069767	6930	0.090909
96	9030	0.069767	3630	0.090909
97	9030	0.069767	8130	0.04059
98	9030	0.069767	3630	0.173554
99	9030	0.036545	24930	0.085439
100	9030	0.003322	69930	0.283569
Mean	9075	0.053282	7551	0.067663
Standard deviation	130.5582	0.021412	9512.982	0.059515

Table O.2: Results of PM Overloading Detection Algorithms (continued)

Algorithm	**LR_1.1**		**LR_1.0**	
Simulation ID	**Time until migration**	**OTF**	**Time until migration**	**OTF**
1	8130	0.077491	28830	0.261186
2	4230	0.148936	86430	0.219021
3	19830	0.198185	46530	0.252095
4	43230	0.271339	73530	0.379845
5	9330	0.03537	76530	0.235594
6	18030	0.084859	73530	0.334965
7	31830	0.161169	86430	0.25026
8	15030	0.081836	86430	0.21555
9	22830	0.067017	76830	0.269816
10	19830	0.092284	86430	0.253731
11	82230	0.299526	86430	0.302326
12	28530	0.263933	35730	0.252729
13	22530	0.147803	25530	0.130435
14	16230	0.168207	42630	0.310345
15	12930	0.095128	75930	0.300672
16	3330	0.099099	43530	0.427981
17	25230	0.036861	80730	0.305091
18	3330	0.009009	15930	0.13371
19	7530	0.003984	24030	0.200999
20	18630	0.146538	76530	0.243434
21	68730	0.441292	75930	0.458712
22	27030	0.123196	66330	0.357757
23	3330	0.189189	26130	0.173364
24	13530	0.068736	76530	0.251274
25	31830	0.142319	86430	0.229434
26	3330	0.099099	76530	0.251274
27	12930	0.025522	70530	0.302425
28	15930	0.114878	75930	0.245358
29	27930	0.215897	43530	0.317712
30	3330	0.099099	55230	0.397067
31	19230	0.095164	86430	0.257202
32	3330	0.189189	69930	0.305019
33	22530	0.254328	74730	0.510237
34	6630	0.095023	82230	0.248449
35	9330	0.067524	86430	0.444637
36	3930	0.083969	69630	0.379578
37	14130	0.065817	86430	0.205137
38	18930	0.175911	45030	0.360426

39	3330	0.099099	66030	0.46388
40	46530	0.219858	67830	0.239275
41	27330	0.187706	68730	0.532955
42	43830	0.178645	80130	0.239985
43	17430	0.122203	72630	0.248245
44	10230	0.061584	76530	0.247354
45	9630	0.034268	68730	0.257966
46	3630	0.008264	74730	0.26937
47	8430	0.003559	86430	0.222492
48	3330	0.009009	37230	0.371475
49	19230	0.063963	84030	0.214566
50	66030	0.304861	72630	0.29368
51	70230	0.209739	86430	0.201666
52	21330	0.07173	86430	0.229434
53	26730	0.158249	66330	0.271823
54	6930	0.047619	20430	0.148311
55	53130	0.265951	68430	0.298553
56	78030	0.219531	84030	0.210996
57	56730	0.190904	82530	0.276627
58	29730	0.112008	86430	0.205137
59	23730	0.190898	75930	0.261162
60	14130	0.129512	86430	0.239847
61	55830	0.193982	86430	0.205137
62	63630	0.203206	75030	0.212315
63	18930	0.223455	77130	0.49436
64	22830	0.185283	53430	0.270073
65	34230	0.298861	44430	0.338285
66	27330	0.132821	76830	0.285435
67	23130	0.20882	26730	0.191919
68	30930	0.097963	84930	0.424232
69	58530	0.226038	76530	0.255194
70	68430	0.237177	73230	0.229824
71	19230	0.157566	86430	0.309268
72	3330	0.189189	81330	0.295463
73	39330	0.244851	69330	0.268715
74	77130	0.295994	86430	0.298855
75	75030	0.312275	79530	0.305922
76	18330	0.116203	86430	0.201666
77	3330	0.099099	84030	0.268119
78	68430	0.245945	77130	0.225982
79	24030	0.076155	86430	0.319681
80	25230	0.143876	64230	0.378795

81	12330	0.124088	15330	0.158513
82	17430	0.122203	70230	0.261
83	68130	0.255834	75330	0.251294
84	16530	0.128857	18330	0.116203
85	15030	0.081836	47130	0.325271
86	9930	0.033233	86430	0.208608
87	23130	0.169909	78630	0.328501
88	8430	0.039146	76230	0.268005
89	37230	0.18614	69630	0.220164
90	3630	0.090909	78030	0.219531
91	68730	0.205587	79230	0.208633
92	14430	0.064449	19230	0.079563
93	3330	0.009009	36030	0.4005
94	23730	0.190898	52230	0.373923
95	8130	0.077491	76830	0.258102
96	3630	0.090909	86430	0.337036
97	73530	0.249286	86430	0.264144
98	19230	0.157566	80130	0.303632
99	53430	0.320606	59130	0.304921
100	75630	0.289964	84330	0.288509
Mean	26712	0.147006	68328	0.278749
Standard deviation	22396.95	0.087209	20273.5	0.083054

Table O.2: Results of PM Overloading Detection Algorithms (continued)

Algorithm	**LRR_1.2**		**LRR_1.1**	
Simulation ID	**Time until migration**	**OTF**	**Time until migration**	**OTF**
1	3330	0.009009	8130	0.077491
2	3330	0.189189	19830	0.107413
3	7230	0.087137	19830	0.198185
4	3930	0.083969	43230	0.271339
5	3330	0.009009	9330	0.03537
6	12030	0.077307	18930	0.080824
7	8730	0.037801	32130	0.159664
8	12030	0.077307	15030	0.081836
9	3330	0.099099	17430	0.08778
10	3930	0.083969	8730	0.072165
11	3330	0.009009	12030	0.052369
12	6630	0.049774	24030	0.213483
13	7230	0.087137	22230	0.149798
14	14430	0.126819	16830	0.16221

15	4230	0.078014	12630	0.097387
16	3330	0.099099	3330	0.099099
17	3330	0.009009	25530	0.036428
18	3330	0.009009	3330	0.009009
19	3330	0.009009	7230	0.004149
20	17130	0.15937	18630	0.146538
21	15930	0.05838	69930	0.442299
22	3330	0.009009	27030	0.123196
23	3330	0.189189	6630	0.095023
24	3330	0.009009	13230	0.070295
25	10830	0.030471	25830	0.152149
26	3330	0.099099	3330	0.099099
27	9930	0.033233	59730	0.281768
28	3330	0.009009	14430	0.106029
29	24030	0.17603	32730	0.275894
30	3330	0.099099	3330	0.099099
31	3930	0.160305	19230	0.095164
32	3330	0.189189	7230	0.087137
33	3330	0.009009	8130	0.04059
34	3330	0.099099	6930	0.090909
35	4530	0.072848	26730	0.135802
36	3330	0.009009	3930	0.083969
37	11730	0.079284	14130	0.065817
38	3330	0.009009	18930	0.175911
39	3330	0.099099	3330	0.099099
40	3330	0.009009	46830	0.21845
41	3330	0.099099	14430	0.064449
42	7830	0.042146	73830	0.2564
43	16830	0.12656	20430	0.148311
44	3330	0.009009	15030	0.081836
45	3330	0.009009	8130	0.04059
46	3330	0.009009	3630	0.008264
47	6930	0.004329	8430	0.003559
48	3330	0.009009	3330	0.009009
49	9930	0.063444	19230	0.063963
50	3630	0.090909	65730	0.306253
51	3330	0.009009	72030	0.204498
52	3330	0.099099	21330	0.07173
53	3930	0.160305	27330	0.154775
54	3330	0.009009	6930	0.047619
55	11430	0.055118	52230	0.26479
56	3630	0.090909	8430	0.039146

57	3330	0.009009	57630	0.187923
58	3330	0.009009	3630	0.090909
59	3330	0.099099	24030	0.188514
60	12930	0.118329	14130	0.129512
61	15330	0.021526	56130	0.192945
62	58530	0.20041	63630	0.203206
63	3330	0.099099	19230	0.219969
64	19230	0.173167	23130	0.182879
65	3330	0.099099	34230	0.298861
66	8430	0.003559	68730	0.28852
67	10530	0.059829	24030	0.200999
68	11130	0.002695	31830	0.095193
69	3930	0.160305	57030	0.231983
70	3930	0.083969	68730	0.236141
71	3330	0.009009	19830	0.152799
72	3330	0.189189	3330	0.189189
73	7530	0.043825	21030	0.201141
74	21030	0.115549	77130	0.295994
75	12930	0.071926	75630	0.309798
76	9330	0.067524	18630	0.114332
77	3330	0.099099	3330	0.099099
78	3330	0.099099	68430	0.245945
79	7530	0.003984	22230	0.068826
80	3330	0.009009	25230	0.143876
81	9330	0.067524	9330	0.067524
82	4230	0.078014	15630	0.136276
83	3330	0.099099	68130	0.255834
84	16530	0.128857	17430	0.122203
85	3330	0.009009	16230	0.075786
86	7230	0.045643	9930	0.033233
87	3330	0.099099	24330	0.173859
88	4530	0.072848	8430	0.039146
89	3630	0.008264	37230	0.18614
90	3330	0.009009	3630	0.090909
91	24930	0.025271	69630	0.20293
92	3630	0.008264	14730	0.063136
93	3330	0.009009	3330	0.009009
94	3630	0.008264	24330	0.18619
95	6930	0.090909	7830	0.08046
96	3630	0.090909	3630	0.090909
97	8130	0.04059	73530	0.249286
98	3630	0.173554	24330	0.161529

99	46230	0.26671	54030	0.317046
100	69930	0.283569	75030	0.288285
Mean	8109	0.071861	26121	0.141437
Standard deviation	10296.89	0.062847	22351.43	0.088527

Table O.2: Results of PM Overloading Detection Algorithms (continued)

Algorithm	**LRR_1.0**		**OTF_10**	
Simulation ID	**Time until migration**	**OTF**	**Time until migration**	**OTF**
1	29130	0.258496	10830	0.113573
2	86430	0.219021	13530	0.113082
3	46530	0.252095	12930	0.118329
4	73530	0.379845	19830	0.107413
5	76830	0.234674	23730	0.102402
6	73830	0.333604	12930	0.118329
7	86430	0.25026	20730	0.10275
8	86430	0.21555	16530	0.110708
9	76530	0.270874	31530	0.105614
10	86430	0.253731	10830	0.113573
11	86430	0.302326	20130	0.105812
12	35730	0.252729	11430	0.107612
13	25230	0.131986	10230	0.120235
14	42630	0.310345	11730	0.104859
15	75930	0.300672	11430	0.107612
16	43830	0.425051	18330	0.116203
17	81030	0.303961	37830	0.103886
18	15930	0.13371	10530	0.116809
19	23730	0.20354	12930	0.118329
20	76830	0.242483	11730	0.104859
21	75930	0.458712	24930	0.109507
22	66330	0.357757	22230	0.109312
23	26130	0.173364	13830	0.110629
24	76230	0.252263	17430	0.104991
25	86430	0.229434	17430	0.104991
26	76830	0.250293	19830	0.107413
27	71730	0.301547	35430	0.102456
28	78030	0.238754	13230	0.115646
29	43230	0.319917	18930	0.11252
30	56430	0.399256	14130	0.10828
31	86430	0.257202	13830	0.110629
32	69330	0.307659	13830	0.110629

33	75330	0.506173	11730	0.104859
34	81930	0.249359	12930	0.118329
35	86430	0.444637	17130	0.10683
36	70530	0.378988	10530	0.116809
37	44730	0.121395	20130	0.105812
38	44730	0.362844	14130	0.10828
39	66030	0.46388	13230	0.115646
40	67830	0.239275	28530	0.106204
41	69030	0.530639	16830	0.108734
42	80130	0.239985	25230	0.108205
43	74730	0.245283	13230	0.115646
44	77430	0.244479	17730	0.103215
45	69030	0.256845	29430	0.102956
46	75030	0.268293	11730	0.104859
47	86430	0.222492	12930	0.118329
48	37830	0.365583	10830	0.113573
49	85830	0.213562	28530	0.106204
50	71730	0.293183	31230	0.106628
51	86430	0.201666	32430	0.102683
52	86430	0.229434	28830	0.105099
53	66330	0.271823	17130	0.10683
54	20430	0.148311	11730	0.104859
55	68730	0.29725	17430	0.104991
56	8430	0.039146	31530	0.105614
57	82830	0.275625	35730	0.101595
58	86430	0.205137	23730	0.102402
59	75930	0.261162	9930	0.123867
60	16230	0.131238	10530	0.116809
61	86430	0.205137	30330	0.109792
62	75030	0.212315	34530	0.105126
63	77730	0.494404	11430	0.107612
64	53130	0.271598	14430	0.106029
65	44130	0.340585	16830	0.108734
66	79530	0.287061	17130	0.10683
67	26730	0.191919	11430	0.107612
68	86430	0.427282	23730	0.102402
69	75030	0.260296	27630	0.109663
70	73230	0.229824	34230	0.106047
71	86430	0.309268	11430	0.107612
72	3330	0.189189	12630	0.12114
73	70230	0.265271	11430	0.107612
74	86430	0.298855	16830	0.108734

