UNIVERSITETI I EVROPËS JUGLINDORE
УНИВЕРЗИТЕТ НА ЈУГОИСТОЧНА ЕВРОПА
SOUTH EAST EUROPEAN UNIVERSITY

FAKULTETI SHKENCAVE DHE TEKNOLOGJIVE BASHKËKOHORE
ФАКУЛТЕТ ЗА СОВРЕМЕНИ НАУКИ И ТЕХНОЛОГИИ
FACULTY OF CONTEMPORARY SCIENCES AND TECHNOLOGIES

Third Cycle of Studies

Doctoral Dissertation Topic:

DYNAMIC RESOURCE ALLOCATION IN CLOUD COMPUTING

CANDIDATE:

M.Sc. Artan Mazrekaj

SUPERVISOR:

Prof. Dr. Bernd Freisleben

July 2020

Abstract

Cloud computing is a widespread Internet-based computing model that enables resource usage on demand and as a service. Under this paradigm, services and resources are delivered in a pay-as-you-go fashion, where cloud clients only pay for the resources they actually used and as long as they use these services and resources. On the other hand, the cloud provider's goal is to provide high-performance services and resources at optimal cost for the cloud clients, and this can be achieved through dynamic allocation of Virtual Machines (VMs) according to workload changes in order to meet Service Level Agreement (SLA) criteria. To schedule the incoming tasks from cloud consumers and to efficiently manage computer resources, cloud providers use scheduling algorithms. Task scheduling and resource allocation enable cloud providers to maximize revenue and utilize resources properly, so both are important issues in cloud infrastructures.

In this dissertation, the problem of task scheduling at the time these tasks are submitted to the proper VMs in the cloud environment is addressed and solutions are proposed. Effective task scheduling approaches reduce the task completion time, increase the efficiency of resource utilization, and improve the quality of service and the overall performance of the system. A novel task scheduling algorithm for cloud environments based on the Heterogeneous Earliest Finish Time (HEFT) algorithm, called experiential HEFT, is proposed. It considers experiences with previous executions of tasks to determine the workload of resources. To realize the experiential HEFT algorithm, we propose a novel way of HEFT rank calculation to specify the minimum average execution time of previous runs of a task on all relevant resources. Experimental results show that the proposed experiential HEFT algorithm performs better than existing approaches considered in our evaluation.

To dynamically allocate resources to VMs in an Infrastructure-as-a-Service (IaaS) cloud environment, a resource management solution is proposed. It combines local and global VM resource allocations. Local resource allocation means allocating CPU resource shares to VMs according to the current load. Global resource allocation means performing live migration actions when a host is overloaded or underloaded in order to mitigate VM performance violations and reduce the number of hosts to save energy. To detect if a host is overloaded or underloaded, an approach based on long-term resource

usage predictions is used, while for the long-term predictions a supervised machine learning approach based on Gaussian Processes is proposed. Experimental results show that long-term predictions of resource usage can increase stability and overall performance of the cloud infrastructure.

Knowing that overload or underload detection based on long-term predictions carries with it the uncertainty of correct predictions, which can lead to erroneous decisions, we propose an approach in which we have considered the uncertainty of long-term predictions and the live migration overhead. To consider the uncertainty of long-term predictions for overload detection, a novel probabilistic model of the prediction error is built online using the non-parametric kernel density estimation method. Based on experimental results, making overload detection decisions proportional to the uncertainty of predictions increases the overall system performance of the live migration process.

Finally, to address the problem of the VM consolidation approaches that rely on a centralized architecture, a distributed resource allocation solution based on multi-agent systems is proposed. Our approach uses a utility function based on host CPU utilization to drive live migration actions. Agents, attached to each physical machine, are responsible for making decisions for the live migration of VMs from one host to another host. The key novel feature of the proposed approach is that allocation decisions are based on the individual agents' utility functions, which offers the flexibility of easily changing the allocation policy. Experimental results show that the utility-based distributed resource allocation approach achieves better overall performance compared to a centralized approach and a threshold-based distributed approach.

Abstrakt

Cloud computing është një model i përhapur kompjuterik i bazuar në Internet që mundëson përdorimin e burimeve sipas kërkesës dhe i cili ofrohet si shërbim. Sipas këtij modeli, shërbimet dhe burimet ofrohen sipas mënyrës pay-as-you-go, ku klientët e cloud-it paguajnë vetëm për burimet që ata i kanë në shfrytëzim dhe për sa kohë që ata i shfrytëzojnë këto shërbime dhe burime.

Nga ana tjetër, qëllimi i ofruesit të cloud-it është të sigurojë shërbime dhe burime me performancë të lartë dhe me kosto optimale për përdoruesit e cloud-it, dhe kjo mund të arrihet përmes alokimit dinamik të Makinave Virtuale (MV) varësisht nga ndryshimet e ngarkesës, në mënyrë që të përmbushë kriteret e *Service Level Agreement (SLA)*.

Për të planifikuar detyrat në hyrje nga përdoruesit e cloud-it dhe për të menaxhuar në mënyrë efikase burimet kompjuterike, ofruesit e cloud-it përdorin algoritme për planifikim. Planifikimi i detyrave dhe alokimi i burimeve u mundëson ofruesve të cloud-it të maksimizojnë të ardhurat dhe të përdorin burimet siç duhet, kështu që të dyja këto janë çështje të rëndësishme në infrastrukturën e cloud.

Në këtë disertacion, është adresuar problemi i planifikimit të detyrave në kohën kur këto detyra u dërgohen VM-ve të përshtatshme në mjediset cloud dhe janë propozuar zgjidhje. Qasjet efektive të planifikimit të detyrave zvogëlojnë kohën e përfundimit të detyrës, rrisin efikasitetin e përdorimit të burimeve dhe përmirësojnë kualitetin e shërbimit dhe performancën e përgjithshme të sistemit. Është propozuar një algoritëm i ri për caktimin e detyrave për mjediset cloud, bazuar në algoritmin *Heterogeneous Earliest Finish Time (HEFT)*, i quajtur *experiential HEFT*. Algoritmi merr në konsideratë përvojat me ekzekutimet e mëparshme të detyrave, ashtu që të përcaktoj ngarkesën e burimeve. Për të realizuar algoritmin *experiential HEFT*, ne propozojmë një mënyrë të re të llogaritjes së rankut të algoritmit HEFT, për të specifikuar kohën mesatare minimale të ekzekutimeve të mëparshme të një detyre, në të gjitha burimet përkatëse. Rezultatet eksperimentale tregojnë se algoritmi i propozuar *experiential HEFT* përformon më mirë se sa qasjet ekzistuese të cilat janë marrë në konsideratë në vlerësimin tonë.

Për të alokuar në mënyrë dinamike burimet për MV-të në një mjedis të cloud-it Infrastruktura-si-shërbim (IaaS), është propozuar një zgjidhje e menaxhimit të burimeve. Ai kombinon alokimet locale dhe globale të burimeve të MV.

Alokimi i burimeve lokale nënkupton ndarjen e burimit të CPU-së në MV-të sipas ngarkesës aktuale. Alokimi I burimeve globale nënkupton kryerjen e veprimeve të migrimit kur një host është i

mbingarkuar ose i nënngarkuar në mënyrë që të lehtësojë shkeljet e performancës së MV-së dhe të zvogëlojë numrin e hostave për të kursyer energji. Për të detektuar nëse një host është i mbingarkuar ose i nënngarkuar, përdoret një qasje e bazuar në parashikimet afatgjata të përdorimit të burimeve, ndërsa për parashikimet afatgjata është propozuar një qasje *supervised machine learning* e bazuar në Proceset Gaussiane. Rezultatet eksperimentale tregojnë se parashikimet afatgjata të përdorimit të burimeve mund të rrisin stabilitetin dhe performancën e përgjithshme në infrastrukturës cloud. Duke e ditur se detektimi i mbingarkesës ose nënngarkesës bazuar në parashikimet afatgjata bartë me vete pasigurinë e parashikimeve të sakta, të cilat mund të çojnë në vendime të gabuara, ne propozojmë një qasje në të cilën kemi marrë parasysh pasigurinë e parashikimeve afatgjata dhe ngarkesën e live migrimit. Për të marrë parasysh pasigurinë e parashikimeve afatgjata për zbulimin e mbingarkesës, një model i ri probabilistik i gabimit të parashikimit është ndërtuar në internet duke përdorur metodën *non-parametric kernel density*. Bazuar në rezultatet eksperimentale, marrja e vendimeve për zbulimin e mbingarkesës në përpjesëtim me pasigurinë e parashikimeve rritë performancën e përgjithshme të sistemit në procesin e live migrimit.

