

OPTIMAL DYNAMIC RESOURCE ALLOCATION FOR HETEROGENOUS
CLOUD DATA CENTERS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

NAZIM UMUT EKİCİ

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

SEPTEMBER 2019

Approval of the thesis:

**OPTIMAL DYNAMIC RESOURCE ALLOCATION FOR HETEROGENOUS
CLOUD DATA CENTERS**

submitted by **NAZIM UMUT EKICI** in partial fulfillment of the requirements for
the degree of **Master of Science in Electrical and Electronics Engineering De-
partment, Middle East Technical University** by,

Prof. Dr. Halil Kalipçılar
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. İlkey Ulusoy
Head of Department, **Electrical and Electronics Engineering** _____

Prof. Dr. Şenan Ece Güran Schmidt
Supervisor, **Electrical and Electronics Engineering, METU** _____

Prof. Dr. Klaus Werner Schmidt
Co-supervisor, **Electrical and Electronics Engineering, METU** _____

Examining Committee Members:

Assoc. Prof. Dr. Cüneyt F. Bazlamaçcı
Electrical and Electronics Engineering Department, METU _____

Prof. Dr. Şenan Ece Güran Schmidt
Electrical and Electronics Engineering Department, METU _____

Prof. Dr. Gözde Bozdağı Akar
Electrical and Electronics Engineering Department, METU _____

Prof. Dr. Özcan Öztürk
Computer Engineering Department, Bilkent University _____

Assist. Prof. Dr. Pelin Angın
Computer Engineering Department, METU _____

Date:



I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname: Nazım Umut Ekici

Signature :

ABSTRACT

OPTIMAL DYNAMIC RESOURCE ALLOCATION FOR HETEROGENOUS CLOUD DATA CENTERS

Ekici, Nazım Umut

M.S., Department of Electrical and Electronics Engineering

Supervisor: Prof. Dr. Şenan Ece Güran Schmidt

Co-Supervisor: Prof. Dr. Klaus Werner Schmidt

September 2019, 69 pages

Today's data centers are mostly cloud-based with virtualized servers to provide on-demand scalability and flexibility of the available resources such as CPU, memory, data storage and network bandwidth. Heterogeneous cloud data centers (CDCs) offer hardware accelerators in addition to these standard cloud server resources. A cloud data center provider may provide Infrastructure as a Service and Platform as a Service (IPaaS), where the user gets a virtual machine (VM) with processing, memory, storage and networking resources, which can be installed with any desired operating system and software. Differently, Software as a Service (SaaS), only enables user access to provided application for example via a web browser without any control of the underlying infrastructure.

In this context, it is important to note that the data processing for SaaS can be executed on different physical resources such as a server as well as a hardware accelerator with different performance and power consumption. To this end, a very significant feature of heterogeneous CDCs is that they offer the flexibility of meeting user demands for SaaS by choosing among the available physical resource alternatives. To utilize this

flexibility, a CDC resource manager must decide which resource alternative will be chosen, along with the decision of the physical resource the request will be assigned to.

In this thesis we propose ACLOUD-MAN (ACCelerated CLOUD MANager), a novel resource manager for heterogeneous CDCs. ACLOUD-MAN's resource management objective is to reduce the power consumption of the CDC in order to support green computing. To this end, the resource allocation problem is modeled as an integer linear programming problem and is implemented in MATLAB, along with a cloud data center simulation platform. We evaluate the performance of ACLOUD-MAN under different realistic cloud workloads. Simulation results show that the proposed ACLOUD-MAN outperforms existing resource allocation methods such as OpenStack.

Keywords: Cloud computing, Heterogeneous data center, Hardware accelerator, Resource management, Green computing

ÖZ

HETEROJEN BULUT VERİ MERKEZLERİ İÇİN OPTİMAL DİNAMİK KAYNAK ATAMA

Ekici, Nazım Umut

Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü

Tez Yöneticisi: Prof. Dr. Şenan Ece Güran Schmidt

Ortak Tez Yöneticisi: Prof. Dr. Klaus Werner Schmidt

Eylül 2019 , 69 sayfa

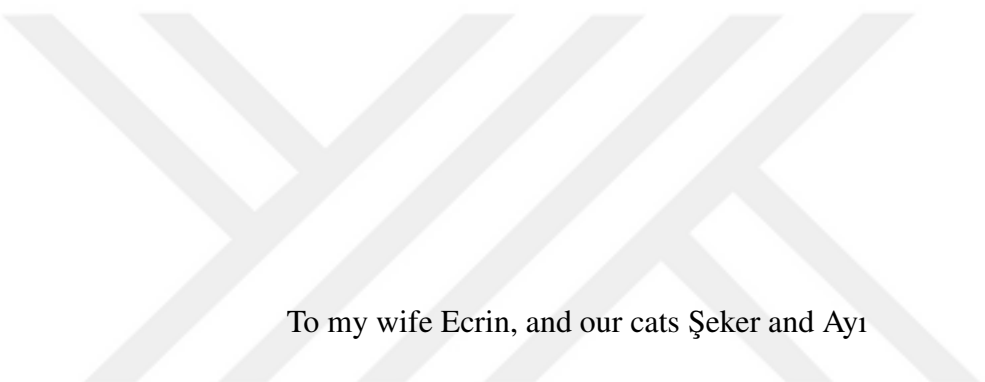
Günümüzün veri merkezleri işlemci (CPU), hafıza, veri depolama ve ağ bantgeniřliđi gibi kaynakları isteđe bađlı řekilde, ölçeklenebilir ve esnek bir řekilde sunmak için sanallařtırılmıř sunucular ile bulut tabanlı yapıdadır. Heterojen bulut biliřim merkezleri (BBM) bahsedilen kaynaklara ek olarak donanım hızlandırıcılar sunmaktadır. Bir bulut biliřim merkezi, kaynaklarını Altyapı ve Platform Olarak Servis (IPaaS) řeklinde sunabilir. Bu serviste kullanıcıya iřlem gücü, hafıza, veri depolama ve ağ kaynakları olan ve istenen iřletim sistemi ve yazılımların da yüklü olduđu bir sanal makine (VM) sunulur. Diđer servis řekli olan Yazılım Olarak Servis'te (SaaS) ise kullanıcıya altyapıya eriřimi olmaksızın bir arayüz (örn. bir web tarayıcısı arayüzü ile) aracılıđıyla sadece sađlanan uygulamaya eriřim sunulur.

Bu noktada dikkat edilmelidir ki, SaaS hizmetler için gerekli veri iřlemleri bir sunucuda yürütülebileceđi gibi bir donanım hızlandırıcıda da farklı performans ve güç tüketimi ile de yürütülebilir. Bu sayede, heterojen BBM'lerin önemli bir özelliđi SaaS kullanıcı isteklerini var olan farklı donanım alternatifleri arasından seçim yaparak bir esneklik sunmasıdır. Bu esnekliđi kullanabilmek için bir BBM kaynak yöneticisi

istekler için hangi sunucunun kullanılacağına yanı sıra hangi kaynak alternatifinin kullanılacağına da karar vermelidir.

Bu tez çalışmasında heterojen bulut veri merkezlerinde yenilikçi bir kaynak atama yöntemi olan ACCLOUD-MAN önerilmektedir. ACCLOUD-MAN'ın kaynak yönetim hedefi bulutun güç tüketimini en aza indirmektir. Bu amaçla ACCLOUD-MAN bir Tamsayı Doğrusal Problem olarak modellendi ve MATLAB üzerinde bir benzetim altyapısı ile gerçekleştirildi. ACCLOUD-MAN'ın performansı gerçekçi bulut iş yükleri ile değerlendirildi. Benzetim sonuçları önerilen ACCLOUD-MAN'ın, OpenStack gibi, var olan kaynak yönetim yöntemlerinden daha iyi sonuç verdiğini gösterdi.

Anahtar Kelimeler: Bulut bilişim, Heterojen veri merkezi, Donanım hızlandırıcı, Kaynak yönetimi, Çevreci bilişim



To my wife Ecrin, and our cats Şeker and Ayı

ACKNOWLEDGMENTS

I would like to express my greatest gratitude to my supervisors Prof. Dr. Ece Güran Schmidt and Prof. Dr. Klaus Werner Schmidt. Without their outstanding guidance and support, this work would not have a fraction of its content nor quality.

I thank ASELSAN Inc. for allowing me to attend my classes and its financial support for my conference attendance. This thesis was supported by the Scientific and Research Council of Turkey (TUBITAK) [Project Code 117E667-117E668]

TABLE OF CONTENTS

ABSTRACT	v
ÖZ	vii
ACKNOWLEDGMENTS	x
TABLE OF CONTENTS	xi
LIST OF TABLES	xiv
LIST OF FIGURES	xvi
LIST OF ALGORITHMS	xix
LIST OF ABBREVIATIONS	xx
CHAPTERS	
1 INTRODUCTION	1
2 CLOUD DATA CENTERS: BACKGROUND	5
2.1 Cloud Resources and Offered Services	5
2.2 Heterogeneous Cloud Data Centers	5
2.3 Cloud Computing Resource Management	6
3 ACCLOUD-MAN: ACCELERATED CLOUD RESOURCE MANAGER	9
3.1 Integer Linear Programming (ILP)	10
3.2 Formulation of the ILP Problem	10

3.3	Complexity Analysis	14
3.4	Comparison of ILP Solvers	14
3.5	Reducing Complexity by Limiting the Number of Servers to Be Considered	16
4	ACCLOUD-MAN EVALUATION	21
4.1	Algorithms for Comparison	21
4.1.1	OpenStack Algorithm	21
4.1.2	Modified OpenStack Algorithm	22
4.2	Simulation Environment	23
4.3	Simulation Setup	23
4.3.1	Initialization of Servers for Random Request Scenario	25
4.3.2	Generation of User Requests for Random Request Scenario	25
4.3.3	CDC State Update for Random Request Scenario	25
4.3.4	Initialization of Servers Realistic Request Scenario	26
4.3.5	Generation of User Requests Realistic Request Scenario	26
4.3.6	CDC State Update Realistic Request Scenario	29
5	EVALUATION RESULTS	33
5.1	Random Request Scenario	33
5.1.1	Experiment 1: Comparison of Resource Allocation Methods	33
5.1.2	Experiment 2: Dependency on the Number of Requests	35
5.1.3	Experiment 3: Dependency on the Size of the CDC	37
5.1.4	Experiment 4: Dependency on the Number of Candidate PMs	39
5.2	Realistic Request Scenario	42

5.2.1	Experiment 5: Comparison of Resource Allocation Methods . . .	42
5.2.2	Experiment 6: Dependency on the Number of CPU cores	49
5.2.3	Experiment 7: Dependency to the Power Parameters	53
6	FURTHER DISCUSSIONS	59
6.1	Generalization of the Model to Any Number and Type of CDC Resources	59
6.2	Generalization of the Model to an Energy-Power Hybrid Optimization	59
6.3	User Latency Implications of Request Bundling	60
6.4	Extending the Algorithm to React to Other Cloud Events	60
7	CONCLUSIONS	63
	REFERENCES	65

LIST OF TABLES

TABLES

Table 3.1	MATLAB MILP solver's computation times for various cases. Results are given in seconds.	15
Table 3.2	TOMLAB CPLEX solver's computation times for various cases. Results are given in seconds.	16
Table 3.3	TOMLAB mipSolve's computation times for various cases. Results are given in seconds.	16
Table 3.4	TOMLAB miqpBB's computation times for various cases. Results are given in seconds.	17
Table 3.5	TOMLAB minlpBB solver's computation times for various cases. Results are given in seconds.	18
Table 3.6	Brute force solver's computation times for various cases. Results are given in seconds.	18
Table 3.7	TOMLAB CPLEX solver's computation times for more cases. Results are given in seconds.	19
Table 4.1	Values of resources requested by each alternative of Software services and the average power would be consumed on the average PM. CPU is in number of cores and FPGA is in number of regions. Memory, Disc, Bandwidth and Power are in GB, TB, Gbps and W respectively.	32
Table 5.1	Comparison of average number of on PMs and average power consumption for different numbers of PMs.	40

Table 5.2 Comparison of average number of on PMs, average power consumption and average time spent for decision for different values of γ . . .	40
Table 5.3 Experiment 5 and 6: Comparison of percentage improvements of average number of on PMs and average power consumption over OpenStack for different numbers of CPU cores.	50
Table 5.4 Experiment 5 and 6: Comparison of average time for allocation decision of all algorithms in milliseconds.	50
Table 5.5 Experiment 7: Comparison of percentage improvements of average number of on PMs and average power consumption over OpenStack for different numbers of CPU cores.	54



LIST OF FIGURES

FIGURES

Figure 4.1	Software architecture of the resource manager.	24
Figure 4.2	Number of requests in CDC for the realistic scenario.	28
Figure 5.1	Experiment 1: Number of PMs that are turned on over time. . . .	34
Figure 5.2	Experiment 1: Power consumption over time.	34
Figure 5.3	Experiment 1: Run-time of the resource allocation computation.	35
Figure 5.4	Experiment 1: Resource utilization of the different methods. . .	35
Figure 5.5	Experiment 2: Number of PMs that are turned on over time. . . .	36
Figure 5.6	Experiment 2: Power consumption over time.	37
Figure 5.7	Experiment 2: Run-time of the resource allocation computation.	38
Figure 5.8	Experiment 3: Number of PMs that are turned on over time. . . .	38
Figure 5.9	Experiment 3: Power consumption over time.	39
Figure 5.10	Experiment 3: Run-time of the resource allocation computation.	39
Figure 5.11	Experiment 4: Number of PMs that are turned on over time. . . .	41
Figure 5.12	Experiment 4: Power consumption over time.	41
Figure 5.13	Experiment 4: Run-time of the resource allocation computation.	42
Figure 5.14	Experiment 5: Number of PMs turned on in the start phase. . . .	43

Figure 5.15	Experiment 5: Power consumption of CDC in the start phase. . . .	44
Figure 5.16	Experiment 5: CPU utilization of CDC in the start phase.	44
Figure 5.17	Experiment 5: FPGA utilization of CDC in the start phase.	45
Figure 5.18	Experiment 5: Peripheral utilization of CDC in the start phase. . .	46
Figure 5.19	Experiment 5: Number of PMs turned on in the steady state. . . .	46
Figure 5.20	Experiment 5: Power consumption of CDC in the steady state. . . .	47
Figure 5.21	Experiment 5: CPU utilization of CDC in the steady state.	47
Figure 5.22	Experiment 5: FPGA utilization of CDC in the steady state.	48
Figure 5.23	Experiment 5: Peripheral utilization of CDC in the steady state. . .	48
Figure 5.24	Experiment 6: Number of PMs turned on in the steady state for [32,64] CPU cores (Case 1).	51
Figure 5.25	Experiment 6: Power consumption of CDC in the steady state for [32,64] CPU cores (Case 1).	51
Figure 5.26	Experiment 6: Number of PMs turned on in the steady state for [128,256] CPU cores (Case 3).	52
Figure 5.27	Experiment 6: Power consumption of CDC in the steady state for [128,256] CPU cores (Case 3).	52
Figure 5.28	Experiment 7: Number of PMs turned on in the steady state for [32,64] CPU cores (Case 1).	55
Figure 5.29	Experiment 7: Power consumption of CDC in the steady state for [32,64] CPU cores (Case 1).	55
Figure 5.30	Experiment 7: Number of PMs turned on in the steady state for [64,128] CPU cores (Case 2).	56
Figure 5.31	Experiment 7: Power consumption of CDC in the steady state for [64,128] CPU cores (Case 2).	56

Figure 5.32 Experiment 7: Number of PMs turned on in the steady state for [128,256] CPU cores (Case 3). 57

Figure 5.33 Experiment 7: Power consumption of CDC in the steady state for [128,256] CPU cores (Case 3). 57



LIST OF ALGORITHMS

ALGORITHMS

Algorithm 1	Helper function to check if j th request's k th alternative can be allocated to i th server	22
Algorithm 2	OpenStack Algorithm to minimize power	30
Algorithm 3	Modified OpenStack Algorithm to minimize power	31

LIST OF ABBREVIATIONS

ABBREVIATIONS

BBM	Bulut Bilişim Merkezi
CDC	Cloud Data Center
CPU	Central Processing Unit
CTFD	Computation Time For Decision
FPGA	Field Programmable Gate Array
GPU	Graphics Processing Unit
IaaS	Infrastructure as a Service
ILP	Integer Linear Programming
IPaaS	Infrastructure or Platform as a Service
OS	Operating System
PaaS	Platform as a Service
PM	Physical Machine
SaaS	Software as a Service
SLA	Service Level Agreement
TPU	Tensor Processing Unit
VM	Virtual Machine

CHAPTER 1

INTRODUCTION

Cloud Data Centers (CDC) dynamically serve centralized physical resources to users by virtualizing them [1]. CDCs offer scalability, reliability and upgrades at a fraction of the cost of privately owning such resources [2][3]. Its internet-based self-serve and on-demand nature provides a great accessibility [4]. CDCs are used both by individuals and businesses for these reasons [5]. As their popularity increase, power consumed by CDCs become even more significant. Even though CDCs consume up to 20% less power than their privately owned counterparts [6], estimates show CDCs will consume more than 1,900TW power in 2020 [7]. Mismanagement of CDC resources further increases the power consumption of a CDC [8] [9] [10]. Therefore resource management approaches to reduce power consumption is a great area of interest due to its environmental and operational impacts.

Resource allocation methods aim to create a configuration that will meet requests of the users according to different performance goals and constraints with the existing physical resources. Heterogeneous cloud architectures having computational resources such as GPU (Graphics Processing Unit)[11], TPU (Tensor Processing Unit) [12] as well as FPGA based accelerators [13] [14] in addition to classical resources like CPU draw attention from both academia and industry. In heterogeneous clouds, resource allocation methods need to cover new computation resources in addition to standard existing resources such as CPU, memory, disc, bandwidth (BW) [13] [14]. Furthermore, a cloud resource manager assigns resources for SaaS (Software as a Service) requests, depending on the type of software the user requests.

In heterogeneous clouds a SaaS request can be satisfied by using standard CPUs or alternatively by using other computing resources introduced by the heterogeneous

clouds. Each of these alternatives have different performance constraints and resource usage costs. Any resource allocation method should keep up with the demand arrival rate without compromising an efficient solution and adding too much latency.

In the literature, [15] considers a cloud with CPUs and FPGAs. Proposed method divides tasks into independent chunks, and allocates chunks to meet deadlines and minimize energy. Method is tested on a setup with one server and one FPGA. Results show a 57% difference in configuration with FPGA and CPU compared to a configuration with only CPU. [16] and [17] predicting resource utilization of users in a homogeneous CDC. Both reduces power consumption by placing users according to predictions. [16] further reduces SLA violations by using predictions to detect which servers will be over-utilized and then migrating users from those servers. [5] considers a CDC with different CPUs. Optimizes energy consumption and task duration. Method has been tested on CloudSim with 15% decrease in energy consumption. [4] achieves load balancing and lower energy consumption. Proposed method is evaluated on a small cluster of 20 servers. [18] defines bidding based truthful greedy methods as an ILP problem for grid jobs. Methods are tested with real workloads taken from Grid Workloads Archive and Parallel Workloads Archive. [19] uses a heuristic approach to minimize resource fragmentation and number of active servers by using VM migration. Finally, in our previous works [20], [21] we evaluated the performance of proposed resource manager under randomly generated workload.