75	79530	0.305922	18630	0.114332
76	86430	0.201666	14430	0.106029
77	84930	0.265277	11730	0.104859
78	76830	0.226865	11430	0.107612
79	86430	0.319681	34830	0.10422
80	63630	0.382367	19530	0.109063
81	9630	0.065421	10830	0.113573
82	70230	0.261	12630	0.12114
83	75630	0.250298	29130	0.104016
84	69630	0.396812	13230	0.115646
85	46830	0.327354	19530	0.109063
86	86430	0.208608	35430	0.102456
87	78330	0.329759	11730	0.104859
88	75330	0.271207	28230	0.107333
89	72030	0.221158	19830	0.107413
90	78030	0.219531	13830	0.110629
91	78630	0.210225	37230	0.10556
92	18630	0.082126	21930	0.110807
93	38730	0.411309	15630	0.117083
94	51930	0.376083	14430	0.106029
95	78330	0.25316	17430	0.104991
96	86430	0.337036	11430	0.107612
97	86430	0.264144	20430	0.104258
98	81630	0.301727	12930	0.118329
99	58830	0.306476	26130	0.104478
100	81930	0.285976	40530	0.104367
Mean	66306	0.27593	18969	0.109056
Standard deviation	22161.34	0.089958	8129.593	0.005248

Table O.2: Results of PM Overloading Detection Algorithms (continued)

Algorithm	**OTF_20**		**OTF_30**	
Simulation ID	**Time until migration**	**OTF**	**Time until migration**	**OTF**
1	14130	0.214437	35430	0.305673
2	54630	0.203734	86430	0.219021
3	18930	0.207607	56730	0.301957
4	32130	0.206349	47730	0.302326
5	51930	0.202773	86430	0.21555
6	38130	0.20535	48930	0.301042
7	47430	0.203036	86430	0.25026
8	61230	0.201372	86430	0.21555

9	51630	0.203951	86430	0.274557
10	56130	0.203634	86430	0.253731
11	56130	0.203634	81930	0.300622
12	20430	0.207048	86430	0.212079
13	36630	0.205569	47730	0.302326
14	19230	0.204368	30930	0.301649
15	36330	0.207267	51330	0.3045
16	26130	0.207807	30630	0.304603
17	51630	0.203951	65730	0.301689
18	38430	0.203747	86430	0.25026
19	15930	0.20904	37830	0.302141
20	38130	0.20535	86430	0.222492
21	40230	0.202088	48330	0.30478
22	35730	0.202351	50430	0.303986
23	32130	0.206349	86430	0.208608
24	33630	0.206066	86430	0.229434
25	54930	0.202622	86430	0.229434
26	37230	0.202256	86430	0.229434
27	42630	0.204785	63930	0.300798
28	27330	0.20966	86430	0.225963
29	26430	0.205448	35430	0.305673
30	27930	0.205156	36930	0.301381
31	49230	0.201706	86430	0.257202
32	32130	0.206349	52830	0.301533
33	16230	0.205176	28530	0.305994
34	53430	0.202695	86430	0.239847
35	40230	0.202088	54930	0.300928
36	15930	0.20904	50730	0.302188
37	67230	0.201249	86430	0.205137
38	21630	0.209431	28530	0.305994
39	21630	0.209431	37530	0.304556
40	42630	0.204785	86430	0.205137
41	31230	0.20269	40830	0.301984
42	48630	0.204195	86430	0.232905
43	34230	0.202454	86430	0.225963
44	39330	0.206712	86430	0.236376
45	45930	0.203135	86430	0.25026
46	19230	0.204368	86430	0.25026
47	41730	0.202013	86430	0.222492
48	17130	0.211909	25530	0.306698
49	59130	0.20345	86430	0.212079
50	43230	0.201943	61530	0.302779

51	62130	0.203283	86430	0.201666
52	71430	0.202016	86430	0.229434
53	29430	0.204893	86430	0.243318
54	38130	0.20535	57930	0.30088
55	34230	0.202454	58830	0.301377
56	64230	0.201308	86430	0.212079
57	60930	0.202363	86430	0.274557
58	65730	0.201278	86430	0.205137
59	17730	0.204738	86430	0.232905
60	42630	0.204785	86430	0.239847
61	52230	0.201608	86430	0.205137
62	56730	0.201481	86430	0.201666
63	15930	0.20904	31830	0.302545
64	26430	0.205448	86430	0.25026
65	24330	0.210851	33930	0.301503
66	45930	0.203135	86430	0.271086
67	19230	0.204368	86430	0.225963
68	46830	0.205637	59430	0.303382
69	46830	0.205637	86430	0.232905
70	55830	0.204729	86430	0.219021
71	61830	0.20427	83730	0.301326
72	48930	0.202943	86430	0.28497
73	17730	0.204738	86430	0.253731
74	49230	0.201706	69630	0.302025
75	42330	0.206237	63330	0.303648
76	78930	0.201824	86430	0.201666
77	35430	0.204064	86430	0.264144
78	49230	0.201706	86430	0.212079
79	55230	0.201521	77730	0.301428
80	29730	0.202825	37530	0.304556
81	17130	0.211909	29430	0.30683
82	36630	0.205569	86430	0.25026
83	56730	0.201481	86430	0.222492
84	33630	0.206066	42930	0.301188
85	26430	0.205448	35430	0.305673
86	74730	0.201124	86430	0.208608
87	39630	0.205148	49830	0.301626
88	38730	0.202169	86430	0.243318
89	39330	0.206712	86430	0.212079
90	42630	0.204785	86430	0.205137
91	65130	0.203132	86430	0.205137
92	63930	0.202252	86430	0.225963

93	20130	0.210134	26430	0.307605
94	27330	0.20966	32730	0.303391
95	39930	0.203606	86430	0.246789
96	24930	0.205776	68730	0.301615
97	51930	0.202773	86430	0.264144
98	30930	0.204656	63930	0.300798
99	34230	0.202454	48630	0.302899
100	52230	0.201608	86430	0.295384
Mean	40254	0.204643	69630	0.26347
Standard deviation	15425.17	0.00275	21613.38	0.03927

Table O.2: Results of PM Overloading Detection Algorithms (continued)

Algorithm	**POTF_8**		**POTF_16**	
Simulation ID	**Time until migration**	**OTF**	**Time until migration**	**OTF**
1	9930	0.081568	13230	0.171764
2	12330	0.089335	51930	0.163191
3	12030	0.084345	18030	0.167124
4	18330	0.085823	30630	0.165492
5	22230	0.081512	49230	0.162624
6	12030	0.084132	35430	0.163664
7	19230	0.0822	44730	0.164053
8	15330	0.088566	57930	0.160493
9	29130	0.083646	49530	0.162753
10	9930	0.081057	52830	0.164333
11	18330	0.084121	52230	0.163315
12	10230	0.085874	18930	0.166467
13	9330	0.085703	34530	0.166305
14	10530	0.082944	18330	0.163086
15	10230	0.085874	34530	0.167472
16	16830	0.082934	24930	0.167908
17	35130	0.083005	48930	0.164793
18	9630	0.084103	36330	0.163609
19	12030	0.084984	15030	0.166814
20	10530	0.083363	36030	0.165923
21	23130	0.086729	38430	0.163691
22	20730	0.087231	33930	0.161881
23	12630	0.087729	30030	0.166317
24	15930	0.083783	32130	0.165265
25	15930	0.082943	51330	0.162705
26	18330	0.085716	35130	0.162412

27	32130	0.081862	40530	0.164443
28	12330	0.082953	25830	0.169615
29	17730	0.089228	24630	0.164153
30	12930	0.085866	25830	0.164535
31	12630	0.088171	45630	0.161163
32	12630	0.088061	30030	0.16673
33	10530	0.082944	15330	0.16373
34	12030	0.084558	49530	0.162561
35	15630	0.084503	37530	0.161266
36	9630	0.083577	15030	0.168277
37	18330	0.083909	62730	0.160798
38	12930	0.085866	20430	0.167964
39	12030	0.082537	20430	0.168802
40	26430	0.08422	39630	0.165262
41	15330	0.086335	29130	0.163571
42	23430	0.086347	45630	0.16499
43	12030	0.082329	32730	0.161558
44	16230	0.082056	36630	0.165783
45	26730	0.081644	43230	0.163118
46	10830	0.082944	17730	0.165334
47	11730	0.084665	39930	0.162822
48	9630	0.081057	15930	0.170163
49	26430	0.08422	56430	0.163981
50	28830	0.085303	41130	0.161958
51	30330	0.08153	58530	0.163643
52	26430	0.083869	67530	0.163431
53	15630	0.084396	27930	0.163709
54	10530	0.083153	36330	0.164896
55	16230	0.082943	32730	0.161761
56	28530	0.084491	60330	0.160442
57	32730	0.081175	58230	0.162093
58	21930	0.081512	61830	0.160419
59	9030	0.088627	16530	0.164609
60	9630	0.083577	40830	0.164852
61	27930	0.086736	48330	0.160883
62	31530	0.083155	53130	0.163199
63	10230	0.085659	14730	0.168277
64	13230	0.084717	25230	0.164153
65	15630	0.086009	22530	0.169735
66	15630	0.084716	42930	0.164133
67	10530	0.085659	18030	0.163495
68	21630	0.081512	44430	0.164304

69	25530	0.087621	44130	0.164921
70	31530	0.084732	51930	0.164397
71	10530	0.085659	58230	0.164641
72	11430	0.086131	46530	0.162151
73	10530	0.085551	16230	0.165633
74	15330	0.086226	46830	0.160962
75	17430	0.081599	39330	0.166227
76	13530	0.084505	75030	0.160854
77	10830	0.083573	33330	0.162843
78	10230	0.085336	46830	0.170845
79	32130	0.082438	52830	0.162023
80	17730	0.08725	27930	0.162058
81	9930	0.081671	15930	0.170375
82	11730	0.087112	34830	0.166511
83	27030	0.082693	53730	0.161588
84	12330	0.082745	31530	0.164235
85	17730	0.08725	24930	0.16518
86	32130	0.081964	70230	0.162106
87	10530	0.082839	37230	0.165964
88	26430	0.0849	36330	0.171641
89	18330	0.084856	36930	0.165577
90	12330	0.088171	39930	0.163214
91	34530	0.08392	60630	0.163115
92	20430	0.08787	59430	0.162409
93	14130	0.083456	18930	0.168528
94	13230	0.084293	26130	0.168986
95	15930	0.083048	37230	0.162681
96	10230	0.085336	23430	0.164209
97	18630	0.08299	49230	0.163435
98	12030	0.084878	29430	0.164134
99	24030	0.082851	31830	0.162166
100	36930	0.083181	49530	0.170561
Mean	0.084445	0.084445	38045	0.164582
Standard deviation	0.002045	0.002045	14561.12	0.002682

Table O.2: Results of PM Overloading Detection Algorithms (continued)

Algorithm	**POTF_24**		**POTF_8.5**	
Simulation ID	**Time until migration**	**OTF**	**Time until migration**	**OTF**
1	29730	0.24307	10230	0.092131
2	86430	0.219021	12930	0.091624

3	48030	0.243977	12330	0.094399
4	39030	0.242905	18930	0.087237
5	86430	0.21555	22830	0.086018
6	40530	0.243979	12330	0.094739
7	69930	0.242671	19530	0.086412
8	86430	0.21555	15630	0.089912
9	69930	0.243094	30330	0.089032
10	76830	0.243628	10230	0.092022
11	64530	0.243059	19230	0.088882
12	86430	0.212079	10530	0.087708
13	38430	0.244264	9630	0.095457
14	27030	0.243698	11130	0.088292
15	44430	0.243971	10830	0.087708
16	23730	0.243541	17430	0.094487
17	55230	0.243496	36030	0.087784
18	67230	0.243931	9930	0.094419
19	33030	0.24339	12330	0.095534
20	86430	0.222492	11130	0.088396
21	38130	0.243332	23730	0.089147
22	44130	0.243049	21330	0.088884
23	86430	0.208608	12930	0.089424
24	86430	0.229434	16230	0.088928
25	86430	0.229434	16230	0.088928
26	86430	0.229434	18930	0.087546
27	53730	0.243422	33330	0.086268
28	86430	0.225963	12330	0.094145
29	30930	0.243964	18030	0.091708
30	32730	0.243706	13530	0.087421
31	72630	0.243764	12930	0.090061
32	42630	0.244109	13230	0.089955
33	22530	0.244077	11130	0.088187
34	86430	0.239847	12330	0.095648
35	48330	0.244306	16230	0.087173
36	42630	0.243444	9930	0.094643
37	86430	0.205137	18930	0.089094
38	25230	0.243802	13230	0.087733
39	32430	0.242512	12630	0.093257
40	86430	0.205137	27330	0.08953
41	31830	0.243063	15930	0.087997
42	86430	0.232905	23730	0.087672
43	86430	0.225963	12330	0.092591
44	86430	0.236376	16530	0.087526