Së fundi, për të adresuar problemin e qasjeve të konsolidimit të MV-ve që mbështeten në një arkitekturë të centralizuar, është propozuar një zgjidhje e shpërndarë e alokimit të burimeve bazuar në sistemet me shumë agjentë. Qasja jonë përdorë një funksion utilitar të bazuar në përdorimin e CPU-së së hostit për të drejtuar veprimet e live migrimit. Agjentët, të bashkangjitur në çdo makinë fizike, janë përgjegjës për marrjen e vendimeve për live migrimin e MV-ve nga një host në tjetrin. Karakteristika kryesore e qasjes së propozuar është që vendimet për alokim bazohen në funksionet utilitare të agjentëve individual, e që ofron fleksibilitet të ndryshimit të lehtë të policës së alokimit. Rezultatet eksperimentale tregojnë se qasja e alokimit të burimeve të shpërndarë të bazuar në utilitet arrin performancë më të mirë të përgjithshme në krahasim me qasjet e centralizuara dhe qasjet e shpërndara të bazuara në prag.

Acknowledgments

First and foremost, I would like to thank my advisor, Prof. Dr. Bernd Freisleben.
His continuous support, advice, and encouragement have made it possible to finish this thesis.
I am especially grateful for his patience in discussing challenges in the context of this thesis and for his advice on how to look at problems from different dimensions.

I am very grateful forever to Dr. Dorian Minarolli for his advice, motivation, and support during my PhD studies, without which the completion of this thesis would have been more difficult.

I am deeply indebted to my parents, brother, and sisters for their lifelong support, trust, and encouragement.

Last but not least, I would like to thank my wife Jehona for her endless patience and love, and my children Erdi and Teo for their unconditional love and happiness.

CONTENTS

List of Figures

List of Abbreviations

Abbreviations	Explanation
AaaS	Analytics as a Service
ACO	Ant Colony Optimization
BFD	Best Fit Decreasing
BPaaS	Business Process as a Service
CaaS	Communications as a Service
CFS	Constant Fixed Selection
CompaaS	Compute as a Service
CP	Constraint Programming
CPOP	Critical Path on a Processor
DaaS	Desktop as a Service
DC	Data Center
DLS	Dynamic Level Scheduling
DSaaS	Data Storage as a Service
DSB	Dynamic Self-Ballooning
DVMC	Dynamic Virtual Machine Consolidation
EHEFT	Experiential Heterogeneous Earliest Finish Time
EFT	Earliest Finish Time
FF	First Fit
FFD	First Fit Decreasing
FFT	Fast Fourier Transform
FQL	Fuzzy Q-Learning
GA	Global Agent
HA	Host Agent
HEFT	Heterogeneous Earliest Finish Time
HPG	High Potential Growth
IaaS	Infrastructure as a Service
IQR	Inter Quartile Range
KVM	Kernel-based Virtual Machine
LP	Linear Programming
LR	Local Regression

LRR	Local Robust Regression
QoS	Quality of Service
MaaS	Monitoring as a Service
MAD	Median Absolute Deviation
MC	Maximum Correlation
MH	Mapping Heuristic
MIPS	Millions of Instructions per Second
ML	Machine Learning
MMT	Minimum Migration Time
MU	Minimum Utilization
NaaS	Network as a Service
NF	Next Fit
PaaS	Platform as a Service
PABFD	Power Aware Best Fit Decreasing Algorithm
P2P	Peer to Peer
PM	Physical Machine
PSO	Particle Swarm Optimization
SaaS	Software as a Service
SecaaS	Security as a Service
RC	Random Choice
RL	Reinforcement Learning
RR	Round Robin
RS	Random Selection
SA	Simulated Annealing
SLA	Service Level Agreement
SLAV	Service Level Agreement Violations
SVMC	Static Virtual Machine Consolidation
TH	Threshold
VM	Virtual Machine
VMC	Virtual Machine Consolidation
VMM	Virtual Machine Monitor
VPC	Virtual Private Cloud
XaaS	X as a Service

Part I.

Fundamentals and Related Work

1

Introduction

1.1. Motivation

Cloud computing has become an effective solution for providing a flexible, manageable, on-demand and dynamically scalable computing infrastructure for many applications. Cloud computing also presents an important trend in the development of computing technologies, systems and architectures, and receives great interest from academia and industry.

In general, this technology has enabled businesses, governmental organizations and other institutions to benefit in time, quality of service, management, and operational cost. From the point of view of enterprises, providers and consumers, other benefits are cost effective, on-demand self-services, high efficiency, availability, flexibility, scalability and reliability, resource utilization, applications as utilities over the Internet, configuring applications online, online development and deployment tools, etc. The cloud consumers can use resources according to the pay-as-you-go model, where payment is made depending on consumption and this in itself constitutes a benefit to consumers.

Cloud service models are classified into three groups:

(1) Software-as-a-Service (SaaS) where applications that are in cloud infrastructure can be accessed by various consumers' devices through a thin client interface (such as a web browser).

(2) Platform-as-a-Service (PaaS) provides developers with a platform where they can develop, test, deploy and host different applications.

(3) Infrastructure-as-a-Service (IaaS) provides consumers with computing resources such are processing, storage, networks and other fundamental resources. The IaaS model is viewed having huge interest for research by the fact that management of resources in the cloud infrastructure is a

complex issue, knowing the ever-increasing demand for computing resources. Seeing this model as important, we have focused our work on IaaS.

Therefore, to increase the efficiency of the management of computing resources in a cloud infrastructure, virtualization technology has a key role in modern data centers and cluster systems. The use of virtualization technologies has greatly reduced operational costs, has enabled the creation of a suitable environment for application development, debugging and testing. In addition, this increase in computing efficiency results in lower space, maintenance, cooling and electricity costs [2].

Cloud computing infrastructures based on virtualization technology consist of multiple virtualized nodes, in which multiple applications and services are running in Virtual Machines (VMs) [1]. The virtualization layer uses lower-level resources to create multiple VMs known as a Virtual Machine Monitor (VMM). Hence, this technology has enabled consumers to benefit from the most efficient use of resources and as well as in the most favourable cost. However, the dynamic allocation of resources to a virtualized infrastructure is a complex issue, based on the large number of physical machines in the data center, rapid increase in demand and workload. Considering this complexity, it is necessary to create the powerful mechanisms through techniques and algorithms for automation and controlled resource management.

1.2. Research Challenges

In a data center, the primary goal of a dynamic resource allocation process is to avoid wasting resources as a result of under- and over-utilization, which may result in a violation of the Service Level Agreements (SLA) between the customers and the cloud provider. A key role in the dynamic resource allocation process is played by important mechanism known as a VM live migration. Live migration of VMs in a cloud infrastructure means moving the VM which is running on a physical machine (source node) to another physical machine (destination node), while the VM is powered up. In general, live migration of VMs is a process that has a cost in terms of consumption of resources. When we consider the architecture of cloud infrastructures consisting of thousands of physical machines, the migration process affects the performance of the system and applications. Task scheduling is also an important issue closely linked to dynamic resource allocation in cloud infrastructures. There are several challenges related to dynamic resource allocation in cloud infrastructure, as described below.