Contributions of this work are as follows:

- The main result of this thesis is the development of a novel resource allocation method for minimizing power consumption of heterogeneous Cloud Data Centers. Method groups requests and considers all alternatives simultaneously to make the optimal allocation. To best of our knowledge, no such method exists in the literature.
- Proposed method is implemented and tested on a simulation environment under a realistic workload along with an existing method and its extension to heterogeneous Cloud Data Centers.
- Evaluation of conducted simulations show the benefit of heterogeneous CDCs

over non-heterogeneous CDCs and benefit of proposed method over other methods.

The remainder of the thesis is organized as the following. Chapter 2 introduces the necessary background information on Cloud Data Centers(CDC) and resource allocation for CDCs, along with the performance metrics to compare various resource allocation methods. Chapter 3 proposes a novel resource allocation method, named ACCLOUD-MAN, for heterogeneous CDCs. In Chapter 4 ACCLOUD-MAN's performance is evaluated in a simulated CDC and Chapter 5 presents the results of simulations. Further discussions regarding the generality and implications of the proposed method are made in Chapter 6. Chapter 7 concludes the thesis.





CHAPTER 2

CLOUD DATA CENTERS: BACKGROUND

2.1 Cloud Resources and Offered Services

Typically CDCs include resources such as CPU, Memory, Disc Space and Network Bandwidth [13] [14] These resources are offered to the users in three distinct service levels. Namely Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). For IaaS, the user is given a dedicated Virtual Machine (VM), with specified hardware resources. Examples to IaaS are Amazon EC2[14] and Google Compute Engine [11] For PaaS, on top of IaaS the cloud provider serves a managed software development or deployment environment. Some PaaS examples are Microsoft Azure[13] and AWS Elastic Beanstalk[22]. In SaaS, user utilizes a software implemented by the cloud service provider and running on a VM. The user is completely abstracted away from the VM as access to the software is typically over a web interface. Two SaaS examples are Salesforce[23] and Dropbox[24]. For IaaS and PaaS, users explicitly declare the types and amounts of resources they require. However, SaaS users only declare the type of software they request. The physical resource types and amounts for the software that the user will be utilizing are decided by the cloud service provider.

2.2 Heterogeneous Cloud Data Centers

A *Heterogeneous CDC* may include more than one type of compute resources such as CPUs, GPUs, FPGAs and TPUs [25]. Heterogeneous CDCs attract users as they offer a one stop shop to users with various needs.

Heterogeneous CDCs are becoming widespread. Cloud providers such as Amazon [14] and Microsoft [13] have included FPGA services to their existing CDCs. Google is offering TPUs [12]. All of these providers offer GPUs as well. Inclusion of ASICs to CDCs are also being considered [26].

Main benefits of heterogeneous CDCs are performance and power. [15] reports 57% less power consumption for video processing jobs for a CDC with CPU and FPGA, comparing to a CDC with only CPU. [12] reports for machine learning jobs, using cloud TPU resources results in 27 times higher performance and 38% lower cost, in comparison to using cloud GPU resources. [27] reports for Amazon's EC2 CDC, using FPGA resources accelerate applications up to 30 times compared to using CPU resources.

2.3 Cloud Computing Resource Management

The users' IaaS/PaaS/SaaS requests are sent to the CDC by means of some software platform such as OpenStack [28]. These requests are then processed by a Resource Manager entity that decides about the allocation of the actual physical resources to grant the service requests. The requested resources are explicitly stated for IaaS/PaaS where SaaS requests further require determining the composition and amount of the physical resources by the Resource Manager. A Service Level Agreement (SLA) between the user and the cloud service provider governs the constraints for the resource allocation .

These constraints define a feasible set of resource allocation decisions. Then, the Resource Manager makes the allocation decision to achieve certain performance *goals* for the CDC such as minimizing the number of running servers, consumption of power or energy or task's termination time for SaaS user requests.

Here it should be noted that it is possible to determine different compositions and amounts of resources for SaaS requests that we call *resource alternatives* provided that the SLA constraints are satisfied. On the one hand, a SaaS requests can be granted by an alternative with a large number of CPU cores to achieve minimal response time in a user-centric business model. On the other hand, a minimum number of

CPU cores can be selected to minimize the power consumption. In addition to the amount of resources of a certain type, the types of the resources might differ among the alternatives for the heterogeneous CDCs. To this end, an encryption task offered as SaaS can be carried out purely in software or on GPUs or on a special purposed accelerator that is realized on FPGA defining the different alternatives to grant the request with different response times and power consumption.

Here it is emphasized that the current state of the CDC in terms of allocated and available physical resources affect the suitability of certain resource alternatives to the resource manager's goals. To this end, a certain alternative might not be feasible because of the lack of resources to realize it or additional power consumption overhead due to the requirement of turning on a new physical machine.

Use of the resource alternatives is demonstrated with an example. Consider a simple CDC with two types of resources: CPU and FPGA. Said CDC hosts a SaaS that can be run with 3 CPU units or 2 FPGA units. The goal of the resource manager for that CDC is to minimize the amount of power utilized. Also say it is given that 1 utilized CPU unit for said CDC consumes 2 units of power, 1 utilized FPGA unit consumes 1 unit of power and any running server consumes 5 units of power independent from its utilized CPUs and FPGAs. With the given system description, the FPGA alternative suits better to the manager's goal. However, consider a state of the CDC such that there is only one running server with 3 available CPU units and 1 available FPGA unit. In order to provide service with FPGA alternative, a new server has to be turned on; causing a higher power consumption than what the CPU alternative would have.

As demonstrated by the above example, resource alternatives provides flexibility to the manager by increasing the number of ways a user request can be met. This added flexibility may result in a better resource allocation according to the goals. However, increase in flexibility also increases the complexity of the problem of resource management; as will be discussed in detail in Section 3.3

The following performance metrics are considered to evaluate and compare proposed resource allocation method to existing methods in the literature.

Average and Instantaneous Power Consumption is defined as the time average of

the power consumed by the CDC during time interval of interest and power consumed by the CDC at a given time respectively. Power consumption of a CDC should be kept as low as possible, in accordance with green computing [29]. From the CDC provider's perspective, lower power means lower operational costs. For the extent of this study, it is assumed that the time a user remains in the CDC is not affected by the allocation decision (i.e. the choice of server or computation device has no effect on user's service duration). With this assumption, power and energy consumption of CDC are analogous. Due to its simplicity in calculations, power is chosen over energy as a metric. An extension to proposed algorithm, which does not necessitate the above mentioned assumption is discussed in Section 6.2. Such extension removes the analogy between power and energy, thus in that case energy consumption should be analyzed as a separate performance metric.

Computation Time For Decision (CTFD) is defined as the time manager requires for computing the allocation decision, given the state of the CDC and relevant user requests. It is desired to have a lower CTFD. CTFD for each request must at least be shorter than inter-arrival time of new requests; otherwise requests will build up indefinitely at the CDC's entry queue. For users, smaller CTFD means time between their request submission and CDC's response to that request will be shorter, resulting in a higher user satisfaction [30]. This is the only performance metric observable by the user, as any decision made is certain to satisfy SLAs by design and decision do not affect the service duration.

Maximum number of running servers is the highest number of concurrently running servers in the CDC. **Average number of running servers** is the time average of number of concurrently running servers in the CDC during time interval of interest. A lower number of running servers mean the workload can be satisfied with a smaller CDC, reducing the hardware costs.

Utilization of a Resource is defined as the ratio of total used amount of that resource type to the total existing amount of resources of that type in running servers. A high utilization shows less resources are being wasted by sitting idle in running servers.

CHAPTER 3

ACCLOUD-MAN: ACCELERATED CLOUD RESOURCE MANAGER

This chapter formulates the proposed cloud computing resource management framework ACCLOUD-MAN in this thesis. First, the premise of resource optimization problem is described. Then Integer Linear Programming (ILP), the mathematical optimization method that we will utilize to define and solve the resource optimization problem which will be defined later in the chapter, is introduced. Then the proposed resource manager that we call ACCLOUD-MAN (ACcelerated CLOUD MANager) is formulated as an ILP problem. The complexity of the problem is analyzed and various ILP solvers are compared for their suitability to formulated problem. Finally an extra constraint to the server selection is added to formulation, to reduce complexity at the cost of diverging from optimal solution.

Cloud data centers (CDC) offer certain types of resources to the user. In this work, we consider that 5 different resource types are offered by physical machines (PM) in a CDC: CPU, FPGA, Memory, Disc and Network Bandwidth. However, the proposed method is applicable for any type and number of resources. In analogy to our previous work [31], we assume that FPGAs are virtualized by partitioning them into smaller reconfigurable regions. These regions are then served to the user as standalone computing resources or as hardware accelerators with traditional CPUs.

Users can request resources from the CDC in two ways. For IaaS/PaaS (IPaaS) requests in accordance with the service models in Section 2.1, user explicitly specifies the amount of requested resources for each resource type. For SaaS requests, user only specifies the requested cloud application. The amount of resources required for such SaaS request is then determined by the resource manager, taking into account that there may be more than one resource alternative to meet a SaaS request. It is

assumed that the time a user remains in the CDC is not affected by the allocation decision (i.e. the choice of server or computation device has no effect on user's service duration). An extension of the proposed method to cases where this assumption does not hold is explored in Section 6.2.

3.1 Integer Linear Programming (ILP)

Integer Programming is a mathematical optimization, where some of the decision variables are constrained to be integers [32]. A system of inequalities, defined in terms of decision variables, constraints the solution space. Another equation in terms of decision variables, called the objective function, is defined. The purpose of the problem is to find the set of decision variables, which minimizes the value of objective function while not violating the set inequalities. If the set of inequalities and the objective function are linear in terms of decision variables, optimization problem is further called as Integer Linear Programming (ILP) problem.

3.2 Formulation of the ILP Problem

This section formulates the ACCLOUD-MAN functionality in the form of an ILP. Hereby, a set $\mathbb{PM} = \{PM_1, \dots, PM_N\}$ of available servers is assumed, whereby each server $PM_i \in \mathbb{PM}$ offers CPU, FPGA, Memory, Disc, and Network Bandwidth resources defined by C_i, F_i, M_i, D_i, B_i , respectively. CPU and FPGA resources are measured and offered to the user in terms of number of cores and reconfigurable regions (module) respectively. Memory, Disc are in terms of GB and Bandwidth is in terms of Mbps.

The main task of ACCLOUD-MAN is to assign user requests to the servers in \mathbb{PM} . Since user requests arrive sporadically, ACCLOUD-MAN collects a set of K pending requests $\mathbb{REQ} = \{\mathbb{REQ}_1, \dots, \mathbb{REQ}_K\}$ within a pre-defined time interval and then determines a suitable assignment of these requests to servers at the end of this time interval. We denote the time instant when a decision is made as a decision instant of ACCLOUD-MAN. This time interval should be chosen such that it would accumu-

late enough user requests but should not keep users waiting too much for their request to be evaluated, implications of delaying requests is further discussed in Section 6.3. Hereby, it is assumed that each request \mathbb{REQ}_j can be served as one of n_j alternatives. That is, $\mathbb{REQ}_j = \{\text{req}_{j,1}, \dots, \text{req}_{j,n_j}\}$ is a set of alternatives $\text{req}_{j,k}$ and each alternative $\text{req}_{j,k}$ requires the physical resources $c_{j,k}$ (CPU), $f_{j,k}$ (FPGA), $m_{j,k}$ (Memory), $d_{j,k}$ (Disc), $b_{j,k}$ (Bandwidth).

In order to formulate the desired minimization of the power consumption, we introduce a model of the power consumption of a CDC. In the form of $P_{total} = P_{newPM} + P_{comp}$ which is due to turning on new PMs, P_{newPM} and allocating the requests to the already on and newly turned on PMs, P_{comp} [4]. For CPU cores, polynomial power model proposed in [33] is used. Proposed model is in the form of $a_0 + a_1u + a_2u^2$ where u is utilization rate of the core. We assume that if a core is allocated to a user then it is fully utilized ($u = 1$); if not, it is not utilized at all ($u = 0$). Therefore if PM has N many CPU cores and M many allocated CPU cores, its power consumption would be $a_0N + (a_1 + a_2)M$. We further assume, utilization of peripherals are linearly correlated to number of allocated CPUs. [33] proposes linear power models for Memory and Disc while [34] proposes a linear power model for networking devices under constant transmission rate, in terms of their utilization. Thus, power consumption of peripherals with respect to number of utilized CPU cores can be written as $P_{perip_static} + MP_{perip_cpu}$. By using the same assumption regarding the utilization, FPGA power consumption can be written as $P_{fpga_static} + KP_{fpga_reg}$, where K is the number of allocated FPGA regions. Dynamic peripheral power consumption due to FPGA can be written as KP_{perip_fpga} . Total PM power consumption can be written as $(a_0N + P_{perip_static} + P_{fpga_static}) + (P_{perip_cpu} + a_1 + a_2)M + (P_{fpga_reg} + P_{perip_fpga})K$, where the first term corresponds to P_{newPM} and other terms corresponds to P_{comp} . It should be noted that, in the above expression only M and K are variables and the rest are determined by the type and model of used resources.

Hereby, we take into account that the CDC is in a well-defined state at each decision instant of ACCLOUD-MAN. That is, a certain number of servers is on, whereas the remaining servers are off. We introduce the parameter on_i such that $on_i = 1$ if PM_i is on and $on_i = 0$ if PM_i is off right before the decision instant. In addition, we introduce a decision variable \bar{q}_i for each PM_i such that $\bar{q}_i = 1$ if PM_i is on and

$\bar{q}_i = 0$ if PM_i is off after the decision instant. We emphasize that on_i is a given parameter, whereas \bar{q}_i is a decision variable to be computed by ACCLOUD-MAN. To avoid confusion between decision variables and constants, decision variables will be denoted with an overline.

The power consumed by each server depends on the state of the server (on/off) and the used resources of the requests served by the server. Accordingly, we introduce $P_{\text{on},i}$ as the idle power consumption of PM_i . In addition, $P_{\text{CPU},i}$ and $P_{\text{FPGA},i}$ indicate the power consumption of one CPU core and one FPGA module of PM_i , respectively in accordance with the introduced power model.

Using the defined parameters and variables, it is now possible to determine the additional power consumption when allocating the pending requests in \mathbb{REQ} to servers in \mathbb{PM} . The additional power consumption consists of two components. First, there is the power P_{newPM} of servers that are newly switched on. It evaluates to

$$P_{\text{newPM}} = \sum_{i=1}^N P_{\text{on},i} \cdot (1 - \text{on}_i) \cdot \bar{q}_i. \quad (3.2.1)$$

In (3.2.1), $(1 - \text{on}_i) \cdot \bar{q}_i$ evaluates to 1 if PM_i was off before the decision instant ($1 - \text{on}_i = 1$) and is on after the decision instant ($\bar{q}_i = 1$). Second, there is the power P_{comp} , which is the power used by the newly assigned CPU cores and FPGA modules. In order to compute this power, we introduce the decision variable $\bar{s}_{i,j,k}$, which is 1 if alternative $\text{req}_{j,k}$ of request \mathbb{REQ}_j is allocated to PM_i and 0 otherwise. Then, we obtain

$$P_{\text{comp}} = \sum_{ijk} (P_{\text{CPU},i} c_{j,k} + P_{\text{FPGA},i} f_{j,k}) \cdot \bar{s}_{i,j,k} \quad (3.2.2)$$

Together, we want to minimize the power consumption, which amounts to

$$\min F = P_{\text{newPM}} + P_{\text{comp}}. \quad (3.2.3)$$

While minimizing F in (3.2.3), various constraints on the resources have to be respected. First, it must hold that each server has enough resources for the requests allocated to it. (3.2.4) to (3.2.8) represent these constraints for the different resource

types: CPU, FPGA, Memory, Disc and Bandwidth.

$$\sum_{j=1}^K \sum_{k=1}^{n_j} c_{j,k} \bar{s}_{i,j,k} \leq C_i \quad \forall i \in PM \quad (3.2.4)$$

$$\sum_{j=1}^K \sum_{k=1}^{n_j} f_{j,k} \bar{s}_{i,j,k} \leq F_i \quad \forall i \in PM \quad (3.2.5)$$

$$\sum_{j=1}^K \sum_{k=1}^{n_j} m_{j,k} \bar{s}_{i,j,k} \leq M_i \quad \forall i \in PM \quad (3.2.6)$$

$$\sum_{j=1}^K \sum_{k=1}^{n_j} d_{j,k} \bar{s}_{i,j,k} \leq D_i \quad \forall i \in PM \quad (3.2.7)$$

$$\sum_{j=1}^K \sum_{k=1}^{n_j} b_{j,k} \bar{s}_{i,j,k} \leq B_i \quad \forall i \in PM \quad (3.2.8)$$

In addition, it must hold that each alternative $\text{req}_{j,k}$ of a given request REQ_j is assigned to a unique server. That is,

$$\sum_{i=1}^N \sum_{k=1}^{n_j} \bar{s}_{i,j,k} = 1 \quad \forall \text{REQ}_j \in \text{REQ}. \quad (3.2.9)$$

Finally, it must be the case that any server that serves at least one request is on. This is represented by

$$\sum_{j=1}^K \sum_{k=1}^{n_j} \bar{s}_{i,j,k} \leq \bar{q}_i \cdot A_1 \quad \forall PM_i \in \mathbb{PM}. \quad (3.2.10)$$

Here, A_1 is a large integer value that is larger than the maximum number of requests that can be handled by a single server.

Altogether, the resource allocation problem of ACCLOUD-MAN is given by minimizing F in (3.2.3) subject to the constraints in (3.2.4) to (3.2.10). It is readily observed that this optimization problem is an ILP. According to the described operation of ACCLOUD-MAN, this optimization problem has to be solved at each decision instant. The result is the information about all servers that need to be on ($\bar{q}_i = 1$), the selection of resource alternatives $\text{req}_{j,k}$ and their assignment to a server PM_i if $\bar{s}_{i,j,k} = 1$.

We denote the server and alternative selected for a given request REQ_j as \overline{pm}_j and \overline{alt}_j , respectively. In order to determine \overline{pm}_j and \overline{alt}_j , the following equations can be

used.

$$\overline{pm}_j = \sum_{i=1}^N \sum_{k=1}^{n_j} (\bar{s}_{i,j,n} \cdot i) \quad \forall \text{REQ}_j \in \text{REQ}, \quad (3.2.11)$$

$$\overline{alt}_j = \sum_i^N \sum_{k=1}^{n_j} (\bar{s}_{i,j,n} \cdot k) \quad \forall \text{REQ}_j \in \text{REQ} \quad (3.2.12)$$

It has to be noted that the defined optimization problem is suitable for all different types of services. In the case of IPaaS requests, the physical resource requirement is given by a single alternative. For SaaS requests, alternatives are determined according to the above-mentioned alternative database.