45	68130	0.243094	28230	0.087101
46	76830	0.242781	10830	0.088921
47	86430	0.222492	12330	0.094285
48	19830	0.243338	10230	0.091695
49	86430	0.212079	27030	0.08953
50	53430	0.242605	29730	0.089888
51	86430	0.201666	30930	0.087178
52	86430	0.229434	27030	0.089334
53	71730	0.243635	16230	0.087173
54	50130	0.244626	11130	0.088292
55	51030	0.242853	16530	0.088823
56	86430	0.212079	30030	0.089244
57	69930	0.243282	33630	0.086051
58	86430	0.205137	22530	0.08653
59	86430	0.232905	9330	0.098578
60	86430	0.239847	9930	0.094868
61	86430	0.205137	28830	0.088747
62	86430	0.201666	32730	0.088306
63	27630	0.24364	10830	0.087398
64	68130	0.243557	13830	0.089383
65	27930	0.243803	15930	0.087683
66	67230	0.24365	16230	0.089737
67	86430	0.225963	10830	0.087708
68	47430	0.242914	22530	0.086325
69	86430	0.232905	26430	0.08938
70	86430	0.219021	32130	0.089928
71	67830	0.243617	10830	0.086985
72	70830	0.24403	12030	0.095594
73	71730	0.244626	10830	0.086778
74	56430	0.243494	15630	0.088519
75	52530	0.242757	17730	0.092746
76	86430	0.201666	13830	0.089489
77	68130	0.24292	11130	0.088816
78	86430	0.212079	10830	0.086985
79	64530	0.244588	33330	0.088171
80	32430	0.243814	18330	0.087948
81	25830	0.244259	10230	0.092131
82	68130	0.242784	12030	0.095827
83	86430	0.222492	27930	0.088414
84	34530	0.24343	12330	0.094034
85	30630	0.24297	18630	0.088786
86	86430	0.208608	33930	0.086985

87	38730	0.243399	10830	0.088921
88	68130	0.243052	26730	0.087274
89	86430	0.212079	18930	0.086721
90	86430	0.205137	13230	0.089424
91	86430	0.205137	35130	0.088776
92	86430	0.225963	20730	0.089461
93	21030	0.244438	14730	0.094753
94	27930	0.243198	13830	0.089489
95	75930	0.244325	16830	0.088193
96	53430	0.244495	10830	0.087398
97	76830	0.243505	19230	0.08862
98	49830	0.244489	12330	0.095421
99	40830	0.243382	24330	0.088806
100	68130	0.24408	38430	0.087773
Mean	63072	0.234311	17982	0.08977
Standard deviation	22819.13	0.014052	7741.942	0.002868

Table O.2: Results of PM Overloading Detection Algorithms (continued)

Algorithm	**POTF_17**		**POTF_25.5**	
Simulation ID	**Time until migration**	**OTF**	**Time until migration**	**OTF**
1	13530	0.183987	30930	0.259133
2	53430	0.175211	86430	0.219021
3	18030	0.176466	50430	0.260166
4	30930	0.176222	42630	0.259269
5	49530	0.172154	86430	0.21555
6	36630	0.176601	40530	0.260551
7	45930	0.17319	86430	0.25026
8	58530	0.170763	86430	0.21555
9	49530	0.174378	72030	0.26113
10	54330	0.172478	86430	0.253731
11	54330	0.174107	74130	0.258925
12	19530	0.177026	86430	0.212079
13	35730	0.175351	41430	0.260274
14	18330	0.174735	26130	0.259268
15	34830	0.17742	41430	0.260661
16	24930	0.17622	26430	0.259927
17	50130	0.175398	55230	0.261371
18	36630	0.173389	86430	0.25026
19	15330	0.177056	33330	0.259983
20	36930	0.176601	86430	0.222492

21	39030	0.170966	42330	0.259104
22	34830	0.171391	45030	0.260536
23	30630	0.175397	86430	0.208608
24	32730	0.17598	86430	0.229434
25	53130	0.17162	86430	0.229434
26	35430	0.172525	86430	0.229434
27	40530	0.174272	59430	0.259661
28	26130	0.178211	86430	0.225963
29	25830	0.17422	30630	0.260906
30	26430	0.176024	30330	0.259046
31	47730	0.172459	73830	0.258582
32	30930	0.177048	45930	0.260571
33	15330	0.175425	26430	0.260405
34	51630	0.171683	86430	0.239847
35	39030	0.172381	48030	0.259968
36	15330	0.177893	41730	0.258892
37	65730	0.171666	86430	0.205137
38	20730	0.178854	23430	0.259546
39	21030	0.179064	31230	0.258931
40	40530	0.173453	86430	0.205137
41	30330	0.173908	36030	0.260129
42	47130	0.174995	86430	0.232905
43	32430	0.172288	86430	0.225963
44	37530	0.176739	86430	0.236376
45	43830	0.172868	86430	0.25026
46	18630	0.17453	86430	0.25026
47	39930	0.171105	86430	0.222492
48	16230	0.18097	21630	0.260586
49	57630	0.172119	86430	0.212079
50	41430	0.172863	55530	0.260298
51	60030	0.173604	86430	0.201666
52	69330	0.172926	86430	0.229434
53	28530	0.175798	86430	0.243318
54	36930	0.176396	49530	0.261367
55	33330	0.172693	48630	0.260252
56	62130	0.170306	86430	0.212079
57	57930	0.173628	74730	0.259006
58	63630	0.171489	86430	0.205137
59	16830	0.17587	86430	0.232905
60	41730	0.174068	86430	0.239847
61	50730	0.171165	86430	0.205137
62	54030	0.172669	86430	0.201666

63	15330	0.177056	28530	0.260487
64	25230	0.17422	86430	0.25026
65	23430	0.181332	29430	0.258974
66	43830	0.174087	74430	0.260108
67	18330	0.17453	86430	0.225963
68	44430	0.176231	51030	0.260754
69	45330	0.174586	86430	0.232905
70	54330	0.175657	86430	0.219021
71	59430	0.173016	79530	0.25898
72	47130	0.173922	76830	0.260821
73	16830	0.174641	86430	0.253731
74	46830	0.173064	66330	0.261417
75	40530	0.176126	55230	0.260587
76	75030	0.170945	86430	0.201666
77	33630	0.173659	74430	0.260676
78	46830	0.170845	86430	0.212079
79	52230	0.1723	68730	0.259966
80	28230	0.171996	32130	0.260402
81	16230	0.181394	25830	0.26094
82	35730	0.175556	86430	0.25026
83	55230	0.170654	86430	0.222492
84	32130	0.175362	37230	0.259546
85	25530	0.174426	31530	0.259067
86	71730	0.172564	86430	0.208608
87	38130	0.176427	43530	0.259688
88	37830	0.171641	86430	0.243318
89	37530	0.175292	86430	0.212079
90	40830	0.173453	86430	0.205137
91	62730	0.172256	86430	0.205137
92	60930	0.171106	86430	0.225963
93	19530	0.180085	21930	0.259718
94	26130	0.179259	30330	0.259425
95	38430	0.175101	86430	0.246789
96	24030	0.17491	63330	0.260674
97	50130	0.173979	76530	0.260119
98	29430	0.174162	58230	0.259474
99	33030	0.172086	42330	0.259395
100	49830	0.170561	75030	0.261067
Mean	38721	0.174508	66102	0.243496
Standard deviation	14892.12	0.002668	23431.03	0.020763

Table O.2: Results of PM Overloading Detection Algorithms (continued)

Algorithm	**POTF_9**		**POTF_18**	
Simulation ID	**Time until migration**	**OTF**	**Time until migration**	**OTF**
1	10530	0.095402	13830	0.195567
2	12930	0.094989	53430	0.184787
3	12330	0.096415	18630	0.191829
4	18930	0.096349	31230	0.189429
5	22830	0.091138	51330	0.182901
6	12330	0.096415	37530	0.187279
7	19530	0.09042	46830	0.186793
8	15930	0.095145	59730	0.185463
9	30330	0.094207	50430	0.188655
10	10530	0.09948	55230	0.188362
11	19530	0.09248	54630	0.184289
12	11130	0.095451	20130	0.191106
13	9630	0.096801	36030	0.190152
14	11130	0.09301	18930	0.185362
15	10830	0.097604	35430	0.18882
16	17430	0.09761	25530	0.190559
17	36930	0.09142	51030	0.186819
18	9930	0.095176	37230	0.184595
19	12330	0.096415	15630	0.191062
20	11130	0.09322	37230	0.188306
21	23730	0.097461	39330	0.182485
22	21330	0.097943	34530	0.186568
23	13530	0.097907	31230	0.187365
24	16530	0.092287	33030	0.190611
25	16530	0.092392	53130	0.186412
26	18930	0.095383	36630	0.185267
27	33930	0.090161	42030	0.186764
28	12630	0.097143	26730	0.189742
29	18030	0.09692	25530	0.189629
30	13530	0.097669	27630	0.185461
31	13530	0.09669	48630	0.186578
32	13230	0.097652	31530	0.187365
33	11130	0.091752	15630	0.186915
34	12330	0.094664	52230	0.187493
35	16230	0.093156	39630	0.182081
36	9930	0.096309	15330	0.192107
37	19230	0.09449	66030	0.185753
38	13230	0.097452	21030	0.193096

39	12630	0.09535	21030	0.193724
40	27330	0.096008	41730	0.184921
41	15930	0.096665	30930	0.187285
42	24330	0.096735	47130	0.184796
43	12630	0.09535	33030	0.18646
44	17130	0.090829	38730	0.188315
45	28230	0.090601	44730	0.184447
46	11130	0.09343	18630	0.185362
47	12330	0.096415	41130	0.182822
48	10530	0.096616	16830	0.192837
49	27330	0.093884	57630	0.183715
50	30330	0.095432	41730	0.185182
51	30930	0.091388	60630	0.185191
52	27630	0.094379	69930	0.18242
53	16230	0.096468	28830	0.187682
54	11130	0.094583	37230	0.188922
55	16830	0.094282	33330	0.18565
56	29730	0.092306	62130	0.183593
57	34230	0.09042	60330	0.186174
58	22530	0.091138	64530	0.183565
59	9630	0.100927	17130	0.185902
60	9930	0.095176	42030	0.189426
61	29430	0.095958	50730	0.181447
62	32730	0.092511	54930	0.182743
63	10830	0.094483	15330	0.188554
64	13830	0.096274	25530	0.186136
65	15930	0.097426	23730	0.190187
66	16230	0.096254	45030	0.183431
67	10830	0.095882	18630	0.186997
68	22530	0.091138	45930	0.187953
69	26430	0.095311	45630	0.185074
70	33330	0.092897	54630	0.189374
71	10830	0.095344	60330	0.18752
72	12030	0.09753	47730	0.187113
73	10830	0.096205	17130	0.185697
74	15930	0.095578	48030	0.18557
75	17730	0.097182	41730	0.190769
76	13830	0.093624	77430	0.186082
77	11130	0.092801	34530	0.186515
78	10830	0.093945	48030	0.181939
79	33630	0.091089	54030	0.183183
80	18630	0.095648	28830	0.185788

81	10230	0.09959	16530	0.191778
82	12030	0.10458	35730	0.187479
83	27630	0.091326	55230	0.18375
84	12630	0.09535	33030	0.187726
85	18630	0.095529	25830	0.188602
86	33630	0.091185	72630	0.185839
87	11130	0.09301	38430	0.184633
88	26730	0.095848	37530	0.183165
89	18930	0.097531	38730	0.191002
90	13230	0.097354	41130	0.189222
91	35730	0.092893	63330	0.183632
92	21030	0.097165	61830	0.184454
93	15030	0.096535	19530	0.193954
94	13530	0.095638	27030	0.191419
95	16830	0.093442	39030	0.185893
96	10830	0.09642	24330	0.190137
97	19530	0.09279	51030	0.18574
98	12630	0.096415	30030	0.187055
99	24630	0.092985	33330	0.18565
100	39030	0.093095	50730	0.184875
Mean	18135	0.095007	39321	0.187003
Standard deviation	7826.688	0.002513	15090.66	0.002972

Table O.2: Results of PM Overloading Detection Algorithms (continued)

Algorithm	**POTF_27**		**MHOD_10**	
Simulation ID	**Time until migration**	**OTF**	**Time until migration**	**OTF**
1	30630	0.275687	9330	0.099678
2	86430	0.219021	11430	0.08136
3	49230	0.274974	10530	0.08831
4	41430	0.275118	19230	0.09516
5	86430	0.21555	22830	0.09329
6	45630	0.276043	10530	0.08831
7	86430	0.25026	20130	0.0909
8	86430	0.21555	10530	0.08831
9	79530	0.275458	11130	0.08355
10	86430	0.253731	10230	0.0909
11	72630	0.275068	13230	0.09297
12	86430	0.212079	11130	0.08355
13	44730	0.275508	9630	0.096573
14	29130	0.275086	11430	0.08136