- **Long-term predictions of resources challenges**. Dynamic workloads in cloud infrastructures can be managed through live migration of virtual machines from overloaded or underloaded physical machines (PMs) to other PMs to save energy and to mitigate performance related to Service Level Agreement (SLA) violations. Therefore, VM live migration is a resource allocation mechanism that dynamically consolidates the low utilization VMs on few PMs. An important issue of the live migration mechanism is to detect when a host is overloaded or underloaded. The question that arises here is that: how to make long-term predictions of PMs' workloads by predicting resource utilization for detecting their overload and underload states? Some existing approaches [19] [44] [46] are based on monitoring resource usage, and if the current or the predicted next value exceeds a specified threshold, then a host is declared as overloaded. The problem with these approaches is that basing decisions for host overload detection on a single resource usage value or a few future values can lead to hasty decisions, unnecessary live migration overhead, and stability issues. Thus, detecting whether a host is overloaded or underloaded is based on current or short-term predictions of resource usage and static usage thresholds, which can be sensitive to short spikes of load that can cause stability problems and unnecessary live migrations. There are other approaches [21] that apply heuristic algorithms for host overload and underload detection based on statistical analysis of historical resource usage data. In this case, an adaptive usage threshold is used based on statistical parameters of previous data, such as CPU usage Median Absolute Deviation (MAD) or interquartile range (IQR). The authors also apply local regression methods for predicting CPU usage values some time ahead into the future.

- **VM resource allocation challenges.** In the process of dynamic resource allocation as a continuation of the above challenges, there are other challenges as well, especially in the mapping of VMs to PMs. The process of mapping VMs to PMs in principle has the role of load balancing and to avoid performance level violations.

  There are several questions that can be raised here, such as: When to migrate a VM? Which VM needs to migrate (selection of VM)? Which is the destination node (PM), and where should the selected VM be placed? The typical problem here is the mapping between VMs to PMs and dynamically changing it, when the workload changes during run time, in order to reduce the number of PMs and to keep application performance to SLA conditions. In general, this can be seen as a bin-packing problem of packing VMs of different sizes into the smallest

numbers of PMs (bins). Some of the existing approaches [36] [46] have treated it as a bin-packing problem, where the sizes of items to be packed are VM resource utilizations (e.g. CPU utilization) and the sizes of bins are resource capacities of PMs. The bin-packing problem itself is an NP-hard problem so it requires heuristic methods to find approximate solutions to the problem. The problem with these approaches is deciding when and which VM to migrate, based on the use of low-level metrics, such as resource utilization. But the decision based solely on this metric omits other high-level performance metrics, especially when it is known that performance changes with the workload. Another problem with the bin-packing technique is that as the workload changes, the size of the VMs also changes, so the problem becomes multi-size bin packing with items to be packed elastically in size. The data center also has heterogeneous PMs of different resource capacities, so the bins are of different sizes. Also, the existing approaches have used policies implemented on heuristic ordering algorithms, such are First-Fit-Decreasing (FFD) and Best-Fit-Decreasing (BFT), to packed VMs to few PMs, but these policies are not suitable when a load balancing policy needs to be implemented in the data center and more PMs are added. Therefore, a more flexible approach that can change according to data center policy changes is required.

- **Centralized resource allocation architecture challenges.** An important mechanism that provides major benefits for data centers is live migration of VMs, from one PM to another. Live migration of VMs enables resource allocation to running services without interruption during the migration process, and this is an important feature especially for services with particular quality of service (QoS) requirements. There are several VM resource allocation approaches [49] [56-57] that use a centralized architecture in order to reduce energy consumption and number of migrations in the data center. These approaches use the hierarchical architecture based on multi-agents for the dynamic VM consolidation task in a large-scale data center. The problem with these approaches is that they use a central controller (agent) and that does not scale well for large cloud infrastructures, might represent a communication bottleneck, and is a single point of failure in terms of reliability. Another issue is that the central controller (agent) should communicate with other local agents to make a decision about the migration of the VMs, and this can lead to delays and decrease the overall communication performance in large-scale systems. A central controller keeps the information about the capacity of available resources of all the PMs. The controller runs the

centralized Virtual Machine Consolidation (VMC) algorithm that selects a destination PM for the selected VM migration, taking into account the resource availability of all PMs. In such situations, there is a need for a new approach based on a distributed architecture where the decision to initiate the live migration process is not taken by a central controller or agent, but the responsibility is distributed to individual agents.

1.3. Research Contributions

The main contributions of this dissertation are several approaches that enable the dynamic allocation of resources in a cloud infrastructure:

- **Long-term predictions for physical machine overload and underload detection in cloud infrastructures**

  In cloud infrastructures, the dynamic workloads can be managed through live migration of VMs from overloaded or underloaded PMs to other PMs in order to reduce power consumption and to satisfy the SLA requirements. An important problem is how to detect when a PM is overloaded or underloaded in order to initiate the live migration actions in time. An approach that tackles long-term predictions of resource demands of VMs for PM overload detection is presented. This approach enables live migration decisions to be based on resource usage predictions several steps ahead into the future. This increases the stability and application performance and allows cloud providers to predict overload states before they happen. To dynamically allocate resources to VMs in an IaaS cloud, the approach combines local and global VM resource allocations. Local resource allocation enables allocating CPU resource shares to VMs according to the current load, while global resource allocation enables live migration actions when a PM is overloaded or underloaded in order to reduce the number of PMs and energy consumption and to mitigate VM performance violations.

  For overload or underload PM detection, long-term predictions of resource usage are made, based on Gaussian processes as a machine learning approach for time series forecasting. Compared to existing works, this approach considers long-term predictions of resource

demand and thus can increase stability and overall performance in cloud environments. The approach is presented more detail in Chapter 4.

- **Tackling uncertainty in long-term predictions**

  Knowing that overload or underload detection based on long-term predictions carries with it the uncertainty of correct predictions, which can lead to erroneous decisions, we consider the uncertainty in the migration process during virtual machine consolidation. Unlike existing approaches, we propose an approach in which we have considered the uncertainty of long-term predictions and the live migration overhead. To consider the uncertainty of long-term predictions for overload detection, a novel probabilistic model of the prediction error is built online using the non-parametric kernel density estimation method. The approach is explained in more detail in Chapter 4.

- **The Experiential Heterogeneous Earliest Finish Time Algorithm for task scheduling in clouds**

  With the enormous growth of cloud computing as a computation model, the number of consumers and the demand for cloud resources also increases accordingly. Two basic functions in cloud resource management, task scheduling and resource allocation, are responsible for assigning costumer jobs to the appropriate resources to perform. In this context, an issue closely related to the dynamic allocation of resources in the cloud environment is the problem of task scheduling at the time the tasks are submitted to the proper VM. The execution of a task has a cost and depends on how the resources are allocated. Resource allocation constraints define restrictions on how to allocate resources, and scheduling under resource allocation constraints provide proper resource allocation to tasks. In cloud environments, the physical machines are located in different geographical locations and have different abilities in the way their resources perform, and each has its own cost ratio. Therefore, in these situations we should consider the cost and makespan associated with the task schedule and the resources allocated. Starting from these premises and constraints, resource allocation and task scheduling should be carefully coordinated and optimized jointly in order to achieve an overall cost and time effective schedule. In this manner, by minimizing cost and makespan, the task scheduling process can be optimized.

  An approach that addresses the problem of task scheduling in cloud environments is presented. The proposed algorithm, called experiential HEFT (EHEFT), is based on the existing

algorithm known as Heterogeneous Earliest Finish Time (HEFT). The EHEFT algorithm considers experiences with previous executions of tasks to determine the workload of resources. The algorithm also defines rank calculation to specify the minimum average execution time of previous runs of a task on all relevant resources. Compared to existing work, this approach enables effective task scheduling by reducing the task completion time, increase the efficiency of resource utilization, and improve the quality of service and the overall performance of the system. The approach is presented in more detail in Chapter 5.