3.3 Complexity Analysis

In the above formulation, the cardinalities of the sets PM and REQ is N and K , respectively. If there is only one alternative per request, then the complexity of the resource allocation problem is $\mathcal{O}(N^K)$. Otherwise, the complexity is $\mathcal{O}((NL)^K)$ if the average number of alternatives per request is L .

3.4 Comparison of ILP Solvers

We compare 5 ILP solvers and an additional brute force solver for their proximity to optimal solution and computation times. These 5 solvers are namely:

- MATLAB's ILP solver
- TOMLAB's CPLEX solver
- TOMLAB's mipSolve
- TOMLAB's miqpBB
- TOMLAB's minlpBB

Solvers are tested against different numbers of randomly generated servers and requests. For each server, the values for Memory (M_i), Disc (D_i) and Bandwidth (B_i)

are randomly selected from a normalized interval $[80, 100]$, whereby 100 represents the maximum possible resource. Likewise, the number of CPU cores (C_i) and the number of FPGAs (F_i) is randomly chosen from the intervals $[16, 32]$ and $[4, 8]$, respectively. Regarding the power consumption, a potentially high power in the interval $[900, 1100]$ is needed to turn on a server ($P_{on,i}$), whereas the power consumption for each CPU ($P_{CPU,i}$) and FPGA ($P_{FPGA,i}$) are in the intervals $[15, 25]$ and $[10, 20]$, respectively. In a CDC with N servers, each server is randomly generated with values from the stated intervals. The number of alternatives for a request is chosen randomly from the interval $[1,3]$. The CPU requirement ($c_{j,k}$) and the FPGA requirement ($f_{j,k}$) are selected randomly from the intervals $[0, 8]$ and $[0, 4]$. The remaining resources (Memory – $m_{j,k}$, Disc – $d_{j,k}$, Bandwidth – $b_{j,k}$) are selected from the normalized interval $[0, 20]$. Table-3.1 to 3.5 shows computation time results. Cases where solver failed due to insufficient memory are notated as "NA".

Table 3.1: MATLAB MILP solver’s computation times for various cases. Results are given in seconds.

		Number Of Servers				
		10	20	50	100	150
Number of Requests	2	0.01	0.08	0.15	0.23	0.25
	5	0.03	0.04	0.43	0.85	1.09
	7	0.19	0.21	0.53	1.67	6.32
	10	0.62	1.13	7.33	26.14	75.76
	15	1.87	44.78	631.41	637.77	NA

All solvers yield the optimal solution for all cases. Except miqpBB, for the cases with more than 100 servers. In those cases miqpBB output an incorrect solution (i.e. assigns requests to servers with insufficient resources).

TOMLAB CPLEX solver is chosen for further studies, as it is the fastest solver among the ones tested. Therefore, CPLEX solver is tested on larger simulated CDCs with randomly initialized servers and user requests.

Table 3.2: TOMLAB CPLEX solver’s computation times for various cases. Results are given in seconds.

		Number Of Servers				
		10	20	50	100	150
Number of Requests	2	0.01	0.01	0.02	0.02	0.03
	5	0.02	0.03	0.05	0.09	0.13
	7	0.03	0.05	0.12	0.42	0.70
	10	0.16	0.65	1.32	3.96	6.64
	15	0.43	1.55	4.39	25.15	33.79

Table 3.3: TOMLAB mipSolve’s computation times for various cases. Results are given in seconds.

		Number Of Servers				
		10	20	50	100	150
Number of Requests	2	0.36	0.51	1.03	4.55	7.63
	5	0.78	1.57	141.98	235.33	405.49
	7	10.81	75.54	146.37	NA	NA

Results on Table 3.7 shows that, even the fastest ILP solver is too slow for making an allocation decision in a larger CDC in a reasonable time. Considering an actual CDC is expected to have far more servers, proposed approach must be modified to reduce its computational complexity.

3.5 Reducing Complexity by Limiting the Number of Servers to Be Considered

One way of reducing the computational complexity is to reduce the number of servers that are included in the optimization problem. For example, assume that a number of

Table 3.4: TOMLAB miqpBB's computation times for various cases. Results are given in seconds.

		Number Of Servers				
		10	20	50	100	150
Number of Requests	2	0.01	0.05	0.45	0.08 [†]	0.03 [†]
	5	0.13	0.31	4.72	0.05 [†]	0.09 [†]
	7	0.12	7.30	75.89	0.07 [†]	0.12 [†]
	10	0.72	2.59	69.99	0.11 [†]	0.19 [†]
	15	41.31	49.31	146.87	0.21 [†]	0.26 [†]

[†]For these cases, solver outputted an invalid result in a relatively short time.

K request has to be allocated and a number of N_{on} servers is currently on. Then, it is possible to include a reduced number of $N_{\text{on}} + \gamma \cdot K$ servers (instead of N servers) in the optimization problem, leading to a reduced computational complexity of $\mathcal{O}(((N_{\text{on}} + \gamma \cdot K) \cdot L)^K)$. Hereby, γ is a coefficient that ensures solvability of the ILP. It is expected that the decisions taken with a smaller number of servers will slightly deviate from the optimal solution. This idea is further explored in the Section 5.1.4.

Table 3.5: TOMLAB minlpBB solver's computation times for various cases. Results are given in seconds.

		Number Of Servers				
		10	20	50	100	150
Number of Requests	2	0.09	0.17	0.40	10.60	90.52
	5	0.58	1.11	334.97	339.30	889.52
	7	2.43	10.45	280.28	784.24	1339.13
	10	72.29	110.10	408.70	1312.40	NA

Table 3.6: Brute force solver's computation times for various cases. Results are given in seconds.

		Number Of Servers		
		10	20	50
Number of Requests	2	0.01	0.02	0.33
	5	27.80	1778.24	NA
	7	10907.56	NA	NA

Table 3.7: TOMLAB CPLEX solver's computation times for more cases. Results are given in seconds.

		Number Of Servers								
		10	20	50	100	150	200	250	300	400
Number of Requests	2	0.01	0.01	0.02	0.02	0.03	0.04	0.05	0.06	0.07
	5	0.02	0.03	0.05	0.09	0.13	0.30	0.34	0.45	0.64
	7	0.03	0.05	0.12	0.42	0.70	0.99	1.29	1.89	2.92
	10	0.16	0.65	1.32	3.96	6.64	10.93	18.84	28.79	46.81
	15	0.43	1.55	4.39	25.15	33.79	89.68	186.84	532.55	646.02



CHAPTER 4

ACCLOUD-MAN EVALUATION

4.1 Algorithms for Comparison

In order to observe the improvement on the performance metrics introduced in Section 2.3 by the proposed algorithm, similar algorithms should be defined; however, without the novelties of the proposed algorithm.

4.1.1 OpenStack Algorithm

This is an algorithm that can be implemented on OpenStack's Nova Scheduler's [35] filter and weight based framework. In this framework, servers are first filtered out according to certain properties of a user request (e.g. a server can be filtered out if it has less memory than user requires). Then choice is made among the remaining servers according to weights defined for some server properties. The filtered and weighted properties are set by the administrator of the OpenStack setup.

This simple algorithm considers each request one by one. If the request has resource alternatives, only the alternative with the least expected power consumption is considered. Expected power consumption of alternatives is calculated as the power the alternative would consume with the average CPU and FPGA in the CDC, as shown in Equations 4.1.1 and 4.1.2. Then the chosen alternative is simply considered for allocation in all servers with adequate resources. The request is allocated to the server that would lead to the least increase in the CDC power consumption according to Equation 4.1.3. A helper function checking if a request can be allocated to a certain server

and detailed steps of the algorithm are given in Algorithms 1 and 2 respectively.

$$P_{CPU,avg} = \frac{\sum_i P_{CPU,i}}{|\mathbb{PM}|} \quad \forall PM_i \in \mathbb{PM} \quad (4.1.1)$$

$$P_{FPGA,avg} = \frac{\sum_i P_{FPGA,i}}{|\mathbb{PM}|} \quad \forall PM_i \in \mathbb{PM} \quad (4.1.2)$$

$$P_{increase,ijk} = P_{on,i} \cdot (1 - on_i) + P_{CPU,i}c_{j,k} + P_{FPGA,i}f_{j,k} \quad (4.1.3)$$

Function DoesFitInServer (i, j, k):

```

if  $c_{j,k} \leq C_i$  and  $f_{j,k} \leq F_i$  and  $m_{j,k} \leq M_i$  and  $d_{j,k} \leq D_i$  and  $b_{j,k} \leq B_i$ 
  then
    return true
  else
    return false
end

```

Algorithm 1: Helper function to check if j th request's k th alternative can be allocated to i th server

This algorithm ignores the resource alternatives. The alternative fitting better to the resource management goal is chosen once, according to the properties of the CDC. State of the CDC does not affect the chosen alternative. Also algorithm does not bundle several requests together. Instead another request is taken into consideration only after the request at hand is allocated to a server. Complexity of this algorithm is $\mathcal{O}(N)$ where N is the number of servers in the CDC.

4.1.2 Modified OpenStack Algorithm

As an improvement to the above described OpenStack Algorithm, this algorithm considers resource alternatives during the allocation. The chosen alternative depends on the state of the CDC, as well as on the CDC properties. Similar to the OpenStack Algorithm, this algorithm considers each request one by one. As such, is a middle ground between the OpenStack Algorithm and the proposed algorithm. Steps of the approach is defined in Algorithm 3. Complexity of this algorithm is $\mathcal{O}(NL)$ where

N is the number of servers in the CDC and L is average number of alternatives for user requests. This algorithm can not be implemented on the filter and weight framework of the Nova scheduler, as OpenStack does not recognize resource alternatives inherently.

4.2 Simulation Environment

ACCLOUD-MAN is implemented in MATLAB, together with a simulation environment as shown in Figure 4.1. The "Simulation Controller" simulates a CDC with physical and software resource information from the "Cloud Assets" database. It sends request events from a "Request Traces" file to ACCLOUD-MAN via the "Controller Interface" and generates resource assignments according to incoming decisions. The "Simulation Settings" block adjusts the "Simulation Controller" on how to create groups of requests or smaller sets of servers. The "Log" block collects the data of all simulations and compares and reports the effectiveness of resource allocation strategies. The "Control Interface" communicates with the "Simulation Controller" via a TCP Socket and provides an interface to the "Allocation Strategist". The "Allocation Strategist" block groups incoming requests and sends them to the "ILP Solver" block. The "ILP Solver" block solves the ILP defined in Section 3.2 and transmits the resulting resource allocation decisions for each request to the "Allocation Strategist". The "Allocation Strategist" collects the decisions given for all request groups and sends them to the "Controller Interface" to be sent to the outside world. The current version of the simulation environment is designed to also be able to communicate with the OpenStack [28] software that is used in real-world applications for cloud resource allocation.

4.3 Simulation Setup

In order to evaluate the performance of ACCLOUD-MAN, we developed a simulation setup within the described environment. The task of the simulation setup is to

1. generate a CDC with a certain number of servers with different resources and

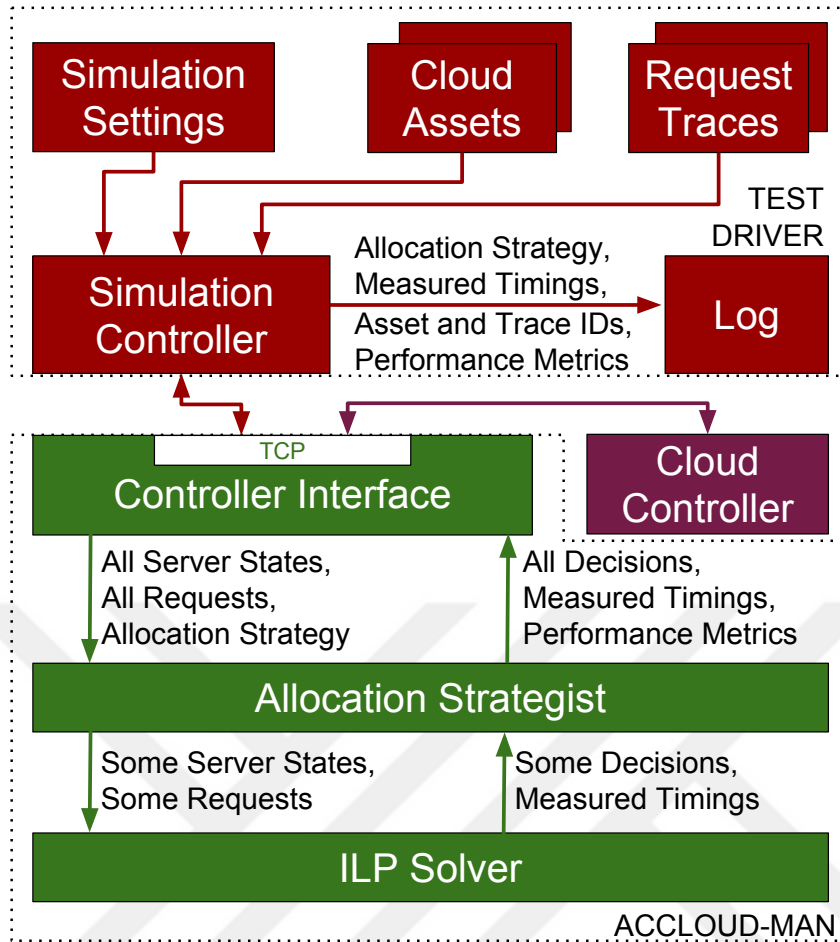


Figure 4.1: Software architecture of the resource manager.

power consumptions,

2. dynamically generate user requests for the CDC with different resource types,
3. determine a resource allocation for pending requests,
4. update the state (resource utilization and power consumption) of the CDC.

Algorithms are tested in two scenarios. In the first scenario, user requests are generated randomly. For the second scenario, user requests are generated to reflect the distribution of an actual CDC. From now on, former scenario will be referred to as *Random Request Scenario*, while the latter will be referred to as *Realistic Request Scenario*. Methods used to create the servers of CDC and user requests for each scenario are detailed in the following subsections.

4.3.1 Initialization of Servers for Random Request Scenario

We define intervals for the possible values of each resource type similar to the dataset in [36]. For each server, the values for Memory (M_i), Disc (D_i) and Bandwidth (B_i) are randomly selected from an normalized interval $[80, 100]$, whereby 100 represents the maximum possible resource. Likewise, the number of CPU cores (C_i) and the number of FPGAs (F_i) is randomly chosen from the intervals $[16, 32]$ and $[4, 8]$, respectively. Regarding the power consumption, a potentially high power in the interval $[900, 1100]$ is needed to turn on a server ($P_{\text{on},i}$), whereas the power consumption for each CPU ($P_{\text{CPU},i}$) and FPGA ($P_{\text{FPGA},i}$) are in the intervals $[15, 25]$ and $[10, 20]$, respectively. In a CDC with N servers, each server is randomly generated with values from the stated intervals.

4.3.2 Generation of User Requests for Random Request Scenario

Alternatives are generated for a number of K requests in a given time interval T . The number of alternatives for a request is chosen randomly from the interval $[1,3]$. In agreement with Section 3.2, the resource requirement (CPU, FPGA, Memory, Disc, Bandwidth) of each request alternative $\text{req}_{j,k}$ has to be specified. To this end, we proceed in a similar way as in item 1). The CPU requirement ($c_{j,k}$) and the FPGA requirement ($f_{j,k}$) are selected randomly from the intervals $[0, 16]$ and $[0, 4]$. The remaining resources (Memory – $m_{j,k}$, Disc – $d_{j,k}$, Bandwidth – $b_{j,k}$) are selected from the normalized interval $[0, 20]$. Hereby, it is respected that request alternatives might not need a certain type of resource.

4.3.3 CDC State Update for Random Request Scenario

We perform a simulation of the resource utilization of a CDC in two main stages. The first stage simulates a cold start and the second one simulates a steady state of the CDC. In the first stage, we initialize the CDC such that all PMs are turned off. Then, we periodically generate a number of K requests and assign them to PMs according to one of the methods in item 3). The first stage continues until a certain state of the CDC is reached where most of the open PMs are almost fully utilized for at least

one resource. The CDC state that is reached after the first stage can be considered as the state of an operating CDC with a high utilization of PMs. In the second stage, new requests are generated and currently served (active) requests are terminated in the CDC. K new requests are generated in a time interval T in the same way as in the first stage. Active requests for termination are randomly selected such that an average number of K requests is removed from the CDC within the time interval T . The second stage is run for a pre-defined amount of time in each experiment. In all our experiments, we assume that the rate of incoming requests is in the order of 1 request per 2.5 seconds similar to [36].

4.3.4 Initialization of Servers Realistic Request Scenario

Number of CPU cores for each PM is randomly chosen from the intervals of [32, 64], [64, 128] or [128, 256]; depending on the test cases which will be detailed in Section 5.2.2. Number of FPGA regions is randomly chosen from the [0, 4] interval. In the scope of another research work of ACCLOUD project, we are employing Xilinx XC7Z100 [37] boards and creating four reconfigurable regions. The amount of resources on each reconfigurable region is enough to implement a realistic accelerator [38]. Amounts of Memory, Disc and Bandwidth are chosen from the intervals of [320, 400]GB, [8, 10]TB and [32, 40]Gbps respectively. Regarding the power consumption, $(P_{on,i})$ and $(P_{CPU,i})$ are in the intervals [90, 100]W and [3, 5]W respectively as reported for commercial CDC servers in [33] or [8, 12]W/core and [9, 11]W respectively as obtained from Oracle's online server power calculator [39]; again depending on the test case. $(P_{FPGA,i})$ is in the interval of [1, 2]W.

4.3.5 Generation of User Requests Realistic Request Scenario

In order to generate realistic user requests, a model based workload generation method named CLOUDGEN [40] is employed. CLOUDGEN takes an actual cloud workload trace as input, performs k-means clustering on the trace data to create sub-distributions for each cluster that reflect the characteristics of the data more closely. Then appropriate distributions for the continuous parameters of VM inter-arrival time

and VM lifetime are fit to the clusters together with identifying the percentages of the discrete categorical parameters of number of cores and amount of RAM.

In the scope of this work, anonymized trace presented in [36] is used as input to CLOUDGEN. This trace was created by recording the information of all VMs running on Azure at the end of 2016 for 1 month. There are approximately 2 million VM records in the trace. k-means clustering is applied with the dimensions of memory, processor core count and VM lifetime. Number of clusters (k) to minimize $f(k)$ value is determined as defined in [41]. Different than [40], the second best k of 5 is chosen to have a more detailed representation of the VM types.

CLOUDGEN workload model is then enhanced by extending the requested VM types [42]. To this end, Disc and Bandwidth requests are added by mapping the existing VM flavors in the generated synthetic trace according to CLOUDGEN, to the ones that are currently offered by Azure [13]. Azure offers VMs with GPUs for accelerated computing. It is assumed that every VM with more than 6 vCPUs (which is the minimum vCPU count of Azure's GPU accelerated VMs) has a possibility of being an accelerated VM. 14% of the VMs in the workload model have more than 6 vCPUs and we extend a fraction of these VM requests with FPGA requests.