15	49230	0.275311	11130	0.08355
16	28530	0.27541	13830	0.08893
17	61230	0.27548	36930	0.09829
18	86430	0.25026	9630	0.096573
19	32430	0.275505	11730	0.07928
20	86430	0.222492	11130	0.08355
21	46830	0.275565	23730	0.08975
22	47130	0.275777	21330	0.09985
23	86430	0.208608	9330	0.099678
24	86430	0.229434	15930	0.09604
25	86430	0.229434	15930	0.09604
26	86430	0.229434	19230	0.09516
27	55230	0.276204	35130	0.09479
28	86430	0.225963	11730	0.07928
29	29730	0.2748	17730	0.08629
30	34830	0.274873	11730	0.07928
31	86430	0.257202	10230	0.0909
32	48330	0.275494	12330	0.09975
33	25830	0.275133	10230	0.0909
34	86430	0.239847	9630	0.096573
35	47130	0.275563	10830	0.08587
36	43530	0.275856	9630	0.096573
37	86430	0.205137	10230	0.0909
38	27030	0.275785	13830	0.08893
39	36630	0.275109	11430	0.08136
40	86430	0.205137	26130	0.09299
41	37230	0.275186	15930	0.09604
42	86430	0.232905	20730	0.08827
43	86430	0.225963	12930	0.09512
44	86430	0.236376	14130	0.08704
45	86430	0.25026	28230	0.0967
46	86430	0.25026	10530	0.08831
47	86430	0.222492	11130	0.08355
48	24030	0.275654	10530	0.08831
49	86430	0.212079	23430	0.0909
50	59130	0.274811	30630	0.09892
51	86430	0.201666	25830	0.09407
52	86430	0.229434	11730	0.07928
53	86430	0.243318	14130	0.08704
54	51030	0.275382	9630	0.096573
55	54930	0.275097	17130	0.08931
56	86430	0.212079	30930	0.09796

57	78330	0.275943	34530	0.09643
58	86430	0.205137	19830	0.09228
59	86430	0.232905	9630	0.096573
60	86430	0.239847	9930	0.093655
61	86430	0.205137	27930	0.09774
62	86430	0.201666	31830	0.09519
63	30030	0.27549	11130	0.08355
64	86430	0.25026	11430	0.08136
65	29430	0.275093	15930	0.09604
66	77430	0.274982	15930	0.09604
67	86430	0.225963	10830	0.08587
68	52830	0.275421	20730	0.08827
69	86430	0.232905	20430	0.08957
70	86430	0.219021	30930	0.09796
71	79530	0.274798	9330	0.099678
72	82830	0.275779	11130	0.08355
73	86430	0.253731	9630	0.096573
74	66930	0.275567	11430	0.08136
75	54630	0.276251	10530	0.08831
76	86430	0.201666	12930	0.09512
77	86430	0.264144	10530	0.08831
78	86430	0.212079	10830	0.08587
79	76230	0.275247	13530	0.0909
80	36630	0.276276	19230	0.09516
81	27030	0.275946	9930	0.093655
82	86430	0.25026	9330	0.099678
83	86430	0.222492	14430	0.08523
84	36630	0.275156	10530	0.08831
85	32430	0.274887	11730	0.07928
86	86430	0.208608	22830	0.09329
87	45330	0.275174	10830	0.08587
88	86430	0.243318	23430	0.0909
89	86430	0.212079	17430	0.08777
90	86430	0.205137	10230	0.0909
91	86430	0.205137	36930	0.09829
92	86430	0.225963	16530	0.09255
93	24330	0.276227	13530	0.0909
94	30630	0.275421	13530	0.0909
95	86430	0.246789	11730	0.07928
96	64530	0.276177	9930	0.093655
97	86430	0.264144	10530	0.08831
98	60930	0.275934	11730	0.07928

99	46530	0.276081	23730	0.08975
100	77730	0.275499	40230	0.09768
Mean	67518	0.251127	15888	0.090635
Standard Deviation	22891.28	0.027302	7553.81	0.005937

Table O.2: Results of PM Overloading Detection Algorithms (continued)

Algorithm	**MHOD_20**		**MHOD_30**	
Simulation ID	**Time until migration**	**OTF**	**Time until migration**	**OTF**
1	13230	0.18367	35130	0.29974
2	52230	0.19586	86430	0.21902
3	17430	0.19104	56430	0.29824
4	31830	0.19886	46530	0.29722
5	50430	0.1969	86430	0.21555
6	37230	0.19419	48630	0.29673
7	47130	0.19796	86430	0.25026
8	60930	0.19743	86430	0.21555
9	49230	0.19561	86430	0.27455
10	54630	0.19824	86430	0.25373
11	55830	0.19935	79530	0.29837
12	20130	0.19523	86430	0.21207
13	35730	0.19395	47430	0.29791
14	18330	0.19803	30630	0.2948
15	35430	0.19559	50730	0.29627
16	25830	0.1986	30330	0.29772
17	51330	0.19929	65430	0.29848
18	38130	0.19748	86430	0.25026
19	15330	0.19765	36630	0.29565
20	37230	0.19419	86430	0.22249
21	39930	0.19609	47730	0.29604
22	33330	0.19891	50130	0.29982
23	18330	0.19803	86430	0.2086
24	29730	0.19273	86430	0.22943
25	52230	0.19586	86430	0.22943
26	35130	0.19726	86430	0.22943
27	42330	0.19914	63330	0.29891
28	26130	0.19632	86430	0.22596
29	25230	0.19143	33630	0.29527
30	26130	0.19632	36630	0.29565
31	47430	0.19671	86430	0.2572
32	29730	0.19273	52530	0.29754

33	15930	0.1902	28230	0.29861
34	49230	0.19561	86430	0.23984
35	39930	0.19609	54330	0.29872
36	15630	0.19385	50430	0.29803
37	62730	0.19655	86430	0.20513
38	16230	0.18669	27630	0.29424
39	21330	0.19831	37230	0.29895
40	41730	0.19482	86430	0.20513
41	26430	0.19409	40530	0.29681
42	46830	0.19923	86430	0.2329
43	24930	0.19374	86430	0.22596
44	38730	0.19442	86430	0.23637
45	45630	0.19789	86430	0.25026
46	18930	0.19175	86430	0.25026
47	41430	0.19623	86430	0.22249
48	13230	0.18367	25230	0.29845
49	52830	0.19931	86430	0.21207
50	42630	0.19774	60630	0.29737
51	59430	0.19737	86430	0.20166
52	71130	0.19865	86430	0.22943
53	29130	0.1967	86430	0.24331
54	33930	0.1954	56730	0.29666
55	32130	0.19701	58530	0.29779
56	63930	0.19755	86430	0.21207
57	60330	0.1994	86430	0.27455
58	61830	0.19941	86430	0.20513
59	16230	0.18669	86430	0.2329
60	36930	0.19577	86430	0.23984
61	49830	0.19927	86430	0.20513
62	53730	0.19597	86430	0.20166
63	13530	0.20177	29730	0.29364
64	22230	0.19028	86430	0.25026
65	23730	0.19089	32430	0.29694
66	44730	0.19517	86430	0.27108
67	17430	0.19104	86430	0.22596
68	46230	0.19532	59130	0.29984
69	45930	0.1966	86430	0.2329
70	53130	0.19819	86430	0.21902
71	60330	0.1994	83430	0.29881
72	48630	0.19802	75630	0.29789
73	14730	0.18533	86430	0.25373
74	44730	0.19517	68730	0.29725

75	41430	0.19623	62730	0.29698
76	77730	0.19722	86430	0.20166
77	34830	0.19896	86430	0.26414
78	44130	0.19782	86430	0.21207
79	54930	0.19716	71730	0.29736
80	27930	0.19441	37230	0.29895
81	16830	0.19786	25230	0.29845
82	35730	0.19395	86430	0.25026
83	54930	0.19716	86430	0.22249
84	33330	0.19891	42630	0.29627
85	26130	0.19632	35130	0.29974
86	73230	0.19705	86430	0.2086
87	39330	0.19908	49530	0.29739
88	38430	0.19594	86430	0.24331
89	35730	0.19395	86430	0.21207
90	39930	0.19609	86430	0.20513
91	62430	0.1975	86430	0.20513
92	62430	0.1975	86430	0.22596
93	17730	0.18781	26130	0.29965
94	21930	0.19288	32430	0.29694
95	37830	0.19904	86430	0.24678
96	21930	0.19288	68430	0.29855
97	50730	0.19574	86430	0.26414
98	29730	0.19273	61230	0.29936
99	33930	0.1954	48330	0.29857
100	50430	0.1969	86430	0.29538
Mean	38352	0.195637	69144	0.2612226
Standard Deviation	15522.98	0.003468	21873.51	0.036976

Table O.3 shows time until migration, OTF, and number of quiescing of POTFT algorithm with VM Quiescing. Results in Table O.3 are copied manually from MS excel files.

Table O.3: Results of POTFT with VM Quiescing

Alg.	**POTF_8_Q_40**			**POTF_16_Q_40**		
Sim. ID	Time until migration	OTF	No. of quiescings	Time until migration	OTF	No. of quiescings
1	47430	0.080733	5	45030	0.175761	3

2	47430	0.088433	4	51930	0.163191	0
3	46230	0.085159	4	42930	0.171016	2
4	44130	0.085802	2	46530	0.169348	1
5	53430	0.082293	2	49230	0.162624	0
6	46830	0.084943	4	49830	0.164198	1
7	46830	0.082989	2	44730	0.164053	0
8	47730	0.088549	3	58530	0.160493	0
9	50130	0.084451	1	49530	0.162753	0
10	47130	0.08103	5	52830	0.164333	0
11	44430	0.083263	2	52230	0.163315	0
12	48930	0.085853	5	43530	0.168676	2
13	44430	0.085682	5	49530	0.166846	1
14	40830	0.082919	4	42030	0.166888	2
15	48930	0.085853	5	52530	0.171372	1
16	40530	0.082087	2	60330	0.170135	2
17	53430	0.083803	1	49230	0.164793	0
18	45930	0.084913	5	53430	0.167423	1
19	46230	0.085805	4	58230	0.167355	3
20	40830	0.084165	4	54330	0.168125	1
21	55530	0.08757	2	57930	0.165866	1
22	49830	0.088077	2	48630	0.16241	1
23	48630	0.087711	4	45630	0.166858	1
24	49530	0.082929	3	47430	0.167459	1
25	49530	0.082096	3	51630	0.162705	0
26	44430	0.086544	2	51030	0.162942	1
27	48630	0.081836	1	40830	0.164443	0
28	48030	0.082928	4	40530	0.171863	1
29	42930	0.090097	2	58230	0.166334	2
30	49830	0.086697	4	60030	0.165071	2
31	48630	0.089028	4	45630	0.161163	0
32	48930	0.08717	4	44730	0.167271	1
33	40530	0.082097	4	59130	0.164264	3
34	46230	0.085374	4	49530	0.162561	0
35	48630	0.083642	3	51930	0.165027	1
36	45630	0.082724	5	58230	0.168822	3
37	44430	0.083886	2	63330	0.160798	0
38	49830	0.086697	4	46530	0.168508	2
39	46230	0.08333	4	46530	0.172732	2
40	45030	0.084197	1	59730	0.167456	1
41	47130	0.086315	3	43830	0.165744	1
42	40230	0.087183	1	45630	0.16499	0
43	46830	0.081487	4	49830	0.162087	1

44	50430	0.08203	3	55230	0.169646	1
45	45930	0.082427	1	43230	0.163118	0
46	41730	0.082097	4	40830	0.167529	2
47	45030	0.084642	4	60930	0.164986	1
48	45630	0.081834	5	42030	0.170712	2
49	45630	0.084197	1	56730	0.163981	0
50	49230	0.085281	1	41130	0.161958	0
51	52230	0.082312	1	59130	0.163643	0
52	45630	0.083845	1	68130	0.163431	0
53	48030	0.083536	3	42330	0.167525	1
54	40530	0.082304	4	55230	0.168739	1
55	49830	0.083741	3	49230	0.163911	1
56	49230	0.085306	1	60930	0.160442	0
57	54030	0.081147	1	58830	0.162093	0
58	53130	0.082293	2	61830	0.160419	0
59	43230	0.089489	5	41730	0.165145	2
60	45930	0.083553	5	41130	0.164852	0
61	45930	0.087576	1	48330	0.160883	0
62	50130	0.083955	1	53130	0.163199	0
63	48930	0.084788	5	57930	0.172195	3
64	41130	0.083855	3	57930	0.166334	2
65	48630	0.085136	3	51330	0.173686	2
66	48030	0.085534	3	42930	0.164133	0
67	40830	0.084788	4	42330	0.164028	2
68	52530	0.082293	2	44730	0.164304	0
69	43830	0.086734	1	44130	0.164921	0
70	51030	0.08555	1	51930	0.164397	0
71	40830	0.086487	4	58230	0.164641	0
72	44430	0.08611	4	46530	0.162151	0
73	40530	0.08553	4	58230	0.169492	3
74	47730	0.086206	3	46830	0.160962	0
75	42330	0.081572	2	59430	0.170099	1
76	42030	0.084482	3	75630	0.160854	0
77	41730	0.08272	4	49230	0.16664	1
78	48330	0.08616	5	47130	0.170845	0
79	49830	0.08323	1	53130	0.162023	0
80	42930	0.086366	2	42330	0.164212	1
81	47130	0.081644	5	41130	0.17434	2
82	45630	0.087093	4	51630	0.17039	1
83	46530	0.081848	1	54030	0.161588	0
84	47430	0.08272	4	47430	0.166416	1
85	42630	0.088097	2	40530	0.165718	1