- **Distributed resource allocation in cloud infrastructures using multi-agent systems**

  The problem with existing approaches is that they are based on centralized VM resource allocation architectures, which is considered a drawback because a central controller can be seen as a single point of failure in the communication process. Therefore, we propose a distributed VM resource allocation approach for VM consolidation relying on multi-agent systems. This approach uses a utility function based on host CPU utilization to drive live migration actions. Agents, attached to each PM, are responsible for making decisions for the live migration of VMs from one PM to another PM. The key feature of the proposed approach is that allocation decisions are based on the individual agents' utility functions, which offers the flexibility of easily changing the allocation policy. Compared to other approaches, the utility-based distributed resource allocation approach achieves a better overall performance avoiding a single point of failure and offers scalability in large-scale clouds. The approach is discussed in more detail in Chapter 6.

1.4. Publications

The research papers that have been published during this dissertation are:

1. Dorian Minarolli, Artan Mazrekaj, and Bernd Freisleben. Tackling Uncertainty in Long-term Predictions for Host Overload and Underload Detection in Cloud Computing. In *Journal of Cloud Computing: Advances, Systems and Applications 6(4)*, Springer, 2017.

2. Artan Mazrekaj, Dorian Minarolli, and Bernd Freisleben. Distributed Resource Allocation in Cloud Computing using Multi-Agent Systems. In *Telfor Journal*, pp. 110-115, 2017.

3. Artan Mazrekaj, Dorian Minarolli, and Bernd Freisleben. Dynamic Resource Allocation in Cloud Environments. In *Information & Communication Technologies at Doctoral Student Conference* (ICT@DSC), Thessaloniki, Greece, pp. 105-114, May 2018.

4. Artan Mazrekaj, Arlinda Sheholli, Dorian Minarolli, and Bernd Freisleben. The Experiential Heterogeneous Earliest Finish Time Algorithm for Task Scheduling in Clouds. In *9th International Conference on Cloud Computing and Services Science (CLOSER 2019)*, Heraklion, Crete, Greece, pp. 371-379, May 2019.

5. Artan Mazrekaj, Shkelzen Nuza, Mimoza Zatriqi, Vlera Alimehaj. An Overview of Virtual Machine Live Migration Techniques. *International Journal of Electrical and Computer Engineering (IJECE)*, Vol. 9, No. 5, 2019.

1.5. Organisation of this Dissertation

This dissertation is organized as follows:

Chapter 2 introduces the fundamental topics of this work. This includes cloud computing, cloud service models, cloud computing deployment models, cloud computing actors, virtualization, and Service Level Agreements (SLAs). An overview of the topic of dynamic resource allocation as the main focus of this thesis is also given.

Chapter 3 presents an overview of related work in the area of resource allocation in cloud infrastructures.

Chapter 4 presents the proposed resource allocation approach based on long-term predictions for PM overload and underload in cloud infrastructures. Uncertainty of long-term predictions is also addressed. The approach, its implementation, and experimental results are presented.

Chapter 5 presents the proposed task scheduling approach as an issue closely related to the dynamic allocation of resources in the cloud environment. The approach is based on the proposed algorithm called experiential HEFT (EHEFT) which addresses the problem of task scheduling at the time the tasks are submitted to the proper VM. The approach, its implementation, and experimental results are presented.

Chapter 6 presents the proposed distributed VM resource allocation approach for VM consolidation relying on multi-agent systems. This approach uses a utility function based on host CPU utilization to drive live migration actions. The approach, its implementation, and experimental results are presented.

Finally, Chapter 7 concludes the dissertation and discusses areas of future work.

2

Fundamentals

2.1. Introduction

In this chapter, we present the basic concepts of cloud computing. Section 2.2 deals with fundamental concepts in the cloud, starting from key cloud characteristics, cloud service models, cloud computing deployment models, cloud computing actors. An overview of virtualization is given in Section 2.3 that has an important role to build a cloud infrastructure environment. Section 2.4 deals with Service Level Agreements (SLA), which represent an agreement between cloud provider and cloud consumers. Resource allocation in cloud infrastructures is covered in Section 2.5. This section also addresses live migration of virtual machines as an important dynamic resource allocation mechanism. The evaluation metrics that are the key indicators to increase efficiency in VM consolidation and quality of services are presented in Section 2.6. In section 2.7, the CloudSim simulator is described, a framework for modelling and simulation of cloud computing infrastructures and services. Section 2.8 summarizes this chapter.

2.2. Cloud Computing

Cloud computing has become a significant technology trend from which businesses and individuals access applications from anywhere in the world. Users can use services from the cloud when and where they need them, in the amount that they need, and pay for only the resources they use.

For the term cloud computing, there are many definitions from academy and industry practitioners.

Buyya *et al*. [3] define a cloud as: *"A Cloud is a type of parallel and distributed system consisting of a collection of inter-connected and virtualized computers that are dynamically provisioned and*

presented as one or more unified computing resources based on service-level agreements (SLA) established through negotiation between the service provider and consumers."

The NIST (National Institute of Standards and Technology) [4] defines cloud computing as: "*Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics, three service models, and four deployment models.*"

Armbrust et al. [5] summarizes the key characteristics of cloud computing as: "*(1) the illusion of infinite computing resources; (2) the elimination of an up-front commitment by cloud users; and (3) the ability to pay for use of computing resources on a short-term basis as needed . . .*" [6].

2.2.1 Key Cloud Characteristics

The main characteristics of cloud computing [7] [41] shown in Figure 2.1:

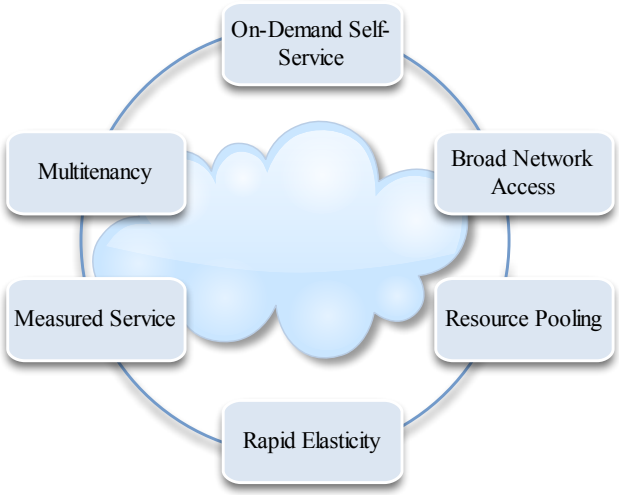


Figure 2.1. Key cloud characteristics

- *On-demand self-service*: a consumer can unilaterally provision computing capabilities.
- *Broad network access*: accessibility through heterogeneous thin or thick client platforms (e.g., mobile phones, laptops, workstations, and PDAs).
- *Resource pooling*: computing resources are pooled to serve multiple consumers, and are dynamically assigned and reassigned according to demand. The consumer has no control or

knowledge over the exact location of resources but may be able to specify location (e.g. country, state, or data center). Examples of resources include storage, processing, memory and network bandwidth.

- *Rapid elasticity:* shared pooled resources are used to enable horizontal scalability.
- *Measured service*: use of resources is automatically controlled and optimized through metering capabilities.
- *Multitenancy*: a feature that enables multiple consumers to use the same resources.

2.2.2 Cloud Service Models

All services related to IT resources, including network resources, infrastructure, platforms and other application offered by cloud computing are called a cloud service.

Cloud services are classified into three categories: a) Software as a Service (SaaS), b) Platform as a Service (PaaS), and c) Infrastructure as a Service, that are shown in the Figure 2.2.