Synthetic trace is further extended with SaaS requests according to the contemporary cloud workload percentages stated in [43]. To this end, the final synthetic trace used for evaluation has a total of 708892 requests with the respective percentages of 9.6% of Social Networking, 16.2% of Video Streaming, 34.1% of Collaboration and 40.1% of IPaaS VM (Compute) requests. Exponentially distributed inter-arrival times are assumed for SaaS requests with proportional means to IPaaS requests. The mean lifetimes of the SaaS requests are selected arbitrarily as 3600s, 7200s and 5400s for Social Networking, Video Streaming and Collaboration respectively with an exponential distribution.

Duration of the generated trace is chosen as 16 weeks (112 days). Fig 4.2 shows the number of user requests in the CDC for the duration of the trace. Number of requests in the CDC reaches a steady state at around 15 days. In the steady state number of requests in the CDC is at around 1500.

Generated user request trace based on the original Azure trace has the average request arrival rate of 1.83 request/second. For a simulation of 16 weeks of CDC behavior, this rate results in a total of 17.7 million user requests. Simulating with that many requests is not feasible due to the time it would take. To solve this, we have multiplied the inter-arrival time of the user requests by 25 while keeping lifetimes constant. This increased the simulation speed in two ways. First, since the number of requests in the trace has dropped by 25 times; giving a speed-up of 25. Second, as lifetime to inter-arrival time ratio has dropped by 25 times, the number of requests in the CDC at a time has also dropped by 25 times. As number of concurrent request is reduced, number of PMs in the simulation can be reduced by the same amount. All considered algorithms have a linear dependency to number of PMs, thus by reducing the number of PMs 25 times another speed-up of 25 has gained, totaling at 625 times. For comparison, simulation of 16 weeks with the OpenStack algorithm takes 50 minutes. Without the 625 times speed-up due to increased inter-arrival time, it would take 31250 minutes or 21.7 days to simulate for a single case.

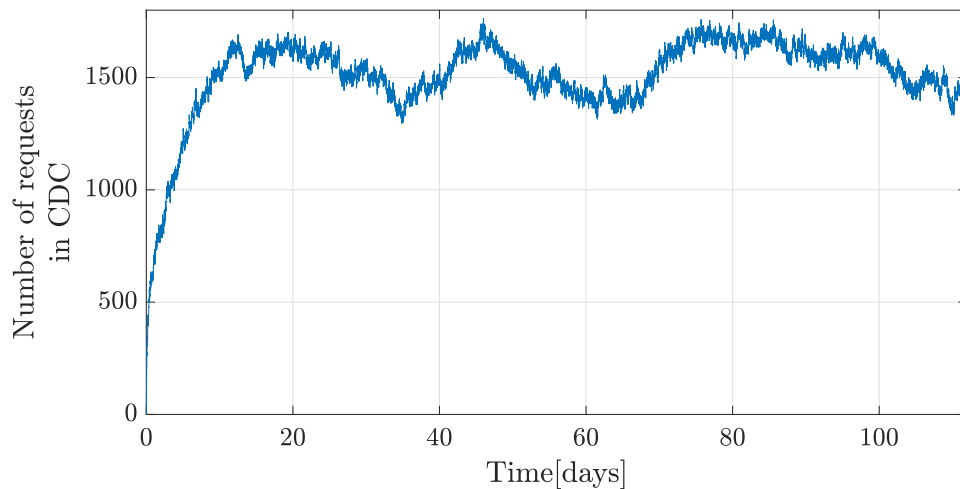


Figure 4.2: Number of requests in CDC for the realistic scenario.

Alternatives for the 3 Software services are suggested as presented in Table 4.1. Social Networking jobs include image processing, natural language processing and graph analysis; therefore, it is reasonably assumed that these jobs can be run on CPUs and FPGAs. Video Processing jobs are thought to be more flexible since each frame can be treated separately; thus, it is assumed these jobs run on CPU, FPGA and a

mix of CPU and FPGA as suggested in [15]. Collaboration jobs are less flexible, they mostly consist of shared tools designed for PC use; and therefore, it is assumed these jobs can only be run on CPUs with no other alternative. Here it is assumed that interactive Video Streaming and Collaboration services require more bandwidth, while Video Streaming and Social Networking services are more data intensive and require more memory. Here, it should be noted that the alternatives with FPGA usage require less power than the alternatives with CPU usage. With given values, Open-Stack algorithm considers only the first alternatives for Social Networking and Video Processing, since they consume the least power on the average PM.

4.3.6 CDC State Update Realistic Request Scenario

Simulation starts with an empty CDC (i.e. all PMs are off and all resources are free). Requests are generated as described above and are sent to the entry queue of the CDC. Whenever K requests are accumulated in the queue, they are assigned to the PMs according to one of the methods in item 3). Requests are evicted from the CDC when a their specified duration is exceeded. Requests' duration are not used in or known by the allocation decision mechanisms, it is only utilized to simulate the user's exit from the CDC.

```

input : RequestID,  $P_{CPU,avg}$ ,  $P_{FPGA,avg}$ 
output: chosenAlternative, chosenPM
j = RequestID
minPower =  $\infty$ 
chosenAlternative = -1
for  $k = 1, \dots, |\text{REQ}_j|$  do
    |  $tempPower = P_{CPU,avg}c_{j,k} + P_{FPGA,avg}f_{j,k}$ 
    | if  $tempPower < minPower$  then
    | | minPower = tempPower
    | | chosenAlternative = k
    | end
end
k = chosenAlternative
minPower =  $\infty$ 
chosenPM = -1
for  $i = 1, \dots, |\text{PM}|$  do
    | if  $DoesFitInServer(i, j, k)$  then
    | |  $tempPower = P_{on,i} \cdot (1 - on_i) + P_{CPU,i}c_{j,k} + P_{FPGA,i}f_{j,k}$ 
    | | if  $tempPower < minPower$  then
    | | | minPower = tempPower
    | | | chosenPM = i
    | | end
    | end
end

```

Algorithm 2: OpenStack Algorithm to minimize power

```

input : RequestID
output: chosenAlternative, chosenPM
j = RequestID
minPower =  $\infty$ 
chosenAlternative = -1
chosenPM = -1
for  $i = 1, \dots, |\text{PM}|$  do
  for  $k = 1, \dots, |\text{REQ}_j|$  do
    if DoesFitInServer( $i, j, k$ ) then
       $tempPower = P_{on,i} \cdot (1 - on_i) + P_{CPU,i}c_{j,k} + P_{FPGA,i}f_{j,k}$ 
      if  $tempPower < minPower$  then
        minPower = tempPower
        chosenAlternative = k
        chosenPM = i
      end
    end
  end
end

```

Algorithm 3: Modified OpenStack Algorithm to minimize power

Table 4.1: Values of resources requested by each alternative of Software services and the average power would be consumed on the average PM. CPU is in number of cores and FPGA is in number of regions. Memory, Disc, Bandwidth and Power are in GB, TB, Gbps and W respectively.

		CPU (core)	FPGA (region)	Mem (GB)	Disc (TB)	BW (Gbps)	Power (W)
Social Networking	Alt. 1	0	2	6	0.2	0.8	3
	Alt. 2	4	0	6	0.2	0.8	16
Video Processing	Alt. 1	0	2	6	0.2	1.6	3
	Alt. 2	4	0	6	0.2	1.6	16
	Alt. 3	2	1	6	0.2	1.6	9.5
Collaboration	Alt. 1	4	0	3	0.2	1.6	16

CHAPTER 5

EVALUATION RESULTS

5.1 Random Request Scenario

5.1.1 Experiment 1: Comparison of Resource Allocation Methods

In this section, we use the simulation setup described in Section 4.3 to validate the functionality of ACCLOUD-MAN (based on the ILP solution) in comparison to the resource allocation of OpenStack (OS) and the Modified OpenStack (MOS). To this end, we run experiments with $N = 400$ PMs, a number of $M = 10$ requests that are served together and $\gamma = 2$. That is, the ILP is formulated under the assumption that $\gamma \cdot M = 20$ candidate PMs can be added to the currently on PMs. In our experiments, the first stage of the simulation is completed after about 45 min and the second stage covers about 3 hours.

Exemplary results are shown in Fig. 5.1 to 5.3. It is readily observed from Fig. 5.1 and 5.2 that ACCLOUD-MAN achieves the smallest number of used PMs and the least amount of consumed power. The improvement compared to the OpenStack and Modified OpenStack (with alternatives) resource allocation is in the order of 10% and 5%, respectively. It can further be seen from Fig. 5.3 that the resource allocation computations can be performed in a small amount of time even when solving the ILP for ACCLOUD-MAN.

In addition, Fig. 5.4 shows the utilization of the CDC for the different resources and resource allocation methods. The fact that ACCLOUD-MAN uses the smallest number of PMs, is consistent with the observation that the resource utilization of ACCLOUD-MAN is the highest among the different methods. It is further interesting

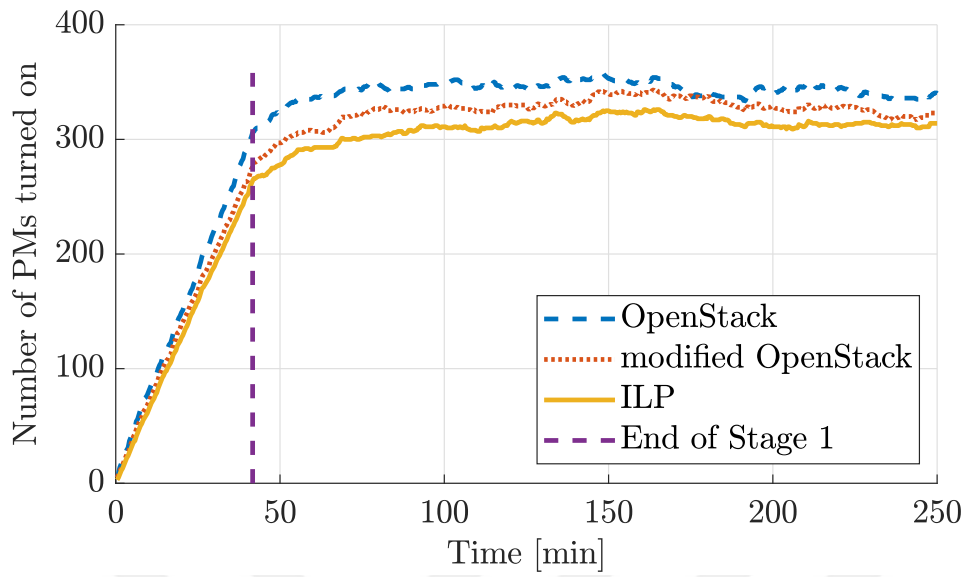


Figure 5.1: Experiment 1: Number of PMs that are turned on over time.

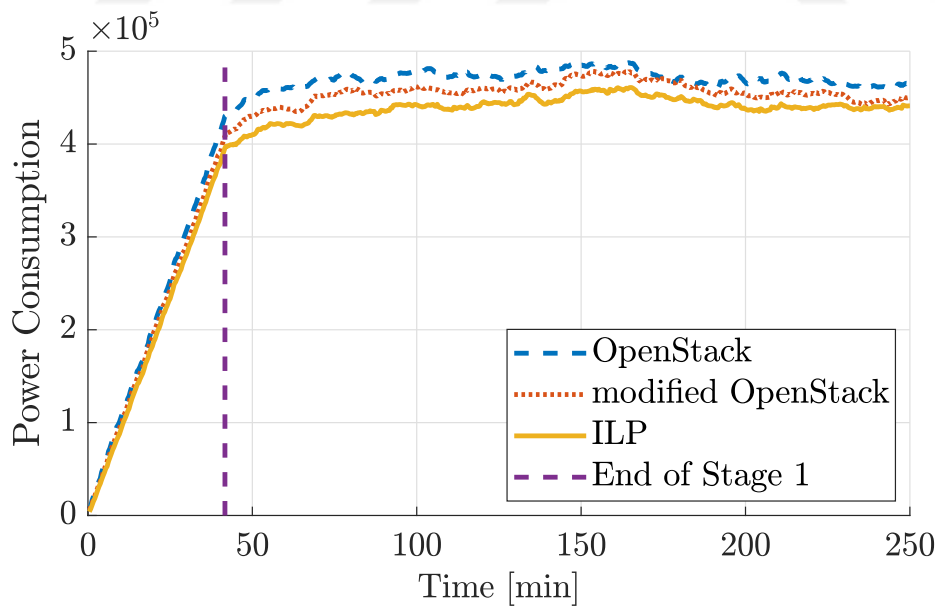


Figure 5.2: Experiment 1: Power consumption over time.

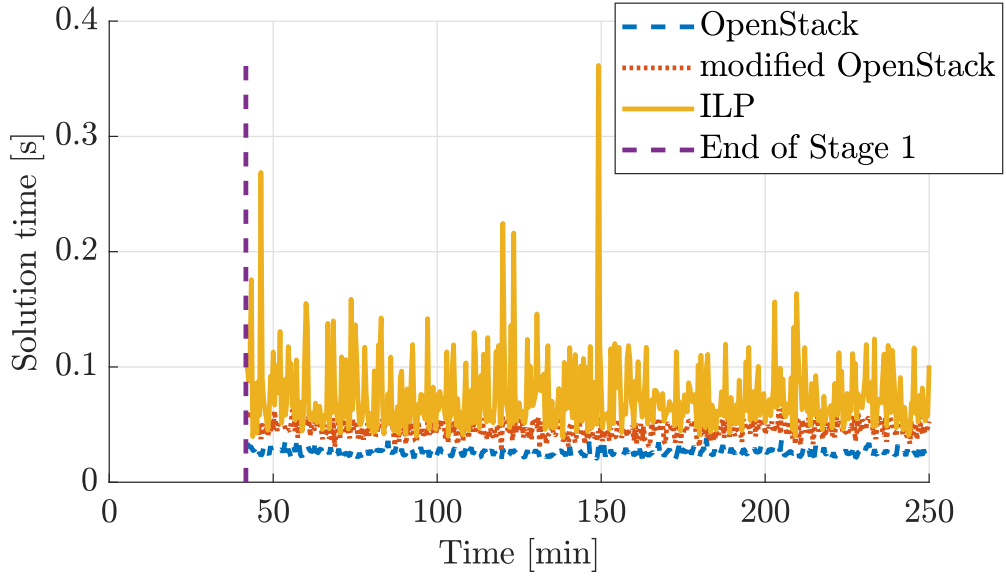


Figure 5.3: Experiment 1: Run-time of the resource allocation computation.

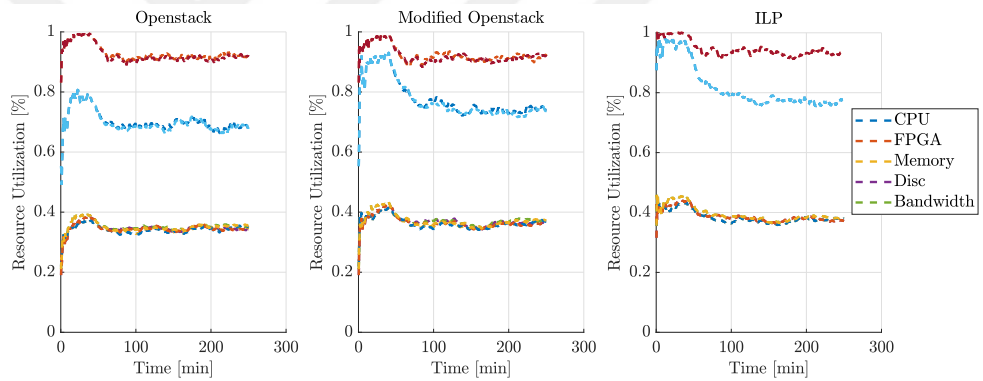


Figure 5.4: Experiment 1: Resource utilization of the different methods.

to note that all the methods achieve a high utilization of one dominating resource. In this experiment, this observation is meaningful since, on average, an equal number of requests with a similar distribution of resources enter and leave the CDC.

5.1.2 Experiment 2: Dependency on the Number of Requests

One parameter that affects the computation time and performance of ACLOUD-MAN is the number of requests M in one group. On the one hand, a larger value of M increases the number of decision variables of the ILP in Section 3.2 and hence leads to

an increased computation time. On the other hand, a smaller value of M is expected to have a negative effect on the performance (number of PMs, power consumption, resource utilization) of ACCLLOUD-MAN. In order to investigate this effect, we perform the same simulation experiments as in Section 5.1.1 with different values of $M = 5$, $M = 7$ and $M = 10$. Exemplary results are shown in Fig. 5.5 to 5.7.

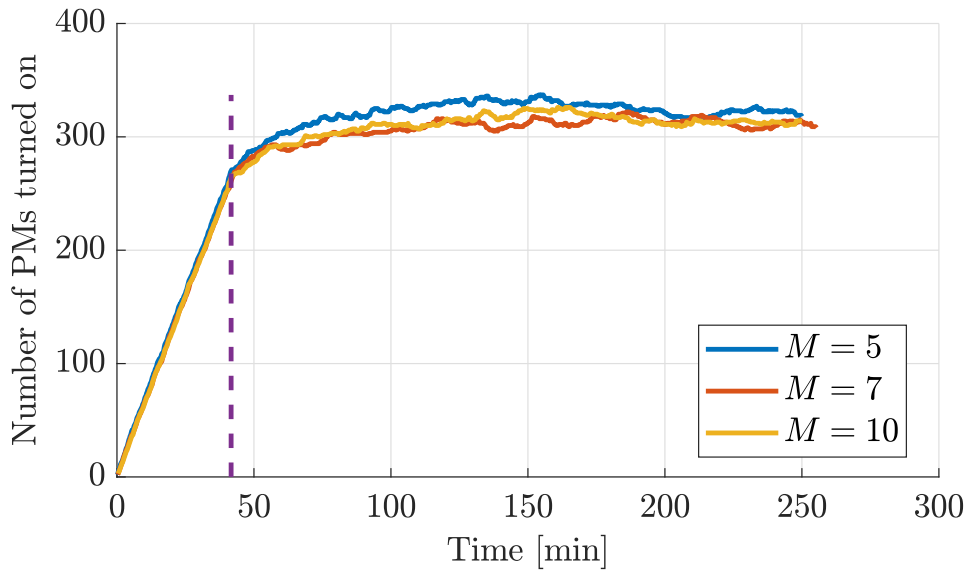


Figure 5.5: Experiment 2: Number of PMs that are turned on over time.

The interesting observation from Fig. 5.5 and 5.6 is that the performance for $M = 7$ is slightly better than that for $M = 10$. This indicates that it is actually not necessary to include a large number of requests in one group in order to achieve a good performance of ACCLLOUD-MAN.

The solver’s computation time for different grouping sizes is shown in Fig. 5.7. The computation time strongly depends on the grouping size. For sizes of 5 and 7, it takes less than 0.1 s to compute an allocation. For a grouping size of 10, the computation takes less than 0.2s for all but 5 cases. None of these 5 exceptional cases exceed 0.4s. Comparing to the time it takes to boot up a typical VM, the worst case of 0.4s is small enough not to hinder the cloud’s operation.