86	46830	0.081938	1	70230	0.162106	0
87	40830	0.081993	4	52830	0.166504	1
88	45630	0.08572	1	54630	0.175635	1
89	44430	0.085675	2	56130	0.167775	1
90	48030	0.088154	4	60930	0.165383	1
91	51930	0.084729	1	61230	0.163115	0
92	49530	0.087852	2	60030	0.162409	0
93	43830	0.08426	3	43530	0.172452	2
94	41130	0.085106	3	41430	0.169532	1
95	49530	0.0822	3	56730	0.163213	1
96	48930	0.08616	5	59430	0.16639	2
97	44730	0.083788	2	49230	0.163435	0
98	46830	0.084014	4	44430	0.164669	1
99	40830	0.083647	1	48330	0.162696	1
100	56130	0.083981	1	49530	0.170561	0
Mean	46626	0.084581	2.93	51264	0.165865	0.88
Stdev	3672.72	0.002153	1.401695	7384.207	0.003615	0.902074

Table O.3: Results of POTFT with VM Quiescing (continued)

Alg.	**POTF_24_Q_40**			**POTF_8.5_Q_40**		
Sim. ID	Time until migration	OTF	No. of quiescings	Time until migration	OTF	No. of quiescings
1	46230	0.250127	1	40830	0.091079	4
2	86430	0.219021	0	42330	0.09239	3
3	48630	0.243977	0	48030	0.095189	4
4	61230	0.247556	1	47130	0.087102	2
5	86430	0.21555	0	41430	0.086735	1
6	41130	0.243979	0	48930	0.093659	4
7	70230	0.242671	0	48630	0.085424	2
8	86430	0.21555	0	50730	0.089775	3
9	70230	0.243094	0	54030	0.088015	1
10	77130	0.243628	0	40830	0.091882	4
11	65430	0.243059	0	46830	0.087867	2
12	86430	0.212079	0	41430	0.08844	4
13	60330	0.246524	1	48330	0.095313	5
14	42030	0.250773	1	43530	0.089029	4
15	44730	0.243971	0	42330	0.087572	4
16	49530	0.248203	2	42030	0.095279	2
17	55530	0.243496	0	61230	0.088516	1
18	68130	0.243931	0	48030	0.094276	5
19	51930	0.250457	1	48930	0.095391	4
20	86430	0.222492	0	42930	0.087386	4

21	59730	0.250397	1	42630	0.088128	1
22	44730	0.243049	0	53430	0.088747	2
23	86430	0.208608	0	41130	0.089286	3
24	86430	0.229434	0	40230	0.08967	2
25	86430	0.229434	0	51930	0.08967	3
26	86430	0.229434	0	46230	0.088277	2
27	54030	0.243422	0	57330	0.086134	1
28	86430	0.225963	0	47730	0.093072	4
29	48630	0.248635	1	44730	0.090661	2
30	51030	0.245961	1	42630	0.086421	3
31	73530	0.243764	0	41430	0.089032	3
32	43230	0.244109	0	40830	0.088927	3
33	46830	0.246336	2	43830	0.088923	4
34	86430	0.239847	0	48930	0.094558	4
35	48630	0.244306	0	40530	0.086176	2
36	43230	0.243444	0	48630	0.095436	5
37	86430	0.205137	0	48330	0.088957	2
38	53130	0.25088	2	42930	0.087598	3
39	51030	0.249553	1	50730	0.092193	4
40	86430	0.205137	0	48930	0.090278	1
41	49530	0.245312	1	40530	0.087861	2
42	86430	0.232905	0	42030	0.086669	1
43	86430	0.225963	0	48930	0.09245	4
44	86430	0.236376	0	40830	0.088257	2
45	69030	0.243094	0	49530	0.086105	1
46	77730	0.242781	0	44130	0.088784	4
47	86430	0.222492	0	48030	0.095075	4
48	41430	0.250403	2	41430	0.091555	4
49	86430	0.212079	0	48930	0.089393	1
50	53730	0.242605	0	51630	0.08975	1
51	86430	0.201666	0	54930	0.087043	1
52	86430	0.229434	0	48930	0.089197	1
53	72030	0.243635	0	40530	0.087901	2
54	50430	0.244626	0	44430	0.089029	4
55	51630	0.242853	0	40230	0.088686	2
56	86430	0.212079	0	52530	0.089106	1
57	70830	0.243282	0	59130	0.086769	1
58	86430	0.205137	0	55830	0.087251	2
59	86430	0.232905	0	46530	0.097456	5
60	86430	0.239847	0	48630	0.095663	5
61	86430	0.205137	0	50430	0.088611	1
62	86430	0.201666	0	59430	0.089043	1

63	43530	0.248304	1	43530	0.088127	4
64	68430	0.243557	0	44730	0.088361	3
65	43830	0.24847	1	49530	0.087548	3
66	68130	0.24365	0	51930	0.090487	3
67	86430	0.225963	0	41730	0.08844	4
68	47730	0.242914	0	57030	0.086191	2
69	86430	0.232905	0	45630	0.088359	1
70	86430	0.219021	0	54630	0.08979	1
71	68130	0.243617	0	42630	0.08685	4
72	71130	0.24403	0	48330	0.09545	4
73	72030	0.244626	0	42330	0.087502	4
74	56730	0.243494	0	48630	0.087507	3
75	52830	0.242757	0	43830	0.093522	2
76	86430	0.201666	0	43530	0.088466	3
77	69030	0.24292	0	42630	0.088679	4
78	86430	0.212079	0	43530	0.08685	4
79	65430	0.244588	0	59430	0.088035	1
80	50430	0.250893	1	46530	0.086943	2
81	40530	0.246519	1	42030	0.092902	4
82	69030	0.242784	0	48030	0.09663	4
83	86430	0.222492	0	48630	0.089152	1
84	53730	0.250498	1	49830	0.093892	4
85	47730	0.245219	1	47430	0.089528	2
86	86430	0.208608	0	58530	0.08599	1
87	60930	0.248059	1	41730	0.087905	4
88	69030	0.243052	0	46830	0.087139	1
89	86430	0.212079	0	47430	0.086587	2
90	86430	0.205137	0	41430	0.088402	3
91	86430	0.205137	0	63030	0.089517	1
92	86430	0.225963	0	51030	0.088439	2
93	44130	0.249117	2	48330	0.094609	3
94	43830	0.247854	1	44130	0.088466	3
95	76230	0.244325	0	40230	0.087185	2
96	53730	0.244495	0	42630	0.087263	4
97	77730	0.243505	0	46830	0.087607	2
98	50130	0.244489	0	50130	0.095277	4
99	41130	0.243382	0	42030	0.088669	1
100	68430	0.24408	0	69630	0.087637	1
Mean	67782	0.235455	0.28	47292	0.08964	2.68
Stdev	16994.21	0.014965	0.551948	5813.53	0.002927	1.286134

Table O.3: Results of POTFT with VM Quiescing (continued)

Alg.	POTF_17_Q_40			POTF_25.5_Q_40		
Sim. ID	Time until migration	OTF	No. of quiescings	Time until migration	OTF	No. of quiescings
1	47430	0.187536	3	55230	0.260974	1
2	54930	0.175211	0	86430	0.219021	0
3	43830	0.179875	2	50430	0.260166	0
4	47130	0.179626	1	42630	0.259269	0
5	51030	0.172154	0	86430	0.21555	0
6	53730	0.181759	1	40530	0.260551	0
7	47730	0.17319	0	86430	0.25026	0
8	59130	0.170763	0	86430	0.21555	0
9	50430	0.174378	0	72030	0.26113	0
10	56430	0.172478	0	86430	0.253731	0
11	54630	0.174107	0	74130	0.258925	0
12	44730	0.182196	2	86430	0.212079	0
13	53430	0.180473	1	41430	0.260274	0
14	42930	0.178111	2	45030	0.263669	1
15	54030	0.179092	1	41430	0.260661	0
16	60330	0.177882	2	46530	0.261773	1
17	50430	0.175398	0	55230	0.261371	0
18	54030	0.17674	1	86430	0.25026	0
19	41730	0.178725	2	59130	0.266962	1
20	57330	0.181759	1	86430	0.222492	0
21	61230	0.172581	1	42330	0.259104	0
22	51630	0.17301	1	45030	0.260536	0
23	47430	0.178785	1	86430	0.208608	0
24	48330	0.177639	1	86430	0.229434	0
25	54030	0.17162	0	86430	0.229434	0
26	52230	0.177566	1	86430	0.229434	0
27	41130	0.174272	0	59430	0.259661	0
28	41730	0.181652	1	86430	0.225963	0
29	40830	0.177587	1	54030	0.265334	1
30	40830	0.179424	1	54030	0.263443	1
31	48030	0.172459	0	73830	0.258582	0
32	47430	0.178716	1	45930	0.260571	0
33	60930	0.177079	3	45930	0.267396	1
34	52530	0.171683	0	86430	0.239847	0
35	55830	0.174008	1	48030	0.259968	0
36	41430	0.179569	2	41730	0.258892	0
37	66330	0.171666	0	86430	0.205137	0
38	48030	0.180539	2	41730	0.266514	1
39	48630	0.184292	2	54030	0.263326	1

40	40830	0.173453	0	86430	0.205137	0
41	46530	0.177269	1	64230	0.264545	1
42	48030	0.174995	0	86430	0.232905	0
43	50730	0.177323	1	86430	0.225963	0
44	58230	0.180153	1	86430	0.236376	0
45	45030	0.172868	0	86430	0.25026	0
46	43530	0.176177	2	86430	0.25026	0
47	61530	0.171105	1	86430	0.222492	0
48	44130	0.184463	2	59730	0.262436	2
49	59130	0.172119	0	86430	0.212079	0
50	42030	0.172863	0	55530	0.260298	0
51	60630	0.173604	0	86430	0.201666	0
52	71430	0.172926	0	86430	0.229434	0
53	44130	0.179194	1	86430	0.243318	0
54	56130	0.178058	1	49530	0.261367	0
55	52230	0.177739	1	48630	0.260252	0
56	62730	0.170306	0	86430	0.212079	0
57	58830	0.173628	0	74730	0.259006	0
58	65430	0.171489	0	86430	0.205137	0
59	43530	0.181007	2	86430	0.232905	0
60	42030	0.174068	0	86430	0.239847	0
61	52230	0.171165	0	86430	0.205137	0
62	56130	0.172669	0	86430	0.201666	0
63	41430	0.182227	2	50430	0.26748	1
64	59130	0.177587	2	86430	0.25026	0
65	55530	0.184831	2	51630	0.263371	1
66	44130	0.174087	0	74430	0.260108	0
67	44130	0.179629	2	86430	0.225963	0
68	45030	0.176231	0	51030	0.260754	0
69	46530	0.174586	0	86430	0.232905	0
70	56430	0.175657	0	86430	0.219021	0
71	61830	0.173016	0	79530	0.25898	0
72	48930	0.173922	0	76830	0.260821	0
73	41130	0.176289	2	86430	0.253731	0
74	48630	0.173064	0	66330	0.261417	0
75	42030	0.176126	0	55230	0.260587	0
76	78030	0.170945	0	86430	0.201666	0
77	50430	0.178732	1	74430	0.260676	0
78	48030	0.170845	0	86430	0.212079	0
79	54330	0.1723	0	68730	0.259966	0
80	42930	0.177022	1	56130	0.264822	1
81	41730	0.186689	2	44430	0.265369	1

82	53730	0.177211	1	86430	0.25026	0
83	57330	0.170654	0	86430	0.222492	0
84	50430	0.180485	1	63930	0.26139	1
85	41730	0.176071	1	54930	0.266022	1
86	72330	0.172564	0	86430	0.208608	0
87	55230	0.179835	1	43530	0.259688	0
88	58830	0.173262	1	86430	0.243318	0
89	58230	0.180412	1	86430	0.212079	0
90	42030	0.173453	0	86430	0.205137	0
91	63930	0.172256	0	86430	0.205137	0
92	62130	0.171106	0	86430	0.225963	0
93	45630	0.185342	2	40530	0.264127	1
94	41730	0.184493	1	53430	0.263828	1
95	57330	0.178485	1	86430	0.246789	0
96	40530	0.176559	1	63330	0.260674	0
97	51930	0.173979	0	76530	0.260119	0
98	45630	0.177528	1	58230	0.259474	0
99	48930	0.173711	1	42330	0.259395	0
100	51630	0.170561	0	75030	0.261067	0
Mean	51288	0.17636	0.78	70518	0.24436	0.2
Stdev	7952.246	0.004084	0.811284	17564.01	0.021526	0.426401

Table O.3: Results of POTFT with VM Quiescing (continued)

Alg.	**POTF_9_Q_40**			**POTF_18_Q_40**		
Sim. ID	Time until migration	OTF	No. of quiescings	Time until migration	OTF	No. of quiescings
1	44130	0.094898	4	47730	0.194367	3
2	43230	0.095432	3	54630	0.184787	0
3	41730	0.095905	3	45330	0.190655	2
4	48330	0.096797	2	48330	0.190153	1
5	58230	0.090664	2	51930	0.182901	0
6	41130	0.096862	3	53730	0.187997	1
7	50430	0.090849	2	47430	0.186793	0
8	41130	0.094643	2	61230	0.185463	0
9	53130	0.093713	1	51630	0.188655	0
10	44430	0.098949	4	56430	0.188362	0
11	49830	0.092915	2	55830	0.184289	0
12	46530	0.094948	4	46830	0.189937	2
13	50130	0.096288	5	53130	0.190878	1
14	46530	0.092524	4	44130	0.186074	2
15	45330	0.098054	4	54630	0.189543	1
16	44430	0.097092	2	41730	0.191287	1