- *Software as a Service (SaaS)*: In this model, applications are hosted by a cloud vendor and offered as a service to the users. This provides a great benefit to consumers because they do not need to take care to install and run applications locally, to upgrade and maintenance software, about software licenses, and so on. They only have to pay for services they use. Some of the SaaS providers are Google Apps, SalesForce, NetSuite, Oracle, IBM, Microsoft, etc.

- *Platform as a Service (PaaS):* In this model, platforms and tools (including all the systems and environments) are hosted by a cloud vendor and offered to developers, allowing them to develop, test, deploy and to host web applications. Some of the PaaS are Google Apps Engine, Microsoft Azure, Amazon's Web services, etc.

- *Infrastructure as a Service (IaaS):* In this model, resources are delivered over the Internet, such as servers, processing, storage, networks and other data center facilities. This model also provides benefits to the consumer because it does not need to manage or control the underlying cloud infrastructure. However, the consumer may have control over storage, operating systems, deployed applications and limited control of select network components. Some of the IaaS providers are Amazon EC2, Google Compute Engine, GoGrid, Flexiscale, OpenNebula, Rackspace, etc.

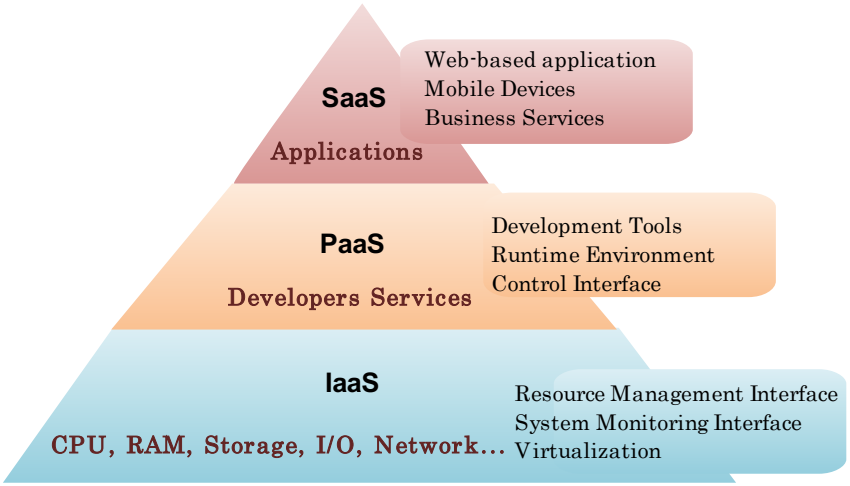


Figure 2.2. Cloud service models

By enhancing the domain of cloud services as well as expanding various functions related to security, privacy, user access management, compliance requirements, etc., the need for other cloud support services emerged. Several of these are [7] [8]:

- *Communications as a Service (CaaS)* serves as an interface of communication across multiple devices, such are video teleconferencing, web conferencing, instant messaging and voice over IP.
- *Compute as a Service (CompaaS)* serves to provide compute capacity.
- *Network as a Service (NaaS)* supports transport connectivity and intercloud network services, including routing, bandwidth, multicast protocols, VPN, security features etc.
- *Data Storage as a Service (DSaaS)* stores data in multiple third-party servers known as cloud storage;
- *Analytics as a Service (AaaS) / Data Analytics as a Service (DaaaS)* provides platforms and tools for analyses and mining of big data. Consumers and businesses can analyze their data in the cloud.
- *Desktop as a Service (DaaS)* provides consumers to build, configure, manage, store and execute remotely.
- *Business Process as a Service (BPaaS)* combines business process outsourcing with SaaS.
- *Security as a Service (SecaaS)* deals with the security services include authentication, antivirus, anti-malware, encryption, web security, etc.

- *Monitoring as a Service (MaaS)* deals with the process of state monitoring of the services, such are networks, systems and other features in the cloud.
- *X as a Service (XaaS) is* a recently developed model which refers as “Anything as a Service”, “Everything as a Service”, and “X as a Service”. So, this model focuses more on the relationship between customers and providers, it also has benefits to costumers over issues as total costs are controlled and reduced, risks are reduced and to accelerate innovation.

2.2.3 Cloud Computing Deployment Models

Depending on the functions, management, ownership and services, the cloud deployment models are categorized as follows.

- *Public Cloud*, also known as external cloud or multi-tenant cloud, in which the infrastructure and computing resources are openly available to the general public such are business, industry, government, non-profit organizations and individuals. Cloud infrastructure and services provided and managed from the cloud providers. The most popular public clouds are Google, Microsoft, Amazon Web Services, etc. The main characteristics of this model are homogenous infrastructure, shared resources, multi-tenant, common policies, leased or rented infrastructure, and economies of scale. Figure 2.3 shows an example of the public cloud model.

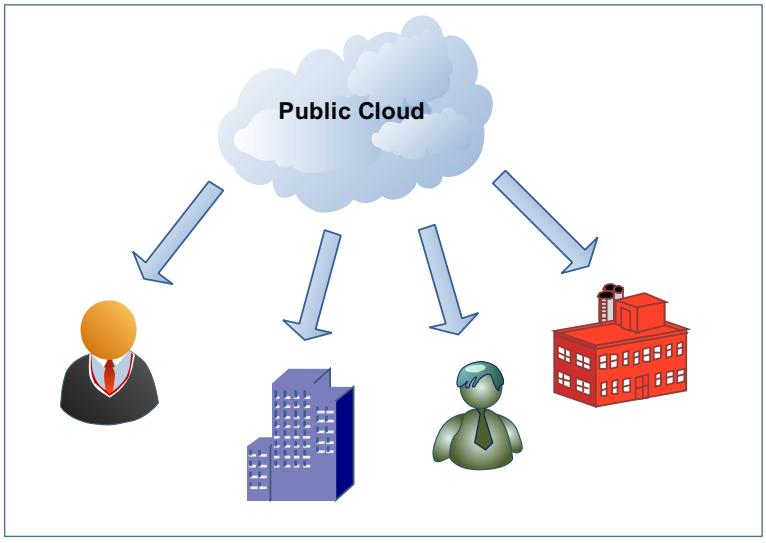


Figure 2.3: A public cloud model

- *Private Cloud* is considered an internal cloud which is provided and managed by an enterprise or a third party and which is not available to the general public. The main features and

benefits of private clouds are higher security and privacy, more control, improved reliability, cost and energy efficiency, customized policies, dedicated resources, in-house infrastructure and so on. Figure 2.4 shows an example of the private cloud model.

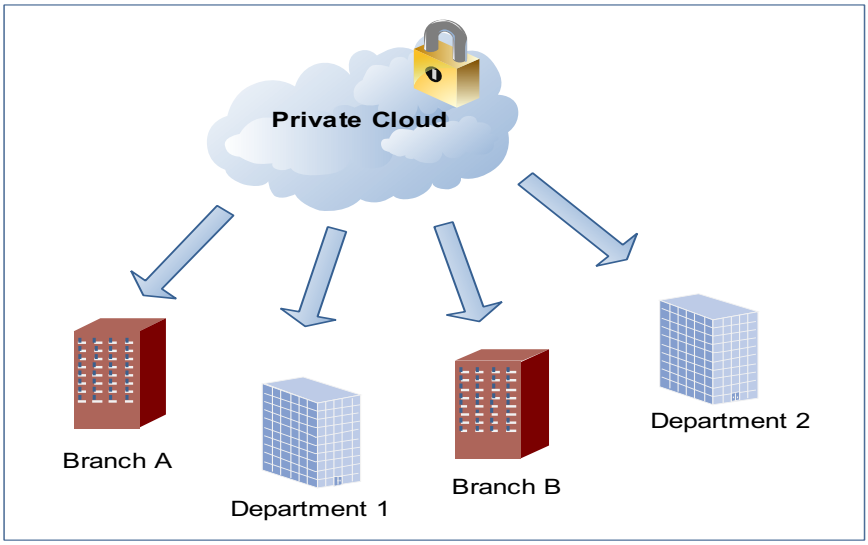


Figure 2.4: A private cloud model

- *Community Cloud* shares infrastructure and computing resources across several organizations. The resource management is governed by organizations that have participated or by third-party providers. Figure 2.5 shows an example of the community cloud model.