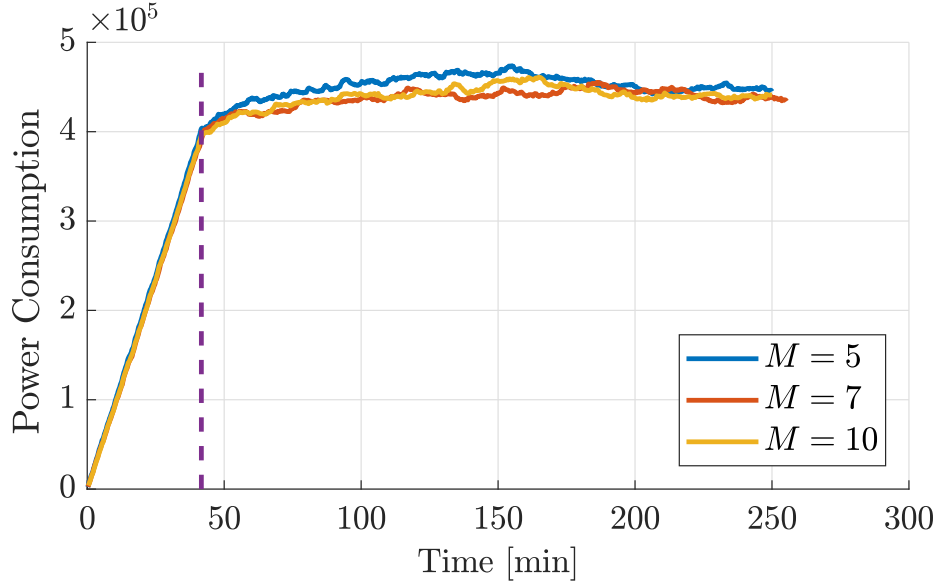


Figure 5.6: Experiment 2: Power consumption over time.

5.1.3 Experiment 3: Dependency on the Size of the CDC

Another interesting parameter is the number of available PMs in a CDC compared to the number of currently used PMs. We next perform an experiment where about 300 PMs are in use, whereas the number of PMs in the CDC is $N = 400$, $N = 1000$ and $N = 4000$. Regarding the other parameters, $M = 10$ and $\gamma = 2$ is used in this experiment. The simulation results are shown in Fig. 5.8 to 5.10.

It can be seen from Fig. 5.8 that N has a small effect on the number of PMs turned on. However, a considerable effect on the power consumption is observed from Fig. 5.9. This result can be explained as follows. In case of a small number of available PMs ($N = 400$), the PMs to be chosen for resource allocation might not fit perfectly for the incoming requests. In case of a large number of available PMs ($N = 1000$ and $N = 4000$), it is more likely to find PMs that fit the incoming requests. That is, although a similar number of PMs is needed, the PMs are more suitable for the incoming request if N is larger, leading to a lower power consumption. There is no notable difference in the computation times depending on N as can be seen from Fig. 5.10.

Table 5.1 shows the average values for the performance parameters for ACLOUD-

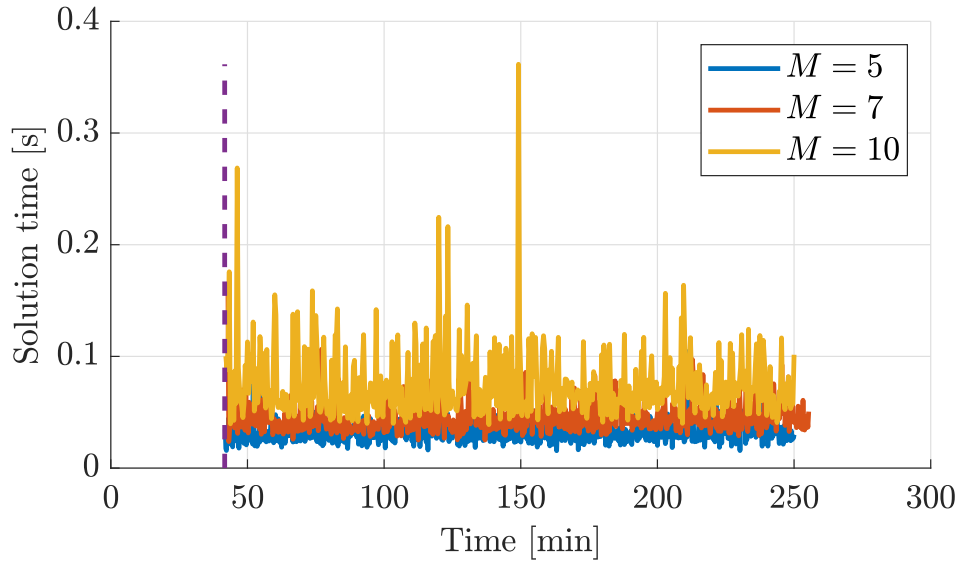


Figure 5.7: Experiment 2: Run-time of the resource allocation computation.

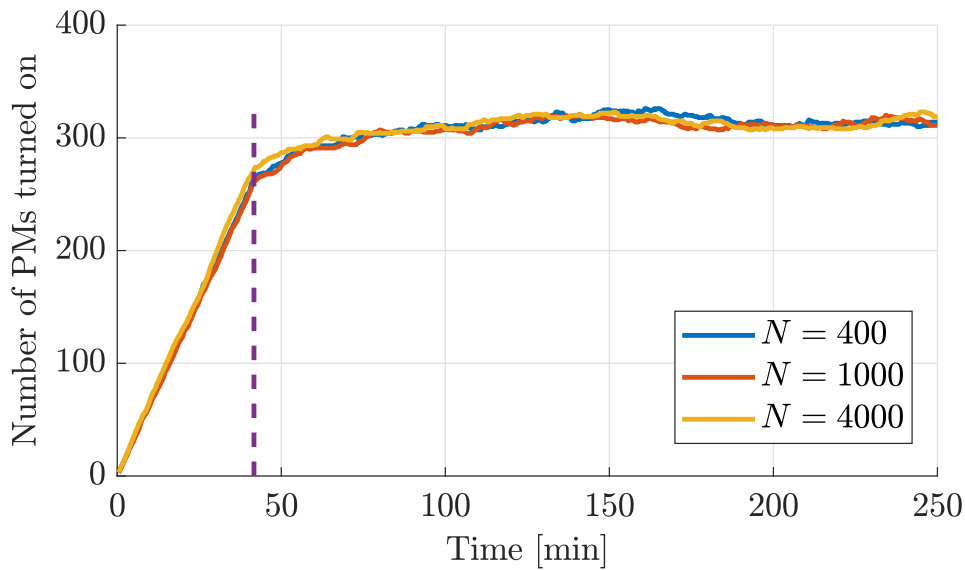


Figure 5.8: Experiment 3: Number of PMs that are turned on over time.

MAN and the modified OpenStack resource allocation. All the values are normalized with respect to the value of the ILP solution for $N = 4000$. The values in the table confirm that a larger number of PMs offers more options for the resource allocation and hence leads to a better performance for both the Modified OpenStack and ACCLOUD-MAN. Hereby, it is interesting to note that the run-time of the Modified OpenStack method increases considerably with the larger number of PMs.

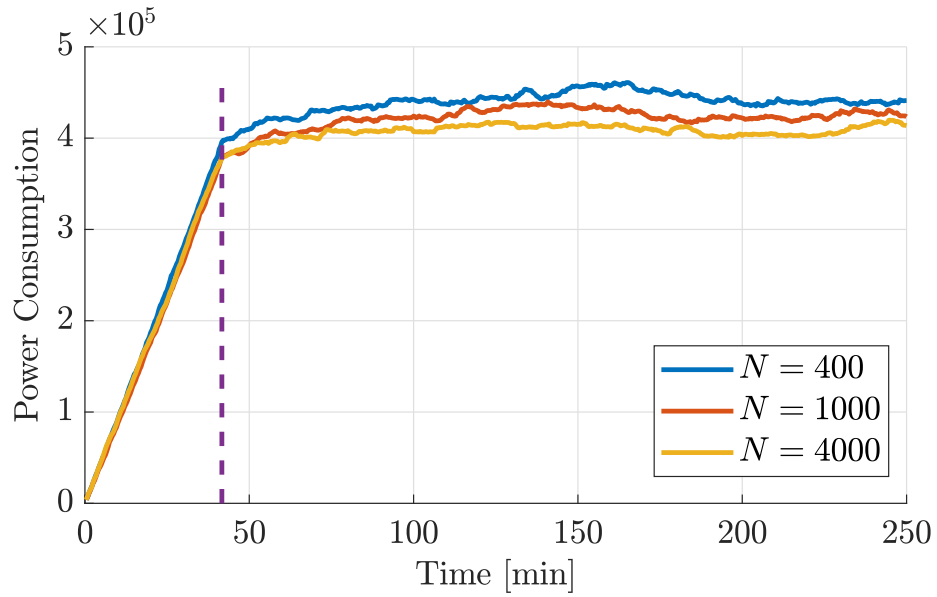


Figure 5.9: Experiment 3: Power consumption over time.

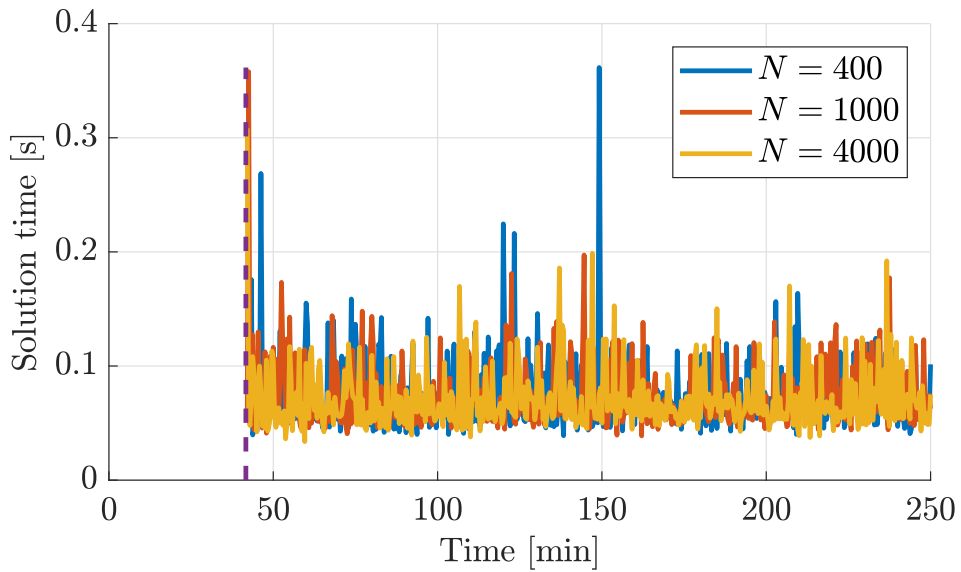


Figure 5.10: Experiment 3: Run-time of the resource allocation computation.

5.1.4 Experiment 4: Dependency on the Number of Candidate PMs

We finally evaluate the dependency of ACCLOUD-MAN performance on the parameter γ that determines the number of additional PMs included in the ILP. We perform simulations for $N = 400$, $M = 10$ and $\gamma = 1, 1.5, 2, 2, 2.5$. The simulation results

Table 5.1: Comparison of average number of on PMs and average power consumption for different numbers of PMs.

	on PM		Power		Time	
	mod. OS	ILP	mod. OS	ILP	mod. OS	ILP
$N = 400$	1.05	1.0	1.12	1.08	0.65	1.06
$N = 1000$	1.04	0.99	1.06	1.03	3.14	1.03
$N = 4000$	1.05	1.0	1.03	1.0	15.84	1.0

Table 5.2: Comparison of average number of on PMs, average power consumption and average time spent for decision for different values of γ .

	$\gamma = 1$	$\gamma = 1.5$	$\gamma = 2$	$\gamma = 2.5$	$\gamma = 3$
on PM	1.02	1.02	1.01	1.01	1.0
Power	1.01	1.01	1.0	1.01	1.0
Time	0.64	0.73	0.81	0.90	1.0

are shown in Fig. 5.11 to 5.13.

The interesting observation in this case is that the difference between the choices of γ is small. That is, offering a small number of additional power efficient PMs for solving the formulated ILP is sufficient to achieve a good performance of ACCLOUD-MAN. This result is confirmed by the values in Table 5.2. Note that the values are normalized with respect to the last column.

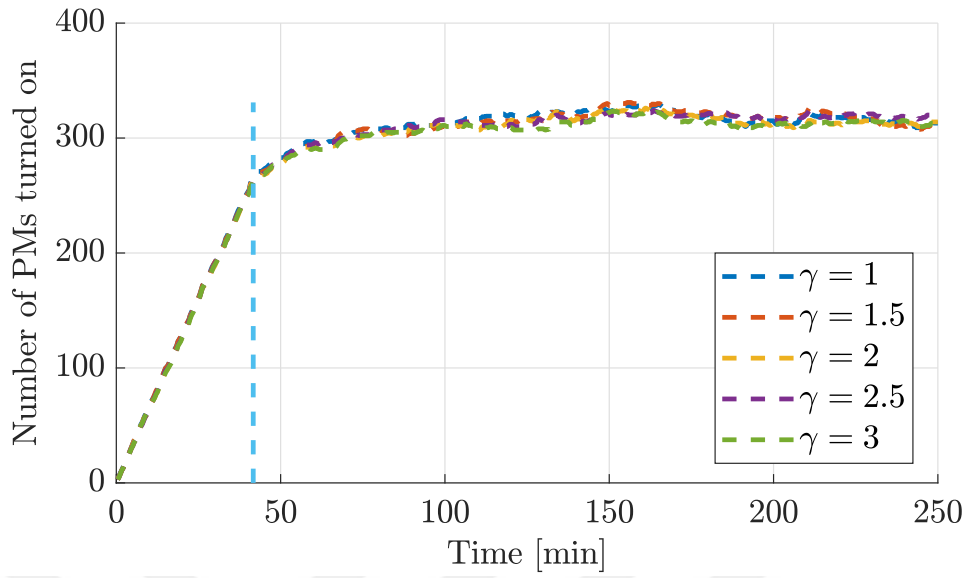


Figure 5.11: Experiment 4: Number of PMs that are turned on over time.

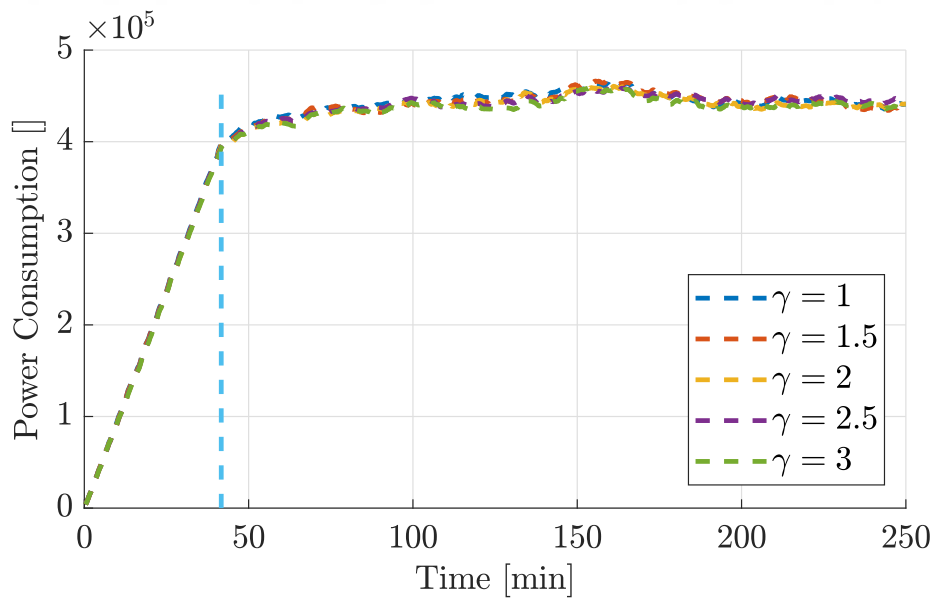


Figure 5.12: Experiment 4: Power consumption over time.

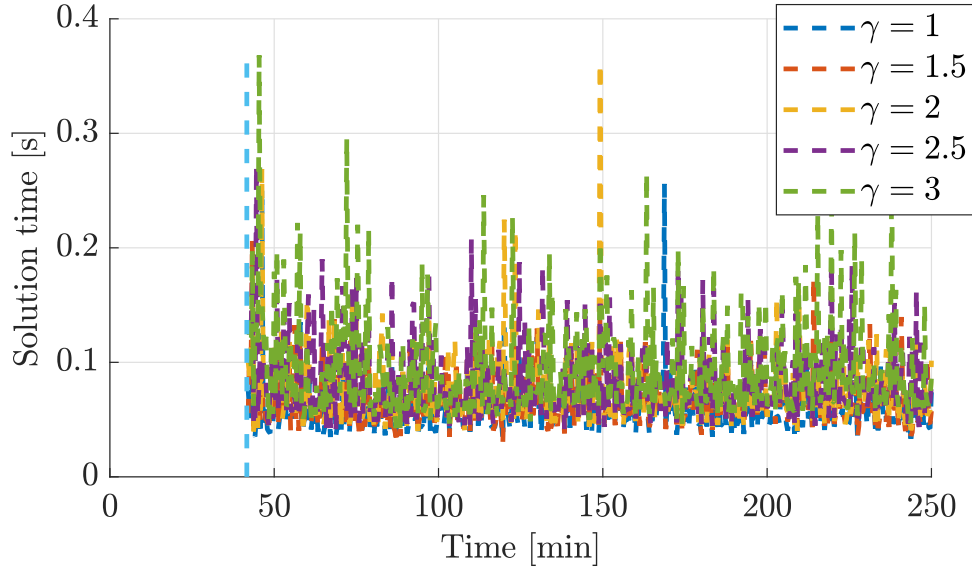


Figure 5.13: Experiment 4: Run-time of the resource allocation computation.

5.2 Realistic Request Scenario

5.2.1 Experiment 5: Comparison of Resource Allocation Methods

In this experiment, we compare the performance of OpenStack and the proposed ILP solution in the steady-state of the CDC using PMs with a number of CPU cores in the interval $[64, 128]$. $(P_{on,i})$, $(P_{CPU,i})$ and $(P_{FPGA,i})$ are in the intervals $[90, 100]W$, $[3, 5]W$ and $[1, 2]W$ respectively. $\gamma = 2$ and $M = 10$ are used as ILP parameters and number of PMs in the CDC is 250. Choice of γ is shown to not have a great effect for performance in Section 5.1.4; therefore the median value in the previously considered options is chosen. M is chosen as the largest value in the previously considered options, as its effect is observed to be significant in Section 5.1.2. Obtained simulation results are depicted in Fig. 5.14 to 5.23.

Fig. 5.14 to 5.18 further indicate that the ILP solution is more power and resource efficient already during the start-up period.