17	57030	0.090944	1	52230	0.186819	0
18	41430	0.09562	4	54930	0.183472	1
19	41730	0.096862	3	42030	0.191791	2
20	46530	0.092732	4	57630	0.189027	1
21	41730	0.097911	1	60030	0.183189	1
22	54930	0.098395	2	50430	0.185431	1
23	45330	0.097386	3	48030	0.186223	1
24	42330	0.091806	2	49530	0.189446	1
25	42330	0.092828	2	53730	0.186412	0
26	48930	0.09488	2	54630	0.184139	1
27	52530	0.090589	1	42930	0.186764	0
28	42630	0.096628	3	42630	0.188583	1
29	46530	0.097369	2	40830	0.190354	1
30	45330	0.09715	3	42930	0.184332	1
31	45330	0.097138	3	49230	0.186578	0
32	44130	0.097133	3	48030	0.186223	1
33	47130	0.091274	4	41430	0.187632	2
34	41730	0.095106	3	53430	0.187493	0
35	41730	0.093593	2	56430	0.182784	1
36	42030	0.0958	4	41130	0.19284	2
37	49530	0.093993	2	67530	0.185753	0
38	44730	0.096935	3	48930	0.193831	2
39	42630	0.095794	3	48630	0.194461	2
40	47430	0.096454	1	42330	0.184921	0
41	41130	0.097113	2	47430	0.188003	1
42	42630	0.097183	1	47730	0.184796	0
43	42630	0.094848	3	51030	0.185324	1
44	43830	0.09126	2	59730	0.187166	1
45	49530	0.091031	1	45330	0.184447	0
46	47130	0.093868	4	43530	0.186074	2
47	41730	0.096862	3	41730	0.182822	0
48	44130	0.097064	4	44730	0.193572	2
49	48030	0.094324	1	58830	0.183715	0
50	53130	0.095877	1	42330	0.185182	0
51	53730	0.09182	1	61530	0.185191	0
52	48030	0.093883	1	71430	0.18242	0
53	41730	0.095957	2	44430	0.188401	1
54	47130	0.095025	4	57630	0.187769	1
55	43530	0.093787	2	51030	0.186363	1
56	52230	0.091825	1	63630	0.183593	0
57	57630	0.090849	1	61830	0.186174	0
58	57630	0.091569	2	65430	0.183565	0

59	50130	0.100385	5	44130	0.18477	2
60	41430	0.094674	4	42630	0.189426	0
61	49230	0.096404	1	51930	0.181447	0
62	52530	0.092946	1	55530	0.182743	0
63	45630	0.094924	4	41130	0.189275	2
64	46230	0.096721	3	41130	0.185002	1
65	40530	0.097876	2	54930	0.190914	2
66	41730	0.096701	2	45630	0.183431	0
67	45630	0.096328	4	44730	0.185857	2
68	57630	0.090664	2	46530	0.187953	0
69	46230	0.095755	1	46230	0.185074	0
70	54930	0.092412	1	55230	0.189374	0
71	45330	0.095788	4	61230	0.18752	0
72	50430	0.097012	4	48330	0.187113	0
73	45330	0.095696	4	42030	0.18641	2
74	40530	0.095073	2	49230	0.18557	0
75	45630	0.096666	2	42630	0.190769	0
76	46830	0.094063	3	78330	0.186082	0
77	46530	0.093237	4	51930	0.18723	1
78	45630	0.093452	4	49230	0.181939	0
79	53130	0.09152	1	55230	0.183183	0
80	47430	0.095143	2	44130	0.186502	1
81	42630	0.099058	4	43230	0.192509	2
82	50430	0.098353	4	54030	0.186336	1
83	48330	0.090852	1	55830	0.18375	0
84	42630	0.094848	3	51030	0.186581	1
85	48030	0.095974	2	42330	0.187451	1
86	49830	0.090712	1	73530	0.185839	0
87	46530	0.092524	4	55530	0.185343	1
88	46530	0.096294	1	57930	0.183871	1
89	48330	0.097982	2	59130	0.191731	1
90	44130	0.096837	3	42030	0.189222	0
91	54030	0.092407	1	64230	0.183632	0
92	53730	0.096649	2	62730	0.184454	0
93	50730	0.096982	3	46230	0.194692	2
94	45330	0.096083	3	42930	0.19215	1
95	43530	0.092953	2	59730	0.186606	1
96	45330	0.09591	4	41430	0.190864	1
97	49830	0.093226	2	51630	0.18574	0
98	42630	0.095905	3	45930	0.185915	1
99	43230	0.092499	1	51030	0.18452	1
100	60330	0.092608	1	51930	0.184875	0

Mean	46947	0.094911	2.51	51081	0.186989	0.74
Stdev	4663.572	0.002343	1.159023	7886.704	0.003085	0.773553

Table O.3: Results of POTFT with VM Quiescing (continued)

Alg.	**POTF_27_Q_40**			**POTF_8_Q_80**		
Sim. ID	Time until migration	OTF	No. of quiescings	Time until migration	OTF	No. of quiescings
1	54030	0.280363	1	80430	0.082433	9
2	86430	0.219021	0	83430	0.089397	7
3	49230	0.274974	0	82230	0.083568	7
4	41430	0.275118	0	86430	0.086733	5
5	86430	0.21555	0	86430	0.080761	4
6	45630	0.276043	0	80730	0.084191	7
7	86430	0.25026	0	81330	0.081442	4
8	86430	0.21555	0	86430	0.08775	6
9	79530	0.275458	0	86430	0.083704	3
10	86430	0.253731	0	82230	0.081917	9
11	72630	0.275068	0	86430	0.085013	5
12	86430	0.212079	0	84630	0.085083	9
13	44730	0.275508	0	83130	0.084914	10
14	51330	0.282468	1	85530	0.083823	9
15	49230	0.275311	0	83730	0.086784	9
16	48930	0.282801	1	84630	0.082992	5
17	61230	0.27548	0	86430	0.08224	2
18	86430	0.25026	0	85830	0.084161	10
19	55230	0.282898	1	80730	0.085885	7
20	86430	0.222492	0	86430	0.082595	9
21	46830	0.275565	0	86430	0.087649	4
22	47130	0.275777	0	86430	0.088156	4
23	86430	0.208608	0	86430	0.08692	7
24	86430	0.229434	0	80730	0.083841	5
25	86430	0.229434	0	80130	0.083001	5
26	86430	0.229434	0	86430	0.086624	5
27	55230	0.276204	0	80430	0.08273	2
28	86430	0.225963	0	84330	0.082189	7
29	50730	0.282174	1	86430	0.088406	5
30	61830	0.279535	1	86430	0.085926	7
31	86430	0.257202	0	85530	0.088233	7
32	48330	0.275494	0	86430	0.088122	7
33	45630	0.282516	1	86430	0.083001	9
34	86430	0.239847	0	81330	0.083779	7
35	47130	0.275563	0	86430	0.083724	6

36	43530	0.275856	0	86430	0.082807	10
37	86430	0.205137	0	86430	0.083967	5
38	47730	0.277741	1	86430	0.086777	7
39	65130	0.282491	1	81330	0.082594	7
40	86430	0.205137	0	86430	0.083444	3
41	64530	0.277137	1	86430	0.08725	6
42	86430	0.232905	0	81930	0.087263	3
43	86430	0.225963	0	80730	0.08157	7
44	86430	0.236376	0	82530	0.0813	5
45	86430	0.25026	0	86430	0.080892	3
46	86430	0.25026	0	86430	0.083001	9
47	86430	0.222492	0	86430	0.085562	8
48	41130	0.283051	1	85830	0.081917	10
49	86430	0.212079	0	86430	0.084278	3
50	59130	0.274811	0	86430	0.085362	3
51	86430	0.201666	0	86430	0.081587	3
52	86430	0.229434	0	86430	0.083096	3
53	86430	0.243318	0	86430	0.084454	6
54	51030	0.275382	0	86430	0.083211	9
55	54930	0.275097	0	83130	0.082179	5
56	86430	0.212079	0	86430	0.08455	3
57	78330	0.275943	0	83730	0.080427	2
58	86430	0.205137	0	86430	0.081569	4
59	86430	0.232905	0	85230	0.089566	10
60	86430	0.239847	0	85830	0.083635	10
61	86430	0.205137	0	86430	0.086796	3
62	86430	0.201666	0	86430	0.083212	3
63	53430	0.277443	1	84630	0.086567	9
64	86430	0.25026	0	86430	0.083937	7
65	51930	0.27976	1	86430	0.085216	6
66	77430	0.274982	0	86430	0.083936	6
67	86430	0.225963	0	86430	0.08487	9
68	52830	0.275421	0	86430	0.082376	4
69	86430	0.232905	0	86430	0.087682	3
70	86430	0.219021	0	86430	0.083951	3
71	79530	0.274798	0	85530	0.08487	9
72	82830	0.275779	0	85230	0.085337	8
73	86430	0.253731	0	86430	0.085611	9
74	66930	0.275567	0	86430	0.086286	6
75	54630	0.276251	0	86430	0.080847	5
76	86430	0.201666	0	86430	0.084564	7
77	86430	0.264144	0	86430	0.083631	9

78	86430	0.212079	0	83130	0.08455	9
79	76230	0.275247	0	82230	0.081679	2
80	65130	0.278235	1	86430	0.086446	5
81	46230	0.277903	1	80430	0.081727	9
82	86430	0.25026	0	86430	0.087172	8
83	86430	0.222492	0	86430	0.081931	3
84	65130	0.277107	1	83430	0.083622	7
85	56430	0.279551	1	86430	0.086446	5
86	86430	0.208608	0	80430	0.081209	2
87	45330	0.275174	0	86430	0.082896	9
88	86430	0.243318	0	86430	0.084118	3
89	86430	0.212079	0	86430	0.085756	5
90	86430	0.205137	0	84330	0.088233	7
91	86430	0.205137	0	86430	0.083147	2
92	86430	0.225963	0	85530	0.088802	4
93	43530	0.280913	1	84030	0.083514	6
94	54630	0.277375	1	86430	0.084352	7
95	86430	0.246789	0	80130	0.083929	5
96	64530	0.276177	0	83130	0.085395	9
97	86430	0.264144	0	86430	0.082225	5
98	60930	0.275934	0	81330	0.085778	7
99	46530	0.276081	0	84030	0.082908	3
100	77730	0.275499	0	86430	0.084062	2
Mean	71913	0.252014	0.19	84954	0.084395	5.97
Stdev	16958.02	0.028165	0.394277	2111.303	0.002183	2.438972

Table O.3: Results of POTFT with VM Quiescing (continued)

Alg.	**POTF_16_Q_80**			**POTF_24_Q_80**		
Sim. ID	Time until migration	OTF	No. of quiescings	Time until migration	OTF	No. of quiescings
1	80430	0.174039	6	86430	0.245319	3
2	84030	0.16536	1	86430	0.219021	0
3	86430	0.171016	5	86430	0.25106	2
4	80130	0.169348	2	86130	0.249958	2
5	80130	0.164785	1	86430	0.21555	0
6	86430	0.167479	2	86430	0.24865	2
7	86430	0.164588	2	86430	0.244917	1
8	86430	0.161019	1	86430	0.21555	0
9	80730	0.166547	1	86430	0.247748	1
10	86130	0.166516	1	86430	0.245882	1
11	84930	0.163848	1	86430	0.250116	1
12	86430	0.168676	5	86430	0.212079	0

13	86430	0.168513	2	84030	0.251355	2
14	86430	0.163618	5	86130	0.248363	3
15	86430	0.169693	2	86430	0.248641	2
16	84930	0.168452	3	86430	0.245795	4
17	86430	0.166981	2	86430	0.250566	2
18	86430	0.167423	2	86430	0.2486	1
19	85530	0.167355	6	86430	0.245643	3
20	86430	0.169789	2	86430	0.222492	0
21	86430	0.167507	2	84330	0.250397	2
22	86430	0.16241	2	86430	0.245298	2
23	86430	0.170192	3	86430	0.208608	0
24	84330	0.167459	2	86430	0.229434	0
25	83430	0.163237	1	86430	0.229434	0
26	86430	0.166199	2	86430	0.229434	0
27	86430	0.164978	2	86430	0.245674	2
28	86430	0.173563	3	86430	0.225963	0
29	84030	0.166334	3	86430	0.251047	3
30	86430	0.16672	3	86430	0.250782	3
31	86430	0.163307	2	86430	0.24843	1
32	86430	0.168943	3	86430	0.251197	2
33	86430	0.167547	6	86430	0.251164	4
34	80730	0.163093	1	86430	0.239847	0
35	86430	0.161794	2	86430	0.246567	2
36	85530	0.170508	6	86430	0.250512	2
37	86430	0.161325	1	86430	0.205137	0
38	83430	0.171875	4	86430	0.248469	4
39	83430	0.17104	4	86430	0.247155	3
40	86430	0.167456	2	86430	0.205137	0
41	86430	0.165744	3	86430	0.247716	3
42	86430	0.16718	2	86430	0.232905	0
43	85830	0.163706	2	86430	0.225963	0
44	86430	0.169646	2	86430	0.236376	0
45	86430	0.16365	2	86430	0.245344	1
46	85530	0.165872	5	86430	0.249831	1
47	86430	0.163354	2	86430	0.222492	0
48	86430	0.174124	6	86430	0.24559	4
49	86430	0.167803	1	86430	0.212079	0
50	86430	0.162488	2	86430	0.24965	2
51	86430	0.164177	1	86430	0.201666	0
52	86430	0.163964	1	86430	0.229434	0
53	86430	0.165884	3	86430	0.250708	1
54	86430	0.168739	2	86430	0.246889	2