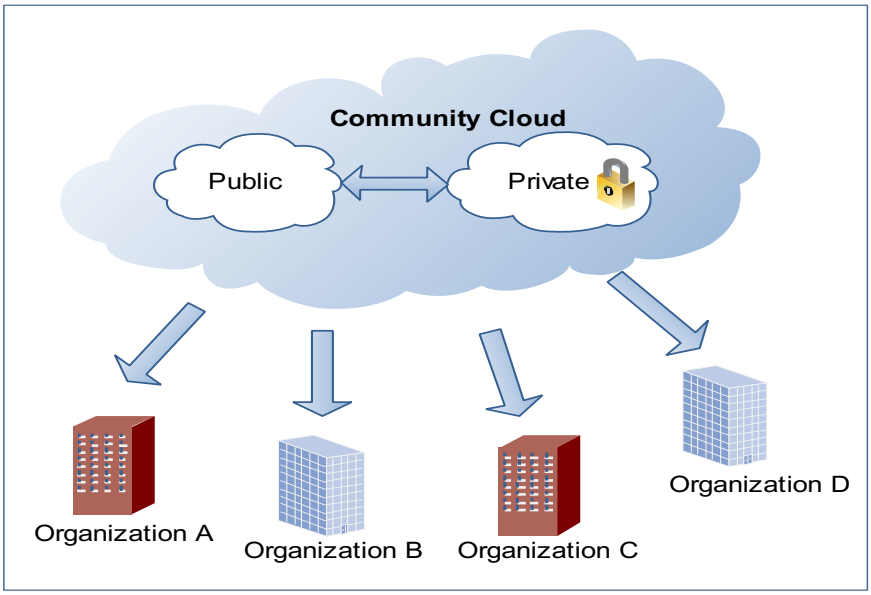


Figure 2.5: A community cloud model

- *Hybrid Cloud* combines of computing resources provided by two or more clouds (private, community, or public). The main benefits of hybrid cloud are in relation to cost benefits, improved security, scalability, risk management, more opportunity for innovation, etc. Figure 2.6 shows an example of the hybrid cloud model.

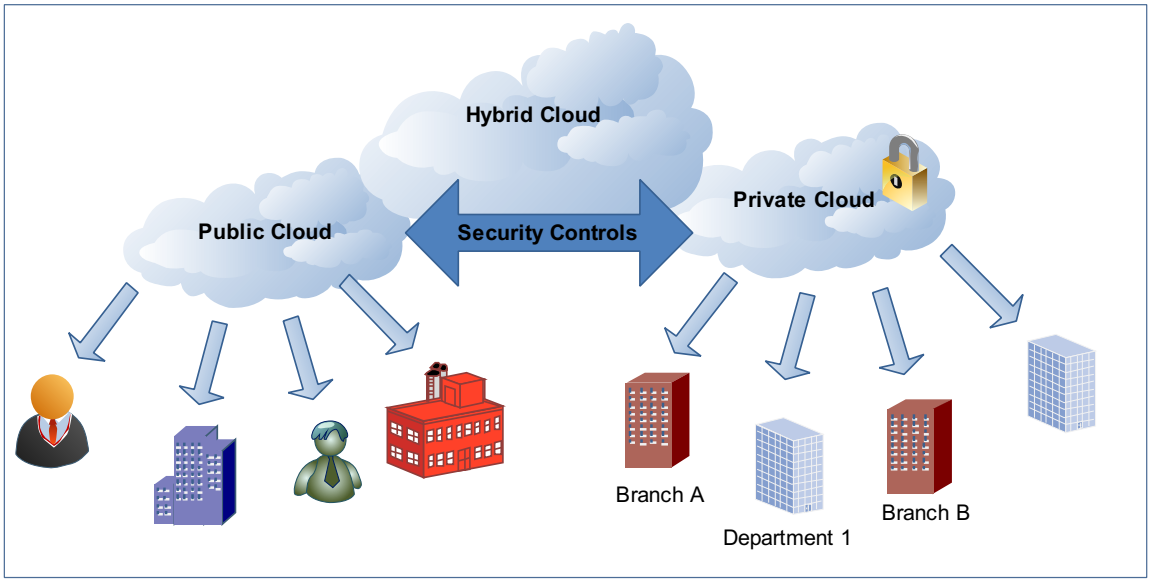


Figure 2.6: A hybrid cloud model

- *Virtual Private Cloud* is a segment of a public cloud that enables the users more control over the resources they use. This model provides secure communication between an enterprise and a public cloud provider. The customer's data is isolated from the data of other customers inside the cloud provider's network.

The NIST cloud computing reference architecture [9] defines the key actors in terms of the roles and responsibilities.

- *Cloud Provider* is a person, organization or entity responsible for making a service available to interested parties. The duties of a cloud provider are to manage the computing infrastructure, runs applications or software and arranges to deliver the cloud services to the cloud consumers over the Internet. In the context of the SaaS model, the cloud provider is responsible to configure, maintain, deploy updates and control the operation of the software applications on a cloud infrastructure. In the context of the PaaS model, the cloud provider is responsible to manage the platforms and tools, such as databases, software stacks, middleware components, IDEs, SDKs, and other development and deployment tools etc.

- *Cloud Consumer is* a person or organization that maintains a business relationship with and uses the service from the cloud provider. The cloud consumer to pick up services from a cloud provider can analyze services and pricing bids and is free to choose a cloud provider. After the cloud provider is determined, the cloud consumer needs to sign an agreement between the two parties (cloud consumer and provider), which is known as Service Level Agreement (SLA). The SLAs specifies the technical criteria that must be fulfilled by the cloud provider for the quality of service, security and privacy issues, service performance, etc. Furthermore, the obligations and limitations that the cloud consumer has should be defined. Finally, the consumer may be billed for the usage of the services.
- *Cloud Auditor* is a party that can perform an independent assessment of cloud services, systems operations, performance, security, privacy and so on. The duty of the cloud auditor is to control and verify the conformance of standards, and for security, auditing should include the verification of the compliance with standards and security policies.
- *Cloud Broker* is an entity that manages the use, performance and delivery of cloud services and negotiates relationships between cloud providers and cloud consumers. The core services offered by cloud brokers are [10] [11]: a) *Service Intermediation*, a cloud broker enhances a given service by improving some specific capability and provides the value-added service to cloud consumers; b) *Service Aggregation*, a cloud broker combines and integrates multiple services into one or more new services. c) *Service Arbitrage* is similar to service aggregation except that the services being aggregated are not fixed.
- *Cloud Carrier* acts as an intermediary that provides connectivity and transport of cloud services between cloud providers and cloud consumers.

2.3. Virtualization

Virtualization is a technology that divides computing resources to enable many operating environments like hardware and software partitioning, machine simulation, emulation, time-sharing and so on. This technology covers a wide range of areas as server consolidation, supporting multiple operating systems, secure computing platforms, kernel development, system migration, etc. [12]. The cloud computing services are located in the data center, where each data center has thousands

of physical machines that need to serve many users and keep many applications, and then in such cases, the hardware virtualization is very useful and a perfect fit.