Inspecting Fig. 5.20, a clear reduction of the used power is observed when using the proposed ILP solution. On average, the reduction amounts to more than 16 % as presented in Table 5.3 and is mostly due to the fact that much fewer PMs are turned on

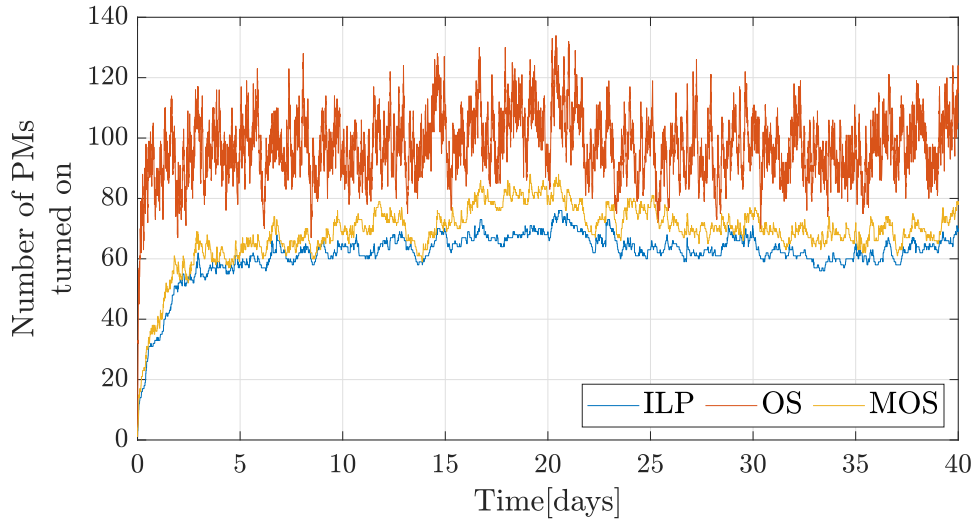


Figure 5.14: Experiment 5: Number of PMs turned on in the start phase.

and use power. The decrease in the number of used PMs goes along with an increase in the CPU utilization and an almost full FPGA utilization when using the proposed ILP solution as can be seen in Fig. 5.21 and 5.22. Likewise, the Memory, Disc and Bandwidth usage of the ILP solution is superior to the other methods (Fig. 5.23). Together, the same workload can be handled with less than half the number of PMs at a reduced power when using proposed ILP method as seen in Fig. 5.19 and Table 5.3.

Another observation from Fig. 5.19 and 5.20 is, that variance in the CDCs power consumption and turned on PM count is greatly reduced with the ILP solution; resulting in a more stable CDC.

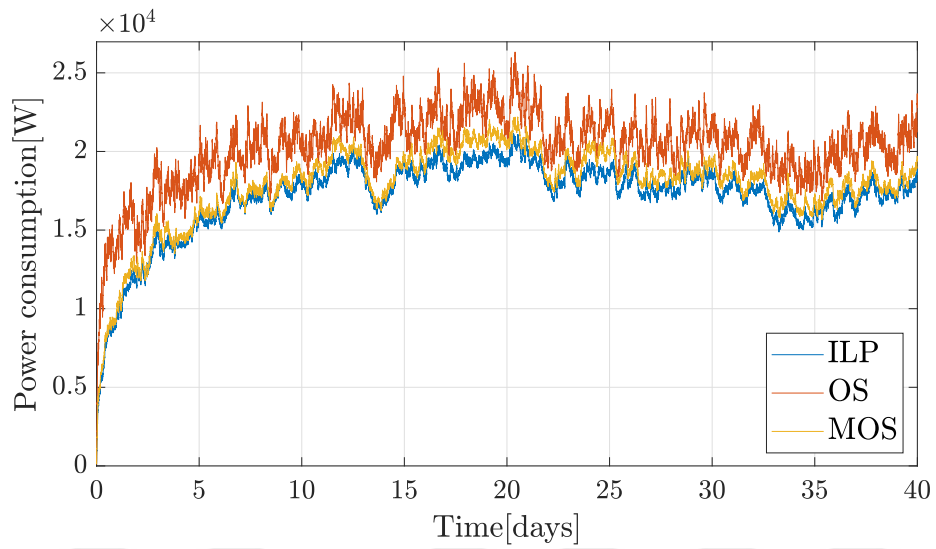


Figure 5.15: Experiment 5: Power consumption of CDC in the start phase.

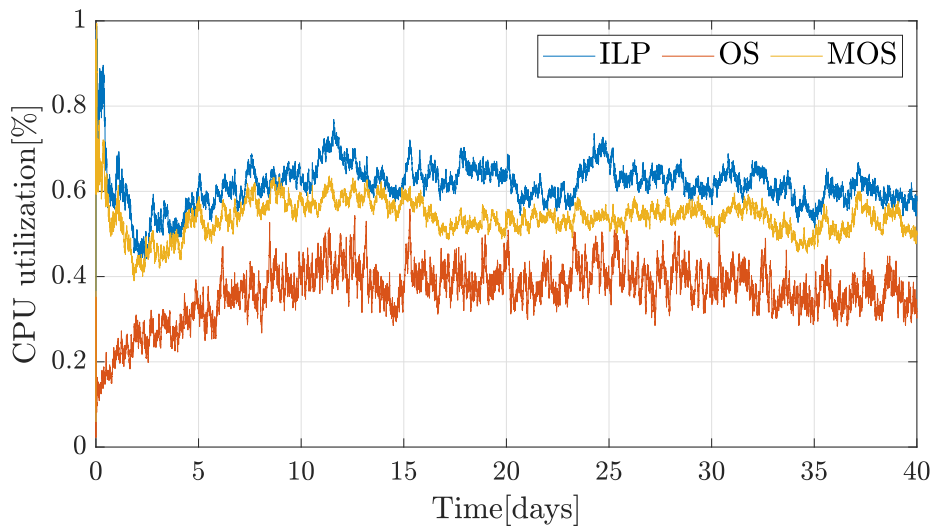


Figure 5.16: Experiment 5: CPU utilization of CDC in the start phase.

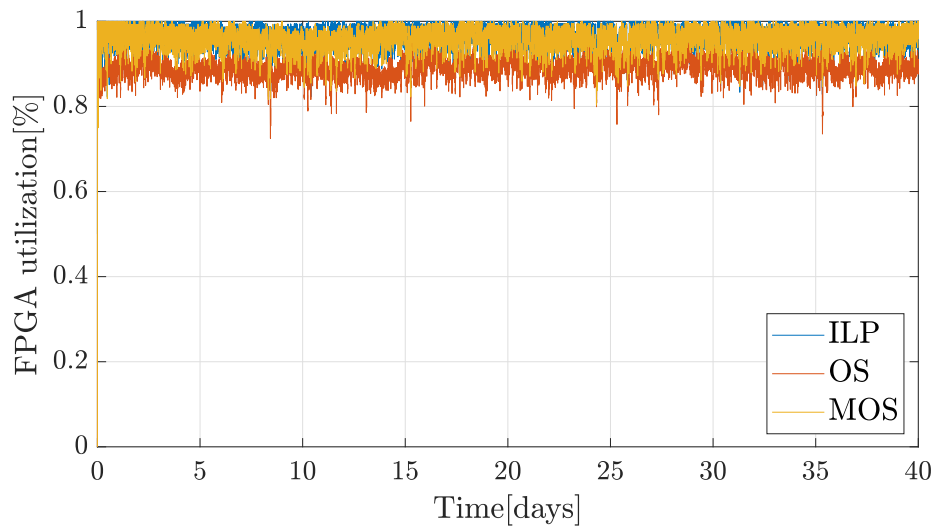


Figure 5.17: Experiment 5: FPGA utilization of CDC in the start phase.

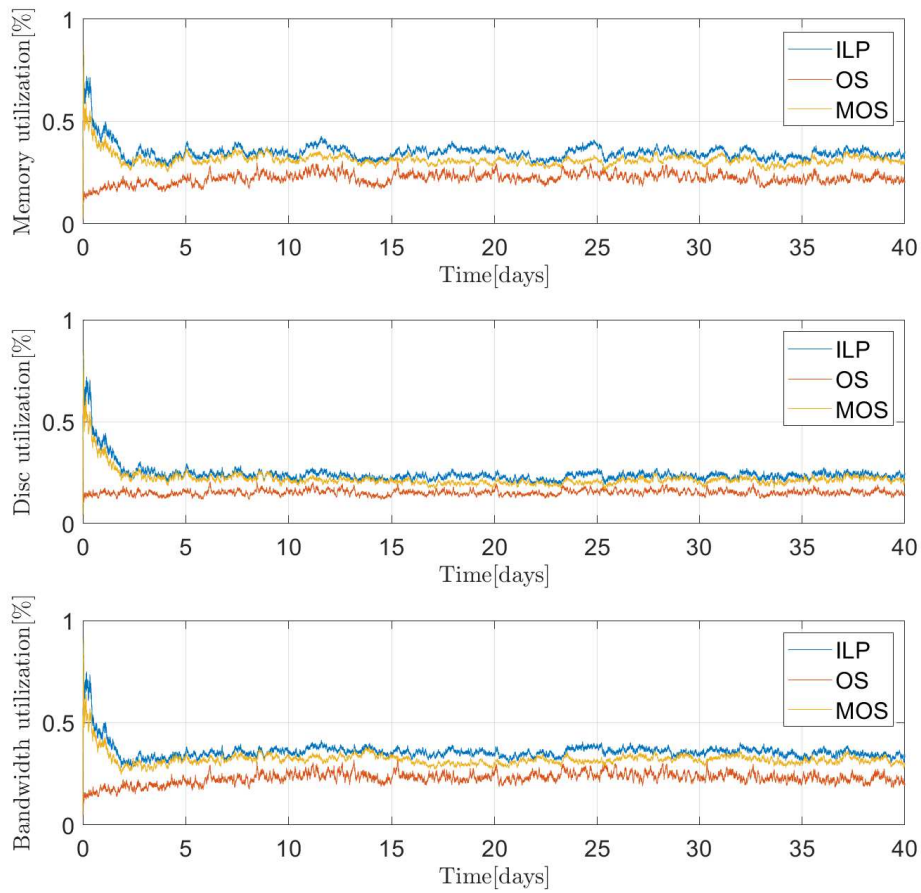


Figure 5.18: Experiment 5: Peripheral utilization of CDC in the start phase.

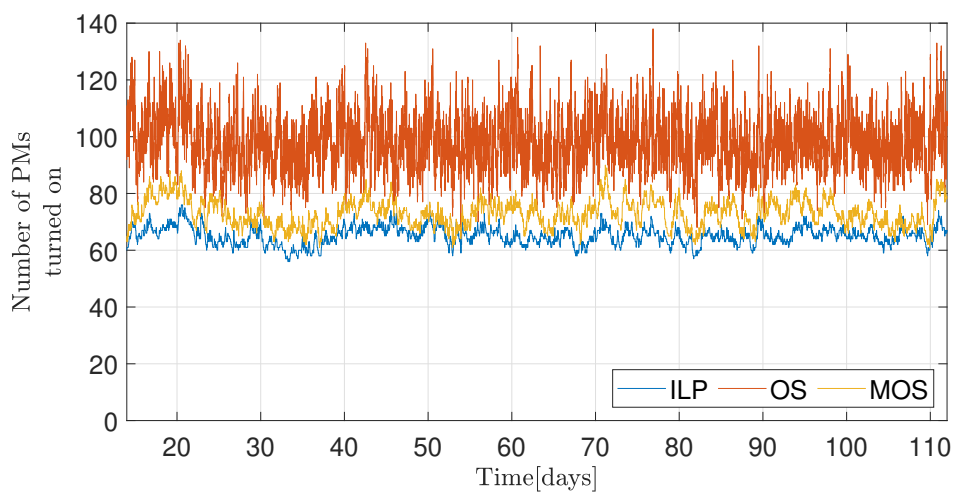


Figure 5.19: Experiment 5: Number of PMs turned on in the steady state.

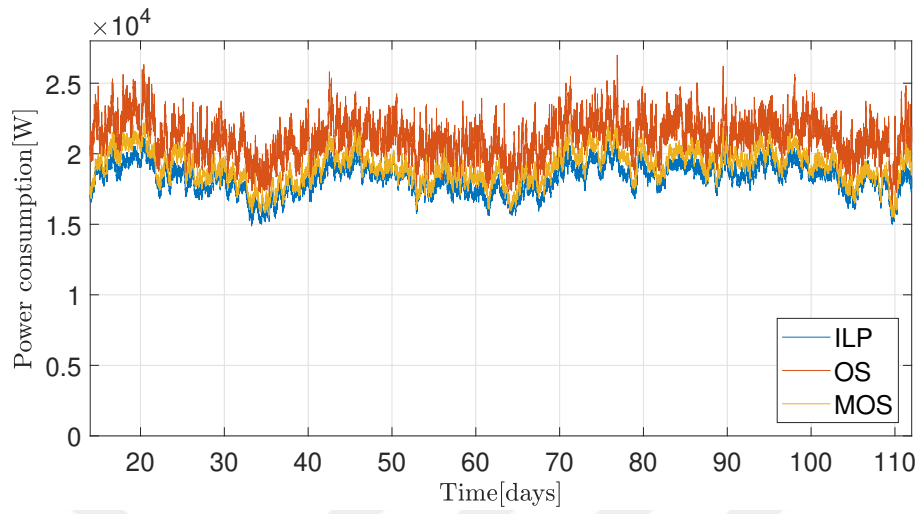


Figure 5.20: Experiment 5: Power consumption of CDC in the steady state.

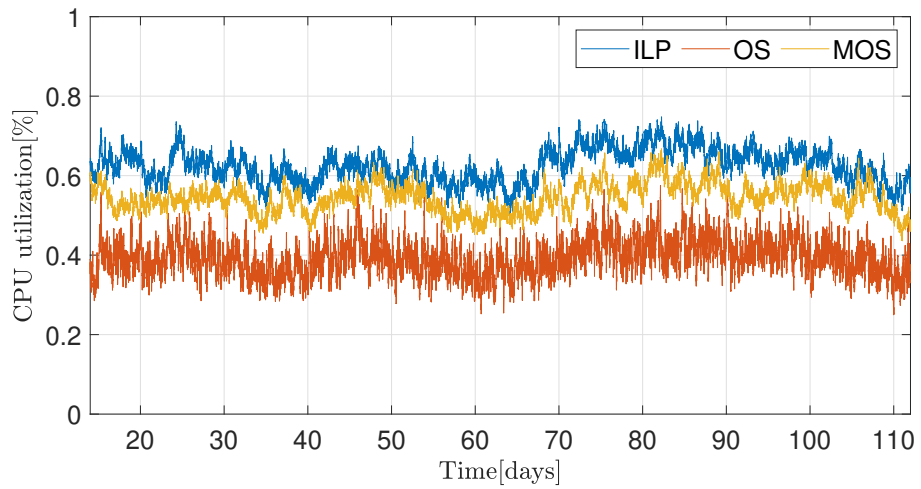


Figure 5.21: Experiment 5: CPU utilization of CDC in the steady state.

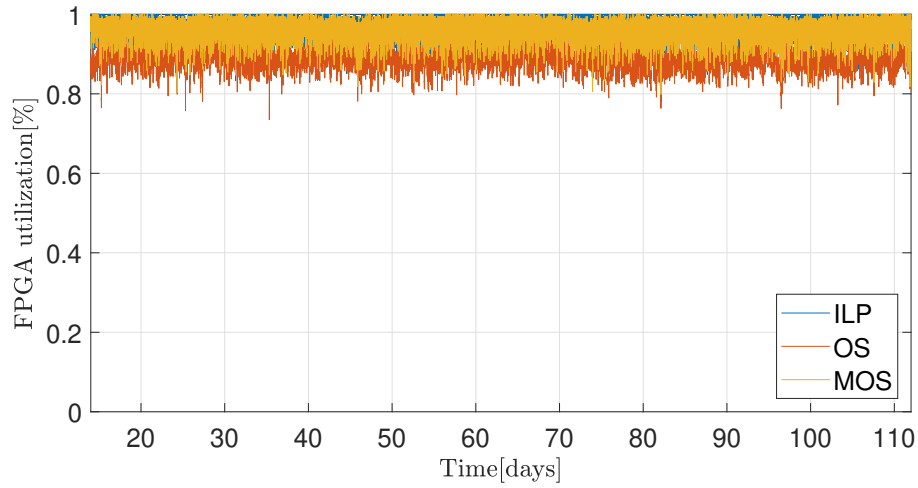


Figure 5.22: Experiment 5: FPGA utilization of CDC in the steady state.

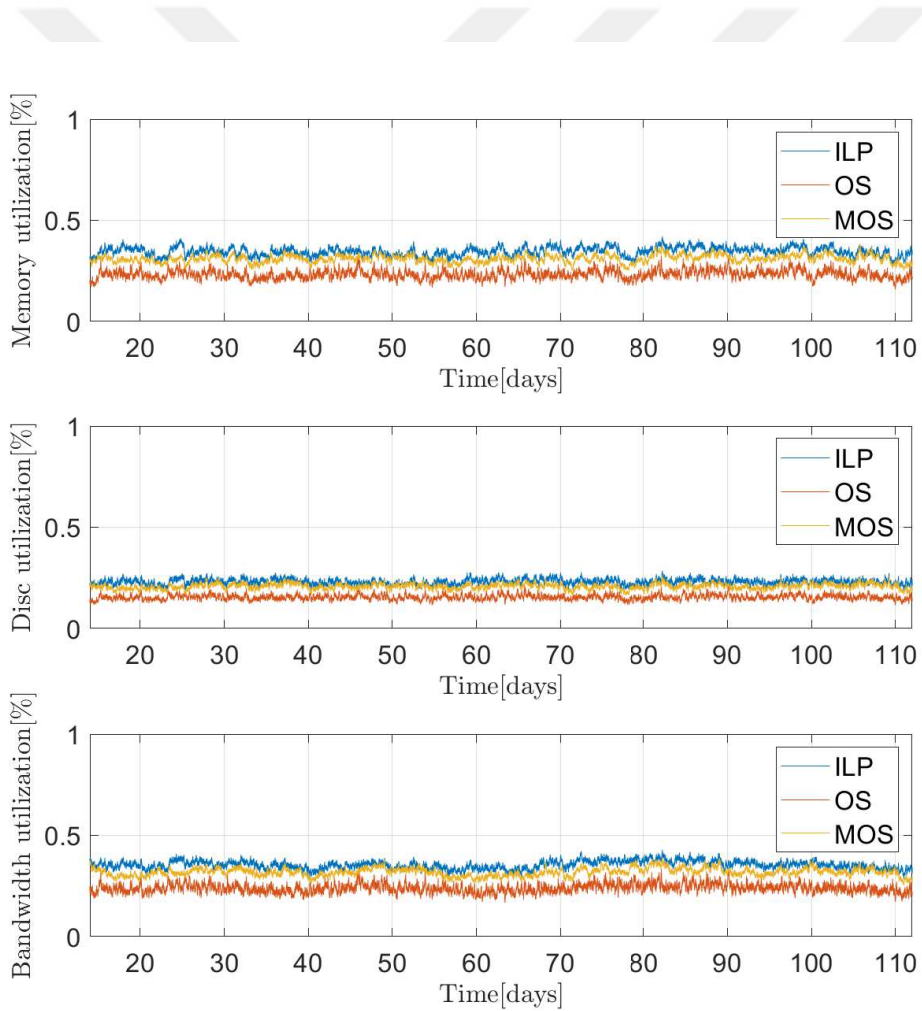


Figure 5.23: Experiment 5: Peripheral utilization of CDC in the steady state.