55	85830	0.16229	2	86430	0.247502	2
56	86430	0.162577	1	86430	0.212079	0
57	86430	0.165873	1	86430	0.24794	1
58	86430	0.164161	1	86430	0.205137	0
59	86430	0.168445	6	86430	0.232905	0
60	86430	0.167041	2	86430	0.239847	0
61	86430	0.164636	2	86430	0.205137	0
62	86430	0.167004	1	86430	0.201666	0
63	83730	0.172195	6	81030	0.250714	3
64	85830	0.164688	3	86430	0.250629	1
65	86430	0.173686	4	81330	0.250882	3
66	86430	0.164668	2	86430	0.250724	1
67	81630	0.167306	5	86430	0.225963	0
68	86430	0.16484	2	86430	0.245163	2
69	86430	0.168764	2	86430	0.232905	0
70	84630	0.166581	1	86430	0.219021	0
71	86430	0.166828	1	86430	0.245872	1
72	86430	0.165932	2	86430	0.248702	1
73	86430	0.167832	6	86430	0.249309	1
74	86430	0.164716	2	86430	0.245748	2
75	86430	0.168433	2	86430	0.249806	2
76	86430	0.162993	1	86430	0.201666	0
77	86430	0.16664	2	86430	0.245168	1
78	86430	0.173109	2	86430	0.212079	0
79	86130	0.164177	1	83430	0.251689	1
80	86430	0.162587	3	86430	0.248482	3
81	86430	0.17434	6	80730	0.246519	3
82	84030	0.17039	2	86430	0.249834	1
83	83130	0.165356	1	86430	0.222492	0
84	82530	0.166416	2	86430	0.245683	3
85	84930	0.165718	3	86430	0.245219	3
86	86430	0.164261	1	86430	0.208608	0
87	86430	0.168167	2	85530	0.250466	2
88	85230	0.175635	2	86430	0.245301	1
89	84630	0.169434	2	86430	0.212079	0
90	85230	0.167019	2	86430	0.205137	0
91	86430	0.163648	1	86430	0.205137	0
92	86430	0.162939	1	86430	0.225963	0
93	85530	0.170762	5	80130	0.2467	4
94	86430	0.17292	3	86430	0.247854	3
95	85530	0.163213	2	86430	0.249002	1
96	86430	0.164744	4	86430	0.246757	2

97	80130	0.167245	1	86430	0.250575	1
98	86430	0.166314	3	86430	0.249169	2
99	83430	0.162696	2	86430	0.245634	2
100	80730	0.17453	1	86430	0.246338	1
Mean	85476	0.166906	2.49	86115	0.237248	1.28
Stdev	1732.25	0.003352	1.520865	1178.886	0.01613	1.239746

Table O.3: Results of POTFT with VM Quiescing (continued)

Alg.	**POTF_8.5_Q_80**			**POTF_17_Q_80**		
Sim. ID	Time until migration	OTF	No. of quiescings	Time until migration	OTF	No. of quiescings
1	85830	0.092272	9	82530	0.185747	6
2	86430	0.092672	7	86430	0.180359	1
3	84630	0.095471	7	86430	0.179905	5
4	81930	0.086521	4	86430	0.181399	3
5	81930	0.087017	3	86130	0.177215	1
6	85530	0.093941	7	86430	0.180042	2
7	83730	0.087415	4	86430	0.176567	2
8	81930	0.090057	5	86430	0.175784	1
9	86430	0.089178	3	86430	0.179502	1
10	84930	0.093073	9	86430	0.174137	1
11	83430	0.089028	4	86430	0.17578	1
12	86430	0.086987	9	86430	0.178725	5
13	80730	0.095594	9	86430	0.180503	2
14	84930	0.089311	8	86430	0.179869	5
15	82530	0.087854	8	86430	0.179122	2
16	86430	0.09556	5	81630	0.177912	3
17	86430	0.087062	2	86430	0.177082	1
18	84030	0.095492	9	86430	0.17677	2
19	86430	0.095672	7	86430	0.178755	6
20	84930	0.089417	8	86430	0.180042	2
21	83430	0.089292	3	86430	0.174302	2
22	86430	0.089029	4	86430	0.17643	2
23	86430	0.088684	7	86430	0.178815	3
24	83730	0.089073	5	84030	0.177669	2
25	84330	0.089952	5	86430	0.173271	1
26	81330	0.088559	4	86430	0.177596	2
27	86430	0.087269	2	86430	0.17767	2
28	84630	0.094285	7	86430	0.181682	3
29	86430	0.090943	5	84630	0.17934	3
30	80730	0.088433	6	86430	0.181195	3
31	86430	0.089314	7	84630	0.175823	1

32	86430	0.090099	7	86430	0.178746	3
33	84030	0.088333	8	86430	0.178844	6
34	85530	0.09484	7	86430	0.175032	1
35	84330	0.088183	5	86430	0.177448	2
36	83430	0.094782	9	86430	0.181358	6
37	81330	0.09012	4	86430	0.173317	1
38	86430	0.087012	7	83130	0.184107	4
39	86430	0.092475	7	83430	0.18078	4
40	86430	0.089675	3	86430	0.178551	2
41	82830	0.089013	5	86430	0.177299	3
42	85230	0.088685	3	83430	0.176675	1
43	85530	0.093648	7	83430	0.175649	2
44	85230	0.086808	5	86430	0.178435	2
45	86430	0.086387	3	86430	0.176239	2
46	83430	0.089066	8	86430	0.179659	5
47	85530	0.093492	7	86430	0.172751	2
48	85830	0.09093	9	86430	0.184493	6
49	86430	0.089675	3	86430	0.177178	1
50	86430	0.090032	3	86430	0.176235	2
51	86430	0.088187	3	86430	0.175272	1
52	86430	0.089479	3	86430	0.176298	1
53	83730	0.087321	5	86430	0.179224	3
54	84930	0.089311	8	86430	0.178088	2
55	85830	0.08809	5	84030	0.174353	2
56	86430	0.090271	3	86430	0.17363	1
57	86430	0.085349	2	86430	0.178731	1
58	86430	0.086678	4	86430	0.174834	1
59	86130	0.098712	10	84330	0.179297	5
60	84030	0.094068	9	86430	0.177461	2
61	86430	0.089771	3	86430	0.174505	1
62	84030	0.089325	2	86430	0.174329	1
63	81930	0.086681	8	86430	0.180506	6
64	82530	0.090411	6	84330	0.175894	3
65	82830	0.088697	5	86430	0.184861	4
66	83730	0.089882	5	86430	0.177481	2
67	83430	0.087854	8	86430	0.176207	5
68	86430	0.086473	4	86430	0.179665	2
69	86430	0.088641	3	81630	0.17799	1
70	82530	0.089183	2	86430	0.180818	1
71	82530	0.086272	8	86430	0.176391	1
72	82530	0.096677	7	82830	0.175593	1
73	83430	0.086068	8	82830	0.178046	5

74	81930	0.08954	5	82230	0.174727	1
75	86430	0.093804	5	86430	0.1813	2
76	83430	0.090518	6	86430	0.175971	1
77	84930	0.088083	8	84930	0.178762	2
78	83430	0.086272	8	81330	0.175868	1
79	86430	0.088317	2	86430	0.173957	1
80	86430	0.088095	5	86430	0.175351	3
81	85830	0.091361	9	83130	0.183131	5
82	84330	0.095016	7	86430	0.178978	2
83	86430	0.08856	3	86430	0.172296	1
84	84630	0.094174	7	81930	0.177046	2
85	86430	0.088932	5	83730	0.179551	3
86	86430	0.087132	2	86430	0.17593	1
87	83430	0.089066	8	86430	0.17812	2
88	86430	0.087422	3	86430	0.173292	2
89	81330	0.087727	4	86430	0.180442	2
90	86430	0.089568	7	86430	0.178551	2
91	86430	0.088922	2	86430	0.17732	1
92	86430	0.088721	4	86430	0.176136	1
93	86430	0.094891	6	86430	0.183591	5
94	82530	0.089633	6	86430	0.184523	3
95	86430	0.087467	5	86430	0.180246	2
96	81930	0.088409	8	80430	0.176589	3
97	84030	0.088765	4	86430	0.177371	1
98	84630	0.094615	7	86430	0.175835	3
99	85530	0.088951	3	84030	0.173741	2
100	86430	0.088787	2	86430	0.172202	1
Mean	84783	0.089938	5.54	85674	0.177821	2.38
Stdev	1727.055	0.002871	2.244724	1493.915	0.002897	1.502725

Table O.3: Results of POTFT with VM Quiescing (continued)

Alg.	**POTF_25.5_Q_80**			**POTF_9_Q_80**		
Sim. ID	Time until migration	OTF	No. of quiescings	Time until migration	OTF	No. of quiescings
1	82830	0.264087	2	85830	0.096091	8
2	86430	0.219021	0	83430	0.095676	6
3	86430	0.267712	1	86430	0.097108	7
4	86430	0.261661	2	86430	0.097042	4
5	86430	0.21555	0	84630	0.091809	3
6	86430	0.262954	2	86430	0.096149	7
7	86430	0.25026	0	86430	0.090189	4
8	86430	0.21555	0	86430	0.095832	5

9	86430	0.268703	1	82230	0.094891	2
10	86430	0.253731	0	86430	0.099197	8
11	86430	0.261314	1	86430	0.093157	4
12	86430	0.212079	0	82530	0.095192	7
13	86430	0.267823	2	86430	0.097495	9
14	86430	0.264225	3	83130	0.093689	7
15	86430	0.263066	2	86430	0.097331	8
16	86430	0.262325	3	86430	0.098308	5
17	86430	0.263781	1	86430	0.092092	2
18	86430	0.25026	0	81630	0.094918	8
19	86430	0.262381	2	86430	0.097108	7
20	86430	0.222492	0	81630	0.0939	7
21	86430	0.264057	2	86430	0.097189	3
22	86430	0.265516	2	86430	0.098643	4
23	86430	0.208608	0	86430	0.098606	6
24	86430	0.229434	0	86430	0.092046	5
25	86430	0.229434	0	86430	0.092151	5
26	86430	0.229434	0	86430	0.095123	4
27	86430	0.267193	1	86430	0.090828	2
28	86430	0.225963	0	82230	0.096873	6
29	81930	0.268473	2	84330	0.097615	4
30	80430	0.263998	2	86430	0.097396	6
31	86430	0.266083	1	86430	0.096423	6
32	86430	0.262975	2	86130	0.09835	6
33	86430	0.265383	3	83130	0.092426	7
34	86430	0.239847	0	86430	0.095349	7
35	84930	0.264937	1	86430	0.092909	5
36	86430	0.261281	2	80130	0.096044	8
37	86430	0.205137	0	86430	0.095175	4
38	86430	0.267075	3	86130	0.098149	6
39	84330	0.263881	2	81330	0.096039	6
40	86430	0.205137	0	86430	0.095745	3
41	86430	0.265102	2	86430	0.097359	5
42	86430	0.232905	0	86430	0.096467	3
43	86430	0.225963	0	81330	0.096039	6
44	86430	0.236376	0	86430	0.090596	5
45	86430	0.25026	0	81930	0.09037	2
46	86430	0.25026	0	82530	0.093182	7
47	86430	0.222492	0	86430	0.097108	7
48	86430	0.265567	3	85830	0.096349	8
49	86430	0.212079	0	86430	0.094567	3
50	86430	0.267848	1	82830	0.095173	2

51	86430	0.201666	0	82830	0.091152	2
52	86430	0.229434	0	86430	0.095063	3
53	86430	0.243318	0	86430	0.096202	5
54	86430	0.263777	1	82530	0.094328	7
55	85230	0.262653	1	86430	0.094029	5
56	86430	0.212079	0	86430	0.092983	3
57	86430	0.263958	1	86430	0.090189	2
58	86430	0.205137	0	83430	0.091809	3
59	86430	0.232905	0	85530	0.099938	9
60	86430	0.239847	0	81030	0.095864	8
61	86430	0.205137	0	85230	0.095696	2
62	86430	0.201666	0	86430	0.092268	2
63	86430	0.26289	3	86430	0.095168	8
64	86430	0.25026	0	86430	0.09601	6
65	86430	0.266487	3	86430	0.097155	5
66	86430	0.262508	1	86430	0.096946	5
67	86430	0.225963	0	86430	0.09562	8
68	86430	0.26316	1	81930	0.091809	3
69	86430	0.232905	0	86430	0.095052	3
70	86430	0.219021	0	86430	0.092653	2
71	86430	0.263931	1	86430	0.096032	8
72	86430	0.268386	1	86430	0.098227	7
73	86430	0.253731	0	86430	0.096897	8
74	86430	0.263828	1	86430	0.096267	5
75	86430	0.268145	1	81930	0.096912	4
76	86430	0.201666	0	86430	0.094305	6
77	86430	0.263081	1	83130	0.092556	7
78	86430	0.212079	0	86430	0.094628	8
79	86430	0.262364	1	86430	0.09176	2
80	86130	0.262804	2	85530	0.096338	4
81	86430	0.263347	3	83430	0.099496	8
82	86430	0.25026	0	86430	0.098967	7
83	86430	0.222492	0	82230	0.091091	2
84	86430	0.264508	2	80430	0.095091	6
85	83730	0.26402	2	85530	0.096219	4
86	86430	0.208608	0	86430	0.090951	2
87	86430	0.264652	2	81630	0.093689	7
88	86430	0.243318	0	86430	0.096539	3
89	86430	0.212079	0	86430	0.098229	4
90	86430	0.205137	0	86130	0.09805	6
91	86430	0.205137	0	86430	0.093571	2
92	86430	0.225963	0	86430	0.097861	4