Depending on the type of application use, virtualization can be categorized as follows [13]:

- *Hardware virtualization (or server virtualization)* enables consolidation of workloads of multiple underutilized machines to fewer machines, so that the hardware and all infrastructure is utilized in the most optimal way. The subcategories of the hardware virtualization are full virtualization, emulation virtualization, and paravirtualization.
- *Software virtualization* enables the creation of multiple virtual environments inside the physical machine. The subcategories of the software virtualization are operating system virtualization, application virtualization and service virtualization.
- *Memory virtualization aggregates* physical memory across different physical machines into a single virtualized memory pool. The subcategories of the memory virtualization are application level control and operating system level control.
- *Storage virtualization allows* multiple physical storage devices operating as a common group to appear as a single storage device. The methods of this virtualization model can be block virtualization or file virtualization.
- *Data virtualization organizes* the data as an abstraction layer, which acts is independent from data structure and database systems.
- *Network virtualization* enables the creation of multiple network segments on the same physical network. This enhances performance and security in communication and data exchange. This can be accomplished both in the internal and external network.

To illustrate virtualization technology, in the Figure 2.7 we have presented a typical hardware virtualization scheme. Hardware virtualization allows running multiple operating systems and software stacks on a single physical platform [6].

Each virtual machine (VM) used to be an instance of the physical machine so that the user gives the impression of accessing the physical machine directly.

A software known as a Virtual Machine Monitor (VMM) or Hypervisor facilitates access to the virtual machine and the physical machine.

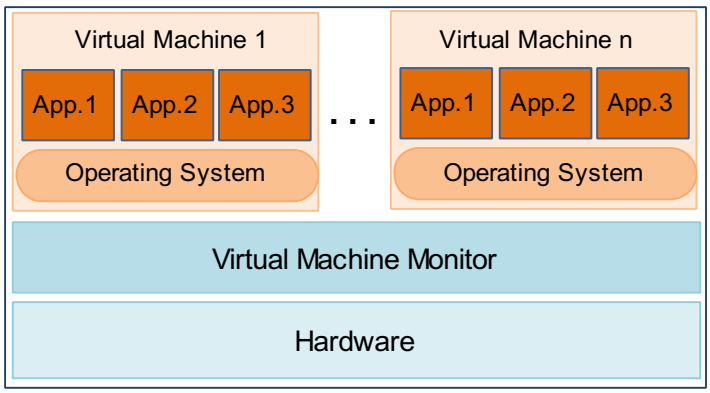


Figure 2.7: A virtualized physical machine

Some VMM platforms that are very powerful in cloud computing environments are:

- *Xen* is a hypervisor that enables to execute a multiple computer operating system on the same computer hardware concurrently. Xen allows the user to instantiate the operating system and perform operations that they want [14]. Xen has served as a base for other open source and commercial hypervisors.
- *KVM,* the kernel-based virtual machine (KVM), is a full virtualization solution for Linux. The KVM supports several guests operating systems, including Linux distributions, Microsoft Windows, OpenBSD, OpenSolaris, FreeBSD, etc. [15].
- *VMWare ESXi* is a hypervisor developed by VMWare that enables advanced virtualization functions and operations in relation to processor, memory and I/O [6].

2.4. Service Level Agreements (SLAs)

One of the main objectives of a cloud provider is to provide QoS requirements to meet customer expectations. Each service is typically accompanied by a service-level agreement (SLA), which defines the service guarantees that a cloud provider offers to its customers [59].

The components of a typical SLA are [60]:

- *Purpose* describes the reason for the creating SLA.
- *Parties* describes the parties involved in the SLA and their roles.
- *Validity period* defines the SLA validity period by specifying the start and end time.
- *Scope* defines the types of services covered by the agreement.

- *Restrictions* defines the essential steps to be done in order to supply the required service levels.
- *Service-level objectives* are approved by service providers and customers. They contain a group of service level indicators like availability, performance and reliability.
- *Service-level indicators* are the base level indicators which are used to measure these levels of service.
- *Penalties* define the situation when the provider cannot achieve the goals in the SLA. If the SLA is taken by an external provider, there should be an option to terminate the contract.
- *Optional services* provide for any services that are not solely requested by the customer but might be required as an exception.
- *Exclusions* specifies what is not covered in the SLA.
- *Administration* defines the procedures formed in the SLA to meet and measure its objectives.

SLA has six main stages to be completed. These stages are shown in Figure 2.8.

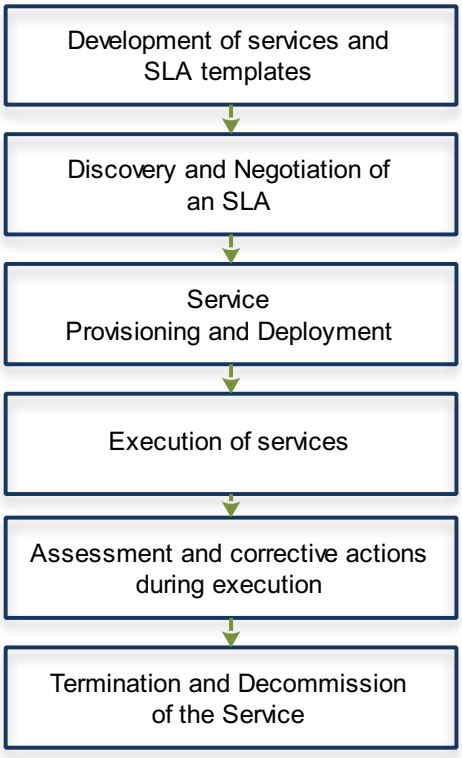


Figure 2.8: SLA life cycle [60]

2.5. Dynamic Resource Allocation in Cloud Computing

In general, resource management in a cloud infrastructure is a complex issue, due to the scale of modern data centers, taking into account the heterogeneity of the resource types and functions, interdependencies among the resources and the variability and unpredictability of the workload [16]. Resource management means the process of allocating computing, storage, networking and power resources to meet the performance objectives in relation to the services that should be offered to consumers by the cloud providers.

The most interesting issues in resource management in cloud computing include allocation of resources, resource utilization through virtualization technology, local and global scheduling of cloud services, workload management, scaling and provisioning, performance and optimization, resource pricing, etc. As well, an important term is resource demand that is defined as the minimum amount of hardware resources that are required for an application to be executed to meet the requirements for Quality of Service (QoS). Therefore, to predict the amount of resources for an application should be made a resource demand estimation [17].

In cloud computing, various consumers demand a variety of services and their requirements and needs may change over time. On the other side, from the cloud provider's perspective, the cloud resources must be allocated a fair and efficient manner; therefore, this is a great challenge to meet the needs of all cloud consumers and QoS requirements.

A typical resource allocation system in the data center is shown in Figure 2.9. A data center consists of a large number of PMs, where each PM runs a Virtual Machine Monitor (VMM), and one or more VMs. Each VM runs an application or an application component. Each PM communicates with the *data center manager* [118].

In most data center management systems, the main components are *controller*, *monitoring engine*, *predictor* and *migration manager*. The *monitoring engine* continuously collects data for each PM through the *controller* on the state of CPU, memory, network and other resources. The processed data and statistics pass to the *migration manager*. The *migration manager* uses information and statistics received and through the VM consolidation techniques determines the migrations to be performed. Changes are then made to the data center configuration through the *controller*. The *predictor* as another component of the data center manager predicts the future workload, which

facilitates decision making of the *migration manager* to generate better configuration for the data center.