5.2.2 Experiment 6: Dependency on the Number of CPU cores

We finally focus on the dependency of the power reduction on the properties of PMs in the CDC. To this end, we conduct three simulation runs for CDCs with different numbers of CPU cores per PM as follows:

1. Number of CPU cores in the interval [32,64].
2. Number of CPU cores in the interval [64,128].
3. Number of CPU cores in the interval [128,256].

All other parameters are as they are in the Experiment 5 presented in 5.2.1.

The obtained results with the same workload as in the previous sections are shown in Fig. 5.24 to 5.27 (note that the results for Case 2 are already available in Fig. 5.19 and 5.20).

The important observation of this experiment is that a reduction in the power consumption is only achieved if a sufficient number of CPU cores is available per PM. Fig. 5.24 and 5.25 shows a case with a small number of CPU cores. Here, no significant difference of the different resource allocation methods is observed. Nonetheless, Fig. 5.20 and Fig. 5.27 indicate that an increase in the number of CPU cores per PM can lead to a significant reduction in the power consumption when using the proposed ILP solution. This reduction amounts to about 16% in Case 2 and 20% in Case 3 as presented in Table 5.3. It is further noted that also the MOS method achieves a significant reduction in the power consumption if a large number of CPU cores is available (5.27).

Table 5.4 shows the average time spent for each decision. Average times for decision lie in the negligible interval of [2.4, 29.1]ms and as such do not interfere with the CDC's operations. ILP decisions take approximately 10 times longer than other approaches; however, it should be noted that ILP decision is actually made for 10 requests at a time. Therefore per request time for decisions are in the same order for all algorithms.

In summary, this experiment shows the effect of offering alternatives for serving SaaS

Table 5.3: Experiment 5 and 6: Comparison of percentage improvements of average number of on PMs and average power consumption over OpenStack for different numbers of CPU cores.

	on PM		Power	
	mod. OS	ILP	mod. OS	ILP
$CPU = [32, 64]$	15.22%	14.97%	6.88%	3.49%
$CPU = [64, 128]$	36.43%	52.27%	11.76%	16.34%
$CPU = [128, 256]$	62.25%	68.14%	18.29%	19.87%

Table 5.4: Experiment 5 and 6: Comparison of average time for allocation decision of all algorithms in milliseconds.

	OS	mod. OS	ILP
$CPU = [32, 64]$	2.4	3.8	28.4
$CPU = [64, 128]$	2.8	4.2	27.3
$CPU = [128, 256]$	3.2	4.4	29.1

requests. Hereby, we emphasize that the selected alternatives for this experiment comprise FPGA and CPU resources. FPGA resources are scarce but more power efficient, whereas CPU resources are abundant but with a higher power consumption. Since OpenStack cannot handle alternatives, it will always pick the power-efficient FPGA resource for such SaaS requests. If necessary, new PMs must be turned on even they are not much utilized. On the other hand, the proposed MOS and ILP methods account for alternatives and avoid turning on new PMs if available CPU resources can be used instead of FPGA resources. Since FPGAs are scarce, this effect becomes more evident if PMs with more CPU cores are available in the CDC as can be readily observed from Fig. 5.21. In addition, the MOS method performs well if there is a large number of CPU cores since alternatives without FPGA usage are always available due to a low CPU usage as can be seen in Fig. 5.21.

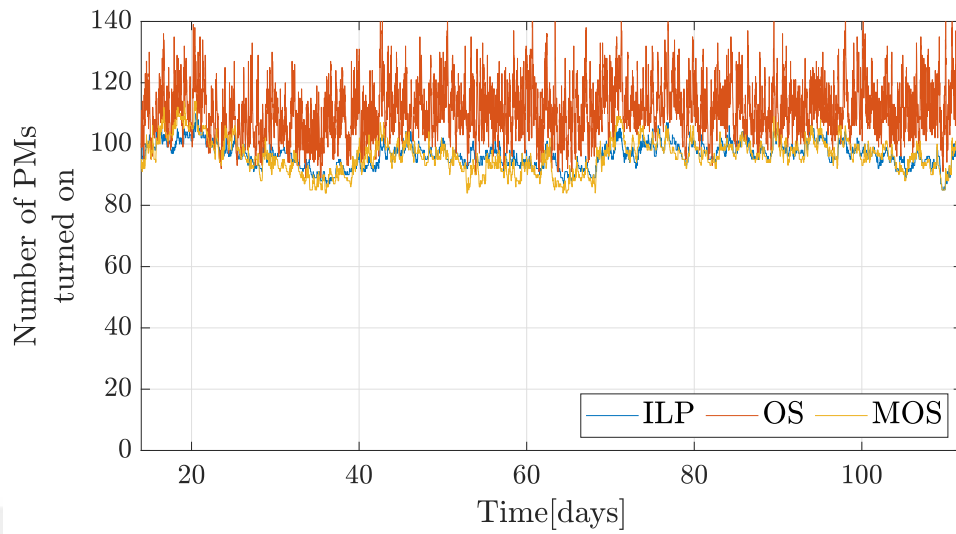


Figure 5.24: Experiment 6: Number of PMs turned on in the steady state for [32,64] CPU cores (Case 1).

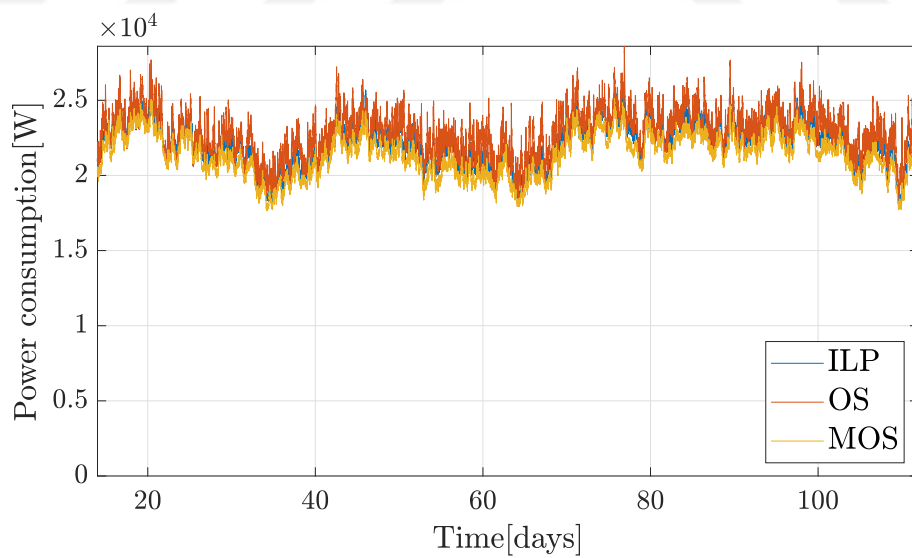


Figure 5.25: Experiment 6: Power consumption of CDC in the steady state for [32,64] CPU cores (Case 1).

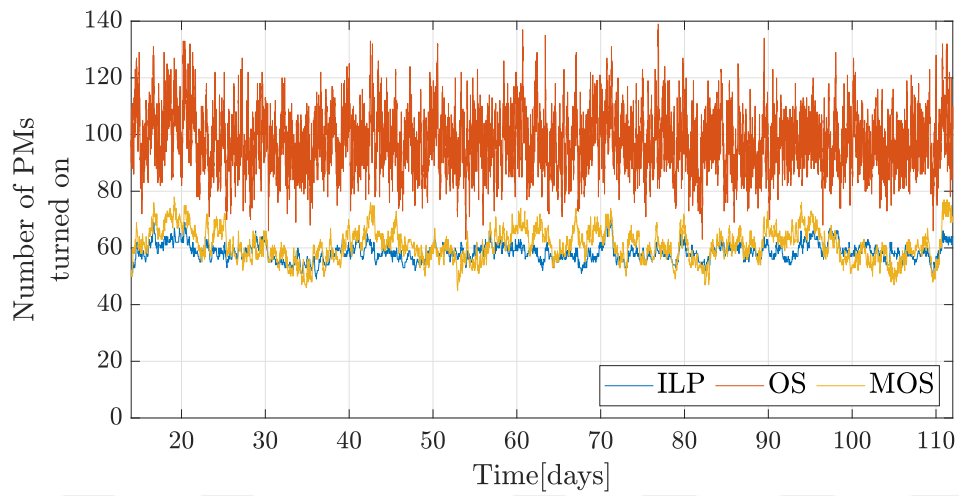


Figure 5.26: Experiment 6: Number of PMs turned on in the steady state for [128,256] CPU cores (Case 3).

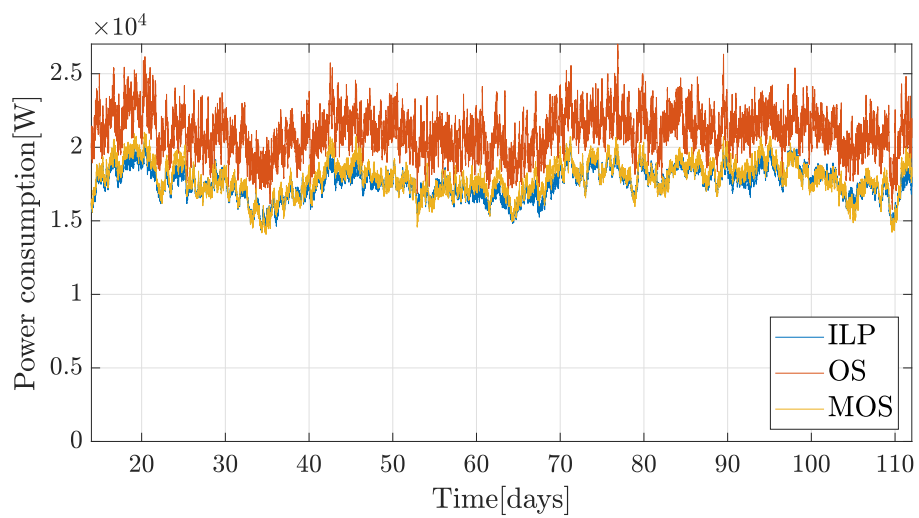


Figure 5.27: Experiment 6: Power consumption of CDC in the steady state for [128,256] CPU cores (Case 3).

5.2.3 Experiment 7: Dependency to the Power Parameters

In this experiment, power parameters are set such as $(P_{on,i})$, $(P_{CPU,i})$ and $(P_{FPGA,i})$ are in the intervals $[8, 12]$ W/core, $[9, 11]$ W and $[1, 2]$ W respectively [39]. Used numbers of CPU cores are in the intervals of $[32, 64]$, $[64, 128]$ and $[128, 256]$. In this experiment, $P_{on,i}$ is defined per CPU core; thus $P_{on,i}$ will be higher for PMs with more CPU cores. For the case where number of CPU cores are in the interval $[64, 128]$, $P_{on,i}$ is in the interval $[512, 1536]$ W, whereas in the previous experiment is within $[90, 100]$ W. The obtained results with the same workload as in the previous sections are shown in Fig. 5.28 to 5.33 and Table 5.5.

Results in Table 5.5 show, to achieve a good performance a higher number of CPU cores are needed as in the previous experiment. Comparing Table 5.3 and Table 5.5 shows percentage improvement of ACCLOUD-MAN is higher for this experiment. This difference is due to the increase in the average $P_{on,i}$. As the $P_{on,i}$ is higher, powering on more PMs result in a greater power consumption. This effect becomes more prominent for cases with higher number of CPU cores, since $P_{on,i}$ is greater for these cases.

Another thing to note from the results presented in Table 5.3 and Table 5.5 is the reduction in the improvement of number of on PMs. Proposed method even uses more PMs in the case where number of CPU cores is in the interval $[32, 64]$. This is a direct result of PM's idle power being proportional to number of CPU cores. Since the idle power is proportional to the number of CPU cores, as long as the CPU utilization is high, choosing between PMs with a high number of CPU cores and low number of CPU cores does not make a difference. On the contrary, in the previous experiment where PM's idle power does not depend on the number of CPU cores, it is beneficial to choose PMs with high number of CPU cores. This is because a higher number of requests can be allocated to the PM and idle power is the same as a PM with a lower number of CPU cores. Since formulation optimizes only power, in this experiment proposed method does not have a preference; however, in previous experiment proposed method heavily prefers larger PMs. Since larger PMs can accommodate more requests, this causes the proposed method to use more PMs for the premise defined in this experiment. It should be noted that even though percentage improvement is

Table 5.5: Experiment 7: Comparison of percentage improvements of average number of on PMs and average power consumption over OpenStack for different numbers of CPU cores.

	on PM		Power	
	mod. OS	ILP	mod. OS	ILP
$CPU = [32, 64]$	-0.09%	-0.22%	4.83%	4.44%
$CPU = [64, 128]$	17.32%	21.72%	18.04%	21.49%
$CPU = [128, 256]$	30.72%	36.62%	32.36%	37.04%

decreased, for the cases where number of CPU cores is in other intervals than $[32, 64]$, proposed method uses fewer PMs than OpenStack.

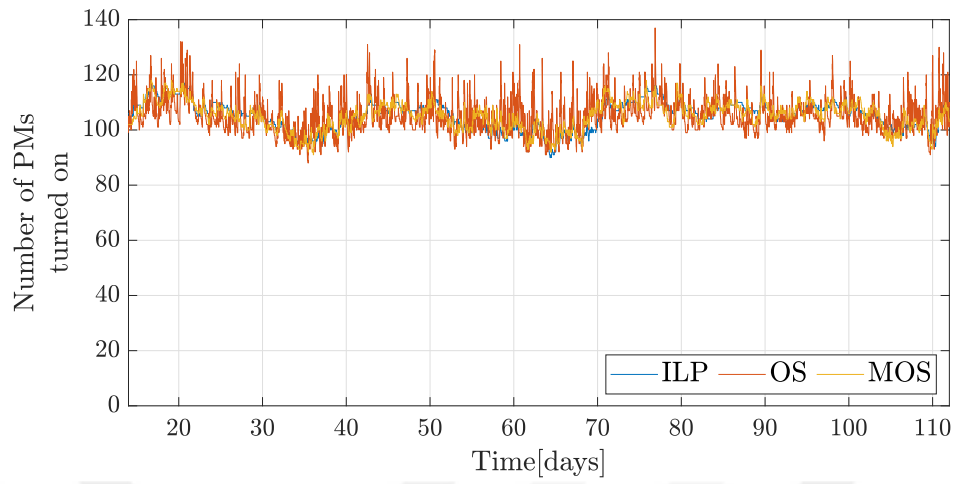


Figure 5.28: Experiment 7: Number of PMs turned on in the steady state for [32,64] CPU cores (Case 1).

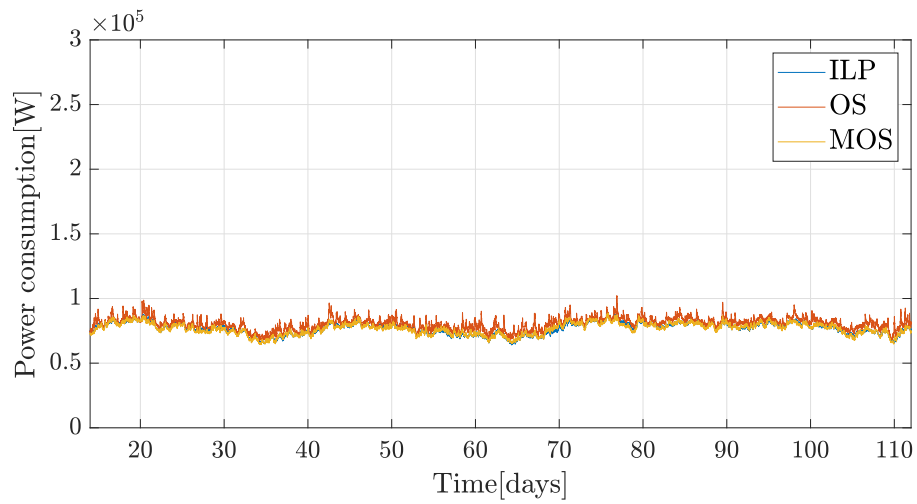


Figure 5.29: Experiment 7: Power consumption of CDC in the steady state for [32,64] CPU cores (Case 1).

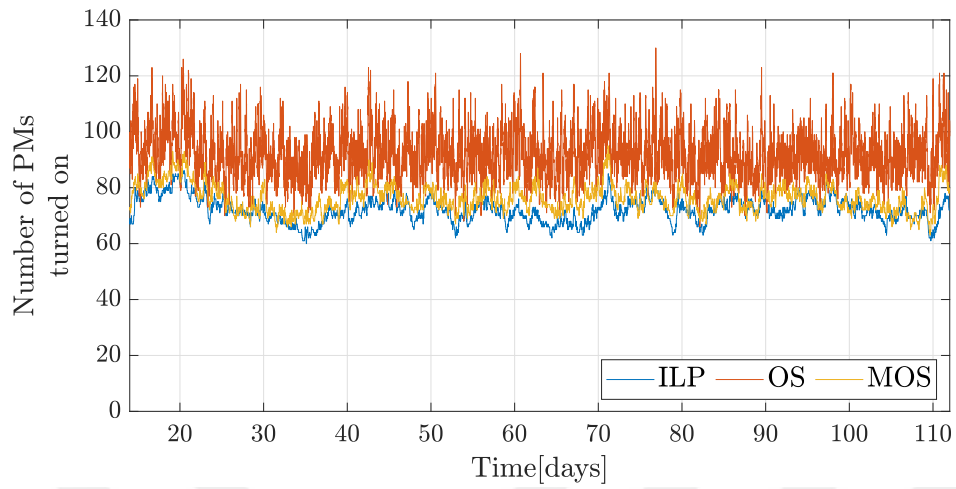


Figure 5.30: Experiment 7: Number of PMs turned on in the steady state for [64,128] CPU cores (Case 2).