93	86430	0.264683	3	83430	0.097228	5
94	81330	0.261818	2	86430	0.096328	6
95	86430	0.246789	0	86430	0.093194	5
96	86430	0.263078	1	86430	0.096154	8
97	86430	0.265091	1	86430	0.093468	4
98	86430	0.264434	1	80430	0.096149	6
99	86430	0.261788	2	86430	0.09274	3
100	86430	0.266057	1	86430	0.093775	2
Mean	86160	0.245797	0.9	85179	0.095201	5.13
Stdev	1022.03	0.022685	1.020002	1938.16	0.002371	2.077562

Table O.3: Results of POTFT with VM Quiescing (continued)

Alg.	**POTF_18_Q_80**			**POTF_27_Q_80**		
Sim. ID	Time until migration	OTF	No. of quiescings	Time until migration	OTF	No. of quiescings
1	86430	0.194367	6	86430	0.28095	2
2	86430	0.185497	1	86430	0.219021	0
3	86430	0.19256	4	86430	0.277505	2
4	81630	0.190153	2	86430	0.280371	2
5	83430	0.183606	1	86430	0.21555	0
6	86430	0.187997	2	86430	0.284043	2
7	86430	0.18751	2	86430	0.25026	0
8	86430	0.184334	1	86430	0.21555	0
9	81930	0.187503	1	86430	0.277993	1
10	86430	0.187212	1	86430	0.253731	0
11	86430	0.183168	1	86430	0.28032	1
12	86430	0.191835	4	86430	0.212079	0
13	86430	0.18899	2	86430	0.278043	2
14	86430	0.186074	4	86430	0.280338	2
15	86430	0.189543	2	86430	0.277845	2
16	86430	0.191287	3	86430	0.283392	2
17	83730	0.187536	1	86430	0.283463	1
18	86430	0.183472	2	86430	0.25026	0
19	85830	0.191791	5	86430	0.27804	2
20	86430	0.189027	2	86430	0.222492	0
21	86430	0.181377	2	86430	0.280826	2
22	86430	0.187283	2	86430	0.283769	2
23	81630	0.186223	2	86430	0.208608	0
24	86430	0.191339	2	86430	0.229434	0
25	86430	0.185276	1	86430	0.229434	0
26	86430	0.185979	2	86430	0.229434	0
27	86430	0.187481	2	86430	0.284209	2

28	86430	0.190467	3	86430	0.225963	0
29	86430	0.190354	3	86430	0.282764	2
30	86430	0.186173	3	83130	0.277402	2
31	86430	0.187294	2	86430	0.257202	0
32	82530	0.188083	2	86430	0.280753	2
33	86430	0.187632	5	86430	0.277665	2
34	84930	0.18635	1	86430	0.239847	0
35	86430	0.182784	2	86430	0.278099	2
36	84030	0.19284	5	86430	0.28385	2
37	86430	0.186467	1	86430	0.205137	0
38	86430	0.193831	4	86430	0.283778	2
39	86430	0.192537	4	85830	0.283082	2
40	86430	0.185632	2	86430	0.205137	0
41	81630	0.186143	2	86430	0.277718	2
42	86430	0.185507	2	86430	0.232905	0
43	86430	0.187176	2	86430	0.225963	0
44	86430	0.187166	2	86430	0.236376	0
45	86430	0.185156	2	86430	0.25026	0
46	86430	0.186074	4	86430	0.25026	0
47	86430	0.183526	2	86430	0.222492	0
48	86430	0.193572	5	86430	0.283643	2
49	86430	0.184423	1	86430	0.212079	0
50	86430	0.184055	2	86430	0.280058	1
51	86430	0.184064	1	86430	0.201666	0
52	86430	0.181312	1	86430	0.229434	0
53	86430	0.186537	3	86430	0.243318	0
54	86430	0.187769	2	86430	0.277916	2
55	86430	0.18452	2	85830	0.28307	1
56	86430	0.182476	1	86430	0.212079	0
57	86430	0.186889	1	86430	0.281211	1
58	86430	0.182449	1	86430	0.205137	0
59	86430	0.18477	5	86430	0.232905	0
60	86430	0.190151	2	86430	0.239847	0
61	82530	0.182148	1	86430	0.205137	0
62	86430	0.183447	1	86430	0.201666	0
63	84930	0.187403	5	86430	0.280749	2
64	86430	0.186851	3	86430	0.25026	0
65	86430	0.190914	3	86430	0.277625	2
66	86430	0.182316	2	86430	0.282951	1
67	81330	0.185857	4	86430	0.225963	0
68	86430	0.186806	2	82530	0.283404	1
69	86430	0.183947	2	86430	0.232905	0

70	86430	0.190098	1	86430	0.219021	0
71	86430	0.188238	1	86430	0.282762	1
72	86430	0.187831	2	82830	0.275779	0
73	86430	0.18641	5	86430	0.253731	0
74	86430	0.18444	2	86430	0.283553	1
75	86430	0.189603	2	81330	0.281525	1
76	86430	0.186796	1	86430	0.201666	0
77	86430	0.185378	2	86430	0.264144	0
78	86430	0.182641	2	86430	0.212079	0
79	86430	0.183888	1	86430	0.283224	1
80	86430	0.184657	3	86430	0.278818	2
81	86430	0.192509	5	86430	0.283944	2
82	86430	0.186336	2	86430	0.25026	0
83	86130	0.182633	1	86430	0.222492	0
84	86430	0.186581	2	85830	0.280409	2
85	86430	0.187451	3	86430	0.280136	2
86	86430	0.184707	1	86430	0.208608	0
87	86430	0.183509	2	86430	0.280428	2
88	86430	0.183871	2	86430	0.243318	0
89	86430	0.191731	2	86430	0.212079	0
90	86430	0.189945	2	86430	0.205137	0
91	86430	0.184339	1	86430	0.205137	0
92	86430	0.185164	1	86430	0.225963	0
93	86130	0.192766	4	86430	0.284233	2
94	86430	0.190249	3	86430	0.277956	2
95	86430	0.18476	2	86430	0.246789	0
96	86430	0.190864	3	84030	0.281449	1
97	82830	0.186453	1	86430	0.264144	0
98	86430	0.185915	3	86430	0.283931	1
99	86430	0.186363	2	86430	0.281352	2
100	83130	0.185586	1	86430	0.283484	1
Mean	85920	0.18694	2.28	86229	0.253842	0.81
Stdev	1313.219	0.003102	1.239746	827.9822	0.029826	0.906709

We calculated the mean and standerd deviation for 100 samples of results of each algoiritm with specific parameter. Then, the 95% confidence interval of the results is calculated for each algorithm using mean and standerd deviation as shown in Table O.1 and Figures 4.2 to 4.7.

Appendix P: Architecture of CloudSim

The layered CloudSim architecture is shown in Figure P.1. The CloudSim simulation supports modeling and simulation of virtualized cloud data centers with dedicated management for PMs, VMs, bandwidth, and memory. The User Code layer defines the specifications and requirements for PMs, VMs, applications, users, and broker scheduling policies. The other CloudSim layers are user interface structures (cloudlet, VM), VM services (cloudlet execution, VM management), cloud services (VM provisioning, CPU allocation, RAM allocation, memory allocation, bandwidth allocation, and workload traces), cloud resources (PM, and data center) and network topology.

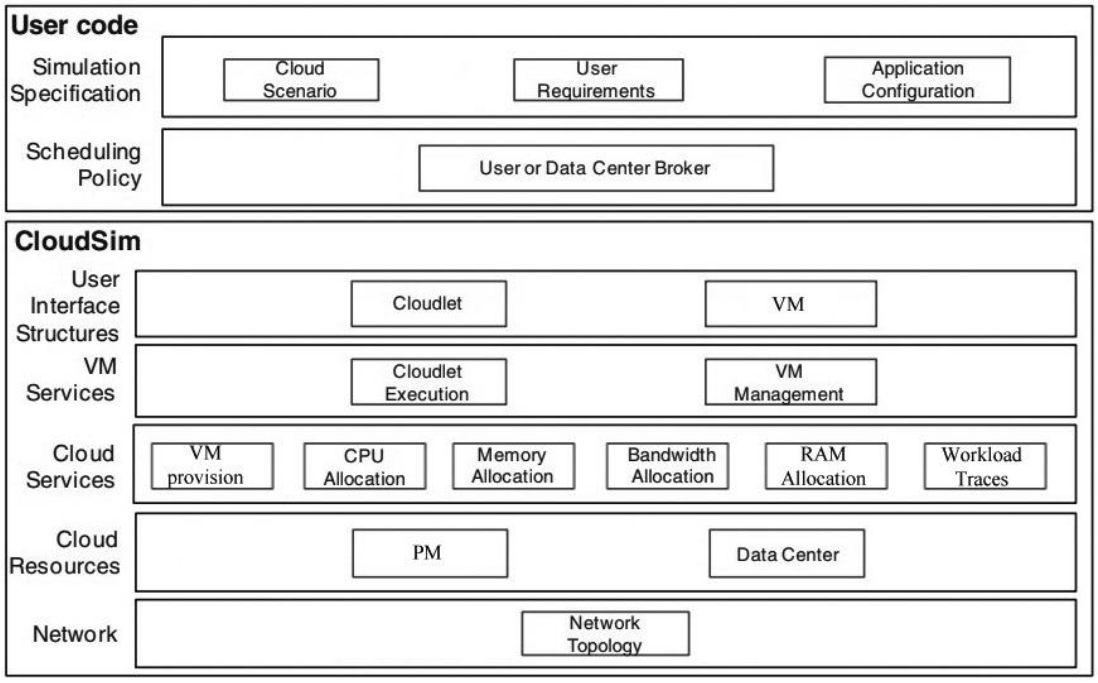

Figure P.1: Layered CloudSim architecture

Appendix Q: The Simulation Model Structure in Clojure

The simulation model structure build using Clojure is shown in Figure Q.1. The simulation model provides support for simulation of PM overloading detection and VM quiescing including management for PM, and VMs. The simulation model includes services (CPU utilization allocation, PM overload detection, VM quiescing, and workload traces), and resources (PM, VMs).

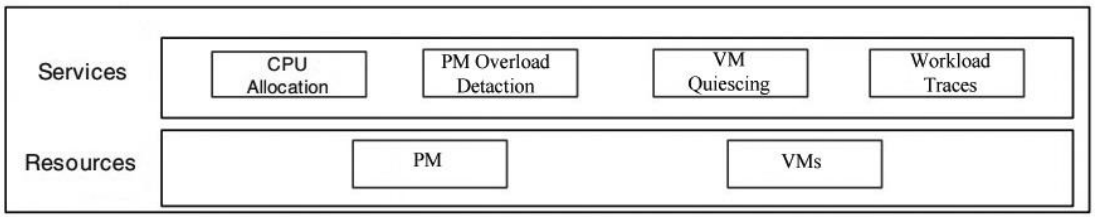

Figure Q.1: The simulation model structures build using Clojure

Appendix R: Example of the CPU utilization computation in Chapter 3

Suppose that:

- Total MIPS rating of PM is 5320.00.
- VM1, VM2, VM3, and VM4 are allocated on PM1 at time 10.
- VM5 are currently being migrated in PM1 at time 10.
- Utilization in MIPS of VM1, VM2, VM3, and VM4 are 144.00, 171.00, 233.95, and 1224.95, respectively.
- Utilization in MIPS, memory size (Byte), and network bandwidth (Byte/Second) of VM5 are 175.00, 600000, and 100000, respectively.

CPU utilization of PM1 at time 10 in MIPS using equation (3.3) equals:

$U_{MIPS_1}(10)$= 144+171+233.95+1224.95+(0.1×175×600000/100000) =1878.9

CPU utilization of PM1 at time 10 using equation (3.4) equals:

$U_1(10)$=1878.9/5320=35.3%

Appendix S: Example of the CPU utilization computation in Chapter 4

Suppose that:

- Total frequency rating of PM is 12000000 Hz
- VM1, VM2, VM3, and VM4 with utilization (5%, 8%, 10%, 15%) are allocated in PM at time 10.
- The frequency of VM1, VM2, VM3, and VM4 are 1700000 Hz, 2000000 Hz, 2400000 Hz, and 3000000 Hz, respectively.

CPU utilization of PM at time 10 using equation (4.2) equals:

$$U(10)=\frac{0.05\times1700000+0.8\times2000000+0.1\times2400000+0.12\times3000000}{12000000}=25.4\%$$