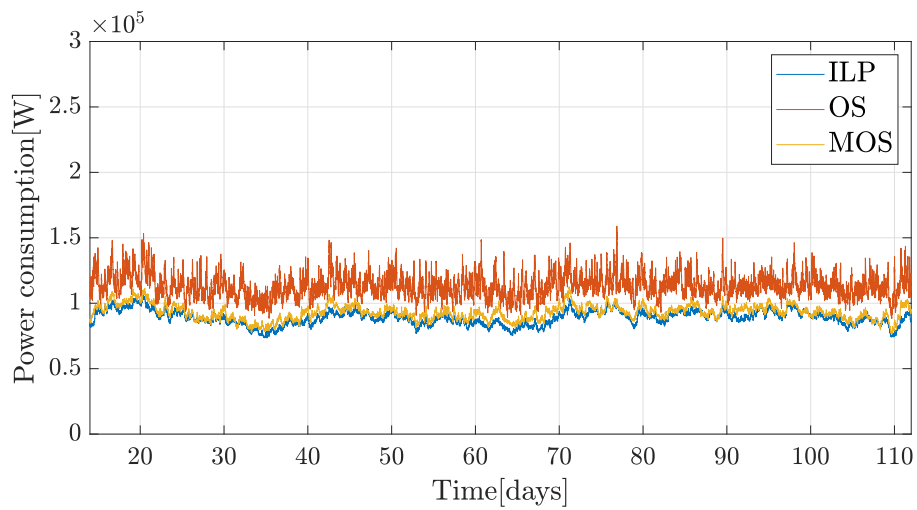


Figure 5.31: Experiment 7: Power consumption of CDC in the steady state for [64,128] CPU cores (Case 2).

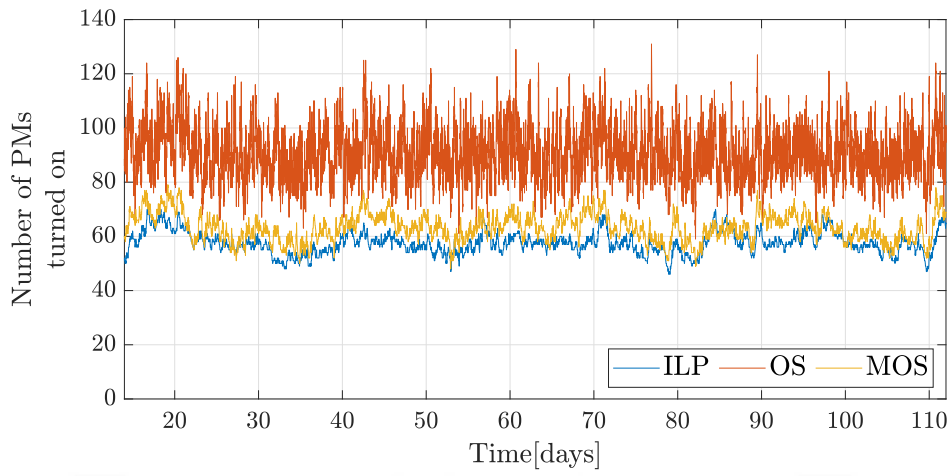


Figure 5.32: Experiment 7: Number of PMs turned on in the steady state for [128,256] CPU cores (Case 3).

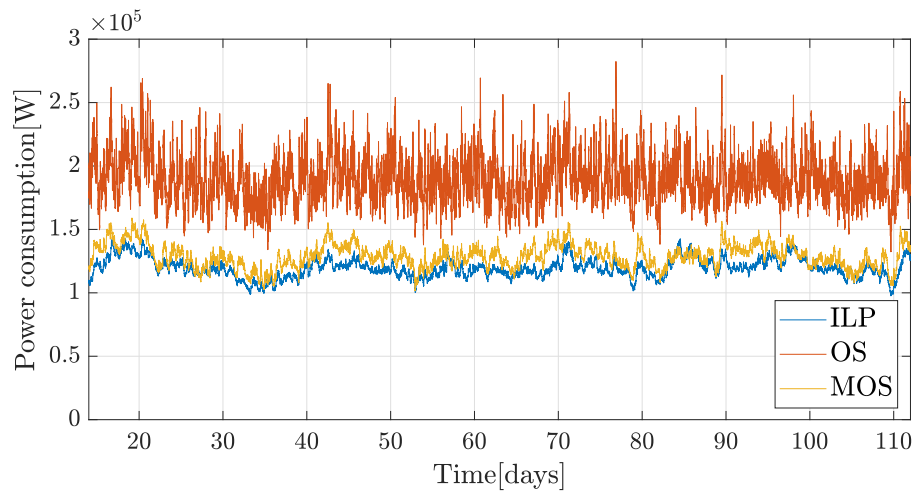


Figure 5.33: Experiment 7: Power consumption of CDC in the steady state for [128,256] CPU cores (Case 3).



CHAPTER 6

FURTHER DISCUSSIONS

6.1 Generalization of the Model to Any Number and Type of CDC Resources

Formulated optimization problem is general in the sense that new/different resources can be added in a straightforward way. If a new resource contributes to the power consumption, then an additional term needs to be included in (3.2.2). Moreover, an additional resource constraint similar to (3.2.4) to (3.2.8) is needed.

6.2 Generalization of the Model to an Energy-Power Hybrid Optimization

For the scope of this work it is assumed that the time user spends in the CDC does not depend on the actions of the manager. This is the case for user-driven or stream based requests such as processing a live video. However for data-driven requests such as encrypting a file, manager's decision changes the completion time for the task. Encryption of a file with a given size may take 3 minutes on 1 CPU core, 1 minute on 1 FPGA region or 1.5 minutes on 3 CPU cores, all with different amounts of energy consumed. In addition to that, for these cases the service user gets is affected by the decision as user might get a faster or slower service.

Note that, this only holds for SaaS requests. For IPaaS requests, there is no way of knowing what user's intended task is or how long it might take. For SaaS requests fitting above description, CDC provider can profile offered software and can obtain the service time for each resource alternative. Let us denote the service time for j th request with k th resource alternative as $t_{j,k}$.

To optimize the energy consumption and user's service time, 2 new terms are defined:

$$E_{\text{comp}} = \sum_{ijk} (P_{\text{CPU},i} c_{j,k} + P_{\text{FPGA},i} f_{j,k}) \cdot t_{j,k} \cdot \bar{s}_{i,j,k} \quad (6.2.1)$$

$$T_{\text{delay}} = \sum_{ijk} \alpha_j \cdot t_{j,k} \cdot \bar{s}_{i,j,k} \quad (6.2.2)$$

E_{comp} is the amount of energy will be consumed for serving SaaS requests with time information. T_{delay} is a measure of time will be spent on those requests, where α_j is a constant weight. Larger the α_j is, larger j th requests service time's effect on T_{delay} term will be.

Finally, objective function is modified to minimize the energy consumption and time spent serving users, along with power consumption

$$\min F = P_{\text{newPM}} + P_{\text{comp}} + \beta E_{\text{comp}} + \alpha T_{\text{delay}}. \quad (6.2.3)$$

α and β are used to normalize terms with different orders. α , α_j and β can be tuned to choose which metric to optimize further, in case they require a trade off.

6.3 User Latency Implications of Request Bundling

Bundling of the user requests introduce an additional delay to CDC's reaction time to users. Added delay is proportional to inter-arrival time of user requests. For Azure service, published 2017 trace [44] has an average inter-arrival time of 0.56s. For a bundling size of 10, this would result in average 2.8s delay. To compare, other actions such as transferring VM image and initialization of VM takes between 100 and 1000 seconds and take a lot longer as presented in [45].

6.4 Extending the Algorithm to React to Other Cloud Events

Scope of this work is to allocate resources to incoming requests. However, this is not the only event occurring in a CDC. After its arrival, a user may change its resource

requirements. This is known as an "update" event. After some time spent in CDC, user leaves the system in an event named "departure". Apart from these, CDC may periodically trigger a "defragmentation" event; in which allocated resources in the CDC are relocated from other servers, in an effort to reduce resource fragmentation and number of on servers. Methods to handle events apart from arrival are discussed in the remainder of this section.

A user's resource demand may increase or decrease with an update event. In case of a decrease, no action is needed and user may use resources from the same server. If the resource demand has increased, but the server can accommodate the increase by allocating its free resources again no action is needed. However if demand has increased and server does not have enough free resources, user has to be relocated. On that case, user is removed from its current server and updated request is added to the pool of incoming requests; to be treated as a newly arrived user.

When a user departs from the CDC, it may leave an under utilized server wasting power. To handle this event proposed method can be used such as the following. Remaining users in the server in which the departure event has occurred are given to the proposed method as newly arrived. Servers of interest are limited to servers already on, excluding the one in which the departure event has occurred. If proposed method can obtain a solution (i.e. all users can fit in on servers) then users are migrated to servers decided by the proposed method and emptied server is turned off. Otherwise no action is taken.

For defragmentation event, a full defragmentation (i.e. complete reallocation of all users) is not possible firstly because such decision's complexity is not feasible for solving in a reasonable time limit. Secondly migrating all users is costly both power-wise and user performance-wise. Thus, this event can be handled in a similar fashion with the departure event. Each server is treated as if a virtual user has departed from it and undergoes the steps described in departure event. To minimize the number of migrations, servers must be treated in the order of increasing number of users residing on them.



CHAPTER 7

CONCLUSIONS

Cloud data centers use the concept of virtualization in order to provide resources on physical machines to users. In order to meet diverse user requests, available resources need to be allocated efficiently. This paper proposes a new cloud computing resource allocation model for heterogeneous cloud architectures with different computational resources and a corresponding resource manager ACLOUD-MAN (Accelerated CLOUD MANagement). The resource allocation model includes IaaS and PaaS request as well as SaaS business requests that can be met with multiple physical resource alternatives. The objective of the ACLOUD-MAN resource manager is to assign the IaaS/PaaS/SaaS request to physical machines (servers) with a minimum power consumption. The resource allocation problem is formulated as an integer linear programming (ILP) problem and solved using the CPLEX solver.

In order to evaluate the performance of ACLOUD-MAN, this work develops a simulation environment and performs several simulation experiments on two scenarios. First scenario offers randomly generated user requests. Other scenario include realistic user requests with different server power parameters.

In all cases, ACLOUD-MAN outperforms existing method by using less power (up to 37%) and fewer servers (up to 68%). This is achieved by favoring under utilized computational resources; therefore ACLOUD-MAN uses existing resources more efficiently. In addition, ACLOUD-MAN shows less variance in power consumption and number of powered on servers than existing method, offering a more stable CDC.

As servers' idle power consumption increase, utilization of resources become more critical, as low utilization leads to powering new servers on at a higher power expense.

Comparison of cases with different power parameters shows that as the servers' idle power consumption increase, ACCLOUD-MAN's benefits become more prominent.

The experiments further show that a good performance of ACCLOUD-MAN can be achieved for reduced versions of the formulated ILP that can be solved in less than 0.5 second. In future work, ACCLOUD-MAN will be implemented and tested in a laboratory scale cloud data center.



REFERENCES

- [1] A. D. JoSEP, R. KAtz, A. KonWinSKi, L. Gunho, D. PAttERSon, and A. RABKin, “A view of cloud computing,” *Communications of the ACM*, vol. 53, no. 4, 2010.
- [2] B. K. Rani, B. P. Rani, and A. V. Babu, “Cloud computing and inter-clouds–types, topologies and research issues,” *Procedia Computer Science*, vol. 50, pp. 24–29, 2015.
- [3] J. Tang and T. Q. Quek, “The role of cloud computing in content-centric mobile networking,” *IEEE Communications Magazine*, vol. 54, no. 8, pp. 52–59, 2016.
- [4] N. J. Kansal and I. Chana, “An empirical evaluation of energy-aware load balancing technique for cloud data center,” *Cluster Computing*, vol. 21, no. 2, pp. 1311–1329, 2018.
- [5] S. K. Mishra, D. Puthal, B. Sahoo, P. P. Jayaraman, S. Jun, A. Y. Zomaya, and R. Ranjan, “Energy-efficient vm-placement in cloud data center,” *Sustainable computing: informatics and systems*, vol. 20, pp. 48–55, 2018.
- [6] L. Minas and B. Ellison, *Energy efficiency for information technology: How to reduce power consumption in servers and data centers*. Intel Press, 2009.
- [7] K. Bilal, S. U. R. Malik, S. U. Khan, and A. Y. Zomaya, “Trends and challenges in cloud datacenters,” *IEEE cloud computing*, vol. 1, no. 1, pp. 10–20, 2014.
- [8] L. Wang and S. U. Khan, “Review of performance metrics for green data centers: a taxonomy study,” *The journal of supercomputing*, vol. 63, no. 3, pp. 639–656, 2013.
- [9] P. Sarwesh, N. S. V. Shet, and K. Chandrasekaran, “Effective integration of reliable routing mechanism and energy efficient node placement technique for low power iot networks,” *International Journal of Grid and High Performance Computing (IJGHPC)*, vol. 9, no. 4, pp. 16–35, 2017.

- [10] L. Wang, S. U. Khan, and J. Dayal, "Thermal aware workload placement with task-temperature profiles in a data center," *The Journal of Supercomputing*, vol. 61, no. 3, pp. 780–803, 2012.
- [11] "Google compute engine." <https://cloud.google.com/compute/>. Accessed: 2019-07-22.
- [12] "Google Cloud TPU." <https://cloud.google.com/tpu/>.
- [13] "Microsoft Azure: General purpose virtual machine sizes." <https://docs.microsoft.com/en-us/azure/virtual-machines/windows/sizes-general>. Accessed: 2019-07-22.
- [14] "Amazon EC2: General purpose virtual machine sizes." <https://aws.amazon.com/ec2/instance-types/>. Accessed: 2019-07-22.
- [15] S. K. Tesfatsion, J. Proaño, L. Tomás, B. Caminero, C. Carrión, and J. Tordsson, "Power and performance optimization in fpga-accelerated clouds," *Concurrency and Computation: Practice and Experience*, vol. 30, no. 18, p. e4526, 2018.
- [16] M. Tarahomi and M. Izadi, "A prediction-based and power-aware virtual machine allocation algorithm in three-tier cloud data centers," *International Journal of Communication Systems*, vol. 32, no. 3, p. e3870, 2019.
- [17] F.-H. Tseng, X. Wang, L.-D. Chou, H.-C. Chao, and V. C. Leung, "Dynamic resource prediction and allocation for cloud data center using the multiobjective genetic algorithm," *IEEE Systems Journal*, vol. 12, no. 2, pp. 1688–1699, 2017.
- [18] M. M. Nejad, L. Mashayekhy, and D. Grosu, "Truthful greedy mechanisms for dynamic virtual machine provisioning and allocation in clouds," *IEEE transactions on parallel and distributed systems*, vol. 26, no. 2, pp. 594–603, 2014.
- [19] K. S. Rao and P. S. Thilagam, "Heuristics based server consolidation with residual resource defragmentation in cloud data centers," *Future Generation Computer Systems*, vol. 50, pp. 87–98, 2015.
- [20] N. U. Ekici, K. W. Schmidt, A. Yazar, and E. G. Schmidt, "Accloud-man - power efficient resource allocation for heterogeneous clouds," in *2019 27th Sig-*

- nal Processing and Communications Applications Conference (SIU)*, pp. 1–4, April 2019.
- [21] N. U. Ekici, K. W. Schmidt, A. Yazar, and E. G. Schmidt, “Resource allocation for minimized power consumption in hardware accelerated clouds,” in *2019 28th International Conference on Computer Communication and Networks (ICCCN)*, pp. 1–8, July 2019.
- [22] “Aws elastic beanstalk.” <https://aws.amazon.com/elasticbeanstalk/>. Accessed: 2019-07-22.
- [23] “Salesforce.” <https://www.salesforce.com/>. Accessed: 2019-07-22.
- [24] “Dropbox.” <https://dropbox.com/>. Accessed: 2019-07-22.
- [25] C. Kachris, B. Falsafi, and D. Soudris, *Hardware Accelerators in Data Centers*. Springer, 2019.
- [26] I. Magaki, M. Khazraee, L. V. Gutierrez, and M. B. Taylor, “Asic clouds: Specializing the datacenter,” in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pp. 178–190, June 2016.
- [27] “EC2 FPGA Infographic.” http://d1.awsstatic.com/Amazon_EC2_F1_Infographic.pdf.
- [28] “Open Source Software For Creating Private and Public Clouds.” <https://www.openstack.org/>.
- [29] A. Patil, D. Patil, *et al.*, “An analysis report on green cloud computing current trends and future research challenges,” *An Analysis Report on Green Cloud Computing Current Trends and Future Research Challenges (March 19, 2019)*, 2019.
- [30] J. A Hoxmeier, C. Dicesare, and M. , “System response time and user satisfaction: An experimental study of browser-based applications,” *Proceedings of the Association of Information Systems Americas Conference*, 01 2000.
- [31] A. Yazar, A. Erol, and E. G. Schmidt, “Accloud (accelerated cloud): A novel fpga-accelerated cloud architecture,” in *2018 26th Signal Processing and Communications Applications Conference (SIU)*, pp. 1–4, IEEE, 2018.

- [32] A. Schrijver, *Theory of linear and integer programming*. John Wiley & Sons, 1998.
- [33] W. Lin, W. Wu, H. Wang, J. Z. Wang, and C.-H. Hsu, “Experimental and quantitative analysis of server power model for cloud data centers,” *Future Generation Computer Systems*, vol. 86, pp. 940–950, 2018.
- [34] D. Boru, D. Kliazovich, F. Granelli, P. Bouvry, and A. Y. Zomaya, “Energy-efficient data replication in cloud computing datacenters,” *Cluster computing*, vol. 18, no. 1, pp. 385–402, 2015.
- [35] “Openstack nova scheduler.” <https://docs.openstack.org/nova/latest/user/filter-scheduler.html>. Accessed: 2019-08-06.
- [36] “Azure Public Dataset.” <https://github.com/Azure/AzurePublicDataset>.
- [37] “Xilinx xc7z100 Product Description.” <https://www.xilinx.com/products/boards-and-kits/1-5okkz7.html>.
- [38] T. Chen, S. Srinath, C. Batten, and G. E. Suh, “An architectural framework for accelerating dynamic parallel algorithms on reconfigurable hardware,” in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 55–67, IEEE, 2018.
- [39] “Oracle Server Power Calculator.” <https://www.oracle.com/it-infrastructure/power-calculators/oracle-server-x8-2-power-calc.html>.
- [40] F. Koltuk, A. Yazar, and E. G. Schmidt, “Cloudgen: Workload generation for the evaluation of cloud computing systems,” in *2019 27th Signal Processing and Communications Applications Conference (SIU)*, pp. 1–4, April 2019.
- [41] D. T. Pham, S. S. Dimov, and C. D. Nguyen, “Selection of k in k-means clustering,” *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, vol. 219, no. 1, pp. 103–119, 2005.
- [42] N. U. Ekici, F. Koltuk, A. Yazar, K. W. Schmidt, and E. G. Schmidt, “Evaluation of resource allocation methods for heterogenous clouds with realistic traces,” in

2019 Digital Transformation and Smart Systems Conference (DTSS), pp. 1–4, IEEE, 2019.

[43] “Workloads and compute instances by application, Cisco Global Cloud Index: Forecast and Methodology, 2016–2021.” https://www.cisco.com/c/en/us/solutions/collateral/service-provider/global-cloud-index-gci/white-paper-c11-738085.html#_Toc503317523.

[44] “Microsoft Azure What is Azure.” <https://azure.microsoft.com/en-us/overview/>.

[45] M. Mao and M. Humphrey, “A performance study on the vm startup time in the cloud,” pp. 423–430, 06 2012.