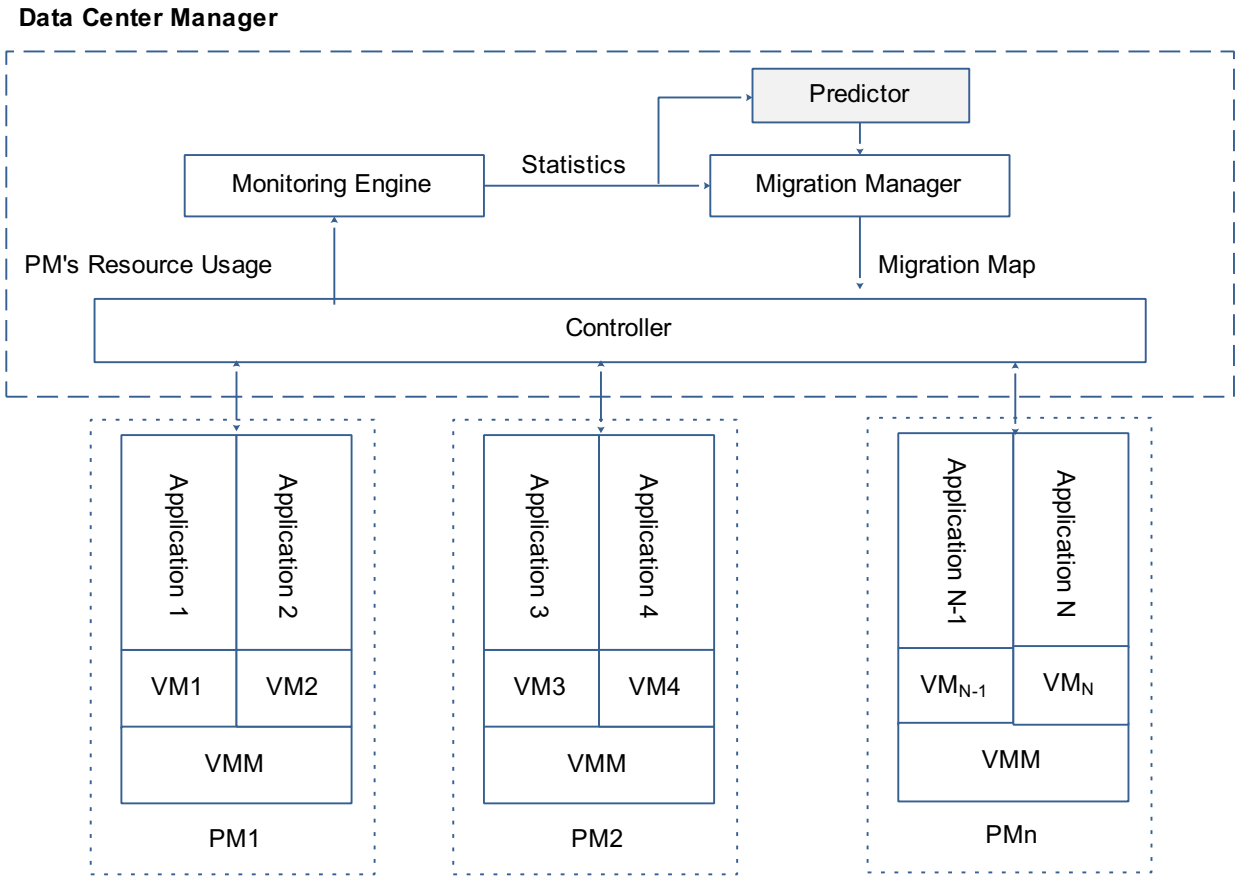


Figure 2.9: A system model for resource allocation in a data center [118]

2.5.1 Virtual Machine Consolidation (VMC)

In a cloud infrastructure, dynamic resource allocation is the process of dynamically assigning resources to the cloud applications according to workload demand. In the Infrastructure as a Service (IaaS) cloud computing service model, resources are allocated in the form of Virtual Machines (VMs) that can be resized and live migrated at runtime. To satisfy the demand of users, the cloud providers should manage resources efficiently.

VM consolidation in a data center is a complex but important process to increase the overall utilization of physical resources and directly increases the quality of services. The goal is to increase energy efficiency by consolidating VMs into the minimum number of PMs and switching idle PMs into a power saving mode. During the consolidation process, the performance of applications based on

QoS is also taken into account, which is predefined in the SLA between the consumer and the provider.

In Figure 2.10, we present an example of VM consolidation where some of the under-utilized PMs could be released and thus save energy in data centers [120].

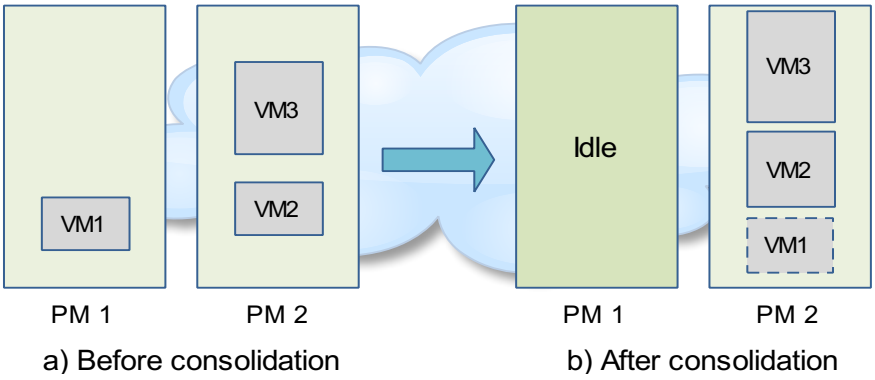


Figure 2.10: An example of VM consolidation

The objective of VMC is to place more VMs into a less number of PMs by increasing the utilization of resources. Two metrics are defined to measure the resource utilization ratio for PMs and in the overall Data Center (DC); *resource utilization ratio of the PM*, R_{PM_i} (Equation 2.1), and the *mean resource utilization ratio of DC*, $\overline{R}_{DC}$. (Equation 2.2) [117].

$$R_{PM_i} = \frac{\text{Utilized Amount of Resource of } PM_i}{\text{Total Amount Resource of } PM_i} \tag{2.1}$$

$$\overline{R}_{DC} = \frac{1}{N}\sum_{i=1}^{N} R_{PM_i} \tag{2.2}$$

N is the total number of active PMs in DC.

To keep under control the use of resources, physical and virtual machines in a data center should be monitored periodically.

One of the important mechanisms for dynamic workload management in cloud infrastructures is live migration of VMs from one PM to another. Live migration of VMs offers the possibility for allocation of resources to running services without interruption during migration process that is important for services with particular Quality of Service (QoS) requirements [18].

In general, live migration of VMs has several benefits including:

- *Load balancing* provides high throughput and availability.
- *Manageability and maintenance*, movements of virtual machines and shutdown of hosts for maintenance.
- *Minimum violation of SLA (Service Level Agreement)*, meeting the SLA requirements between cloud providers and cloud users.
- *Energy management*, consolidation of virtual machines, switch off underutilized servers to reduce data center's heat loss and power consumption.
- *Improved performance and reliability*, the application performance will not be degraded.
- *Improving the utilization of resources.*
- *Reducing management costs.*

Depending on the workload and environmental conditions, a single or a multiple VMs can be migrated. Three kinds of migration are generally categorized:

- **Single VM migration**, where only one VM migrates at a time.
- **Multiple VM migration**, where two or more VMs are migrated simultaneously.
- **Single and multiple VM migration,** where one or more VMs are migrated simultaneously.

Dynamic consolidation of VMs based on a proactive framework can be divided into the following domains [129]:

(a) **Workload Prediction** consists of a clustering process, predicting the window size, evaluating the VM and user behaviour, and predicting the entire process chain as part of this domain. This includes various strategies, such as workload estimation and scheduling, dynamic provisioning and admission control, which estimate and determine whether resources should be added or removed, whether the order of the query execution should be rearranged, or whether a new incoming request should be allowed or rejected [130]. Predicting a resource's workload for a short or long-time interval is a fundamental process in dynamic resource allocation and capacity planning in a data center. Proper and timely resource planning leads to increased service performance and is concentrated on energy saving on a data center. This process is made even more difficult by the fact that in a cloud environment, it is a challenge in itself to predict the demand for all types of resources, knowing that there are different VM

requests with different numbers and types such as processors, memory, storage, and network bandwidth.

Figure 2.11 illustrates one of the proactive dynamic VM consolidation frameworks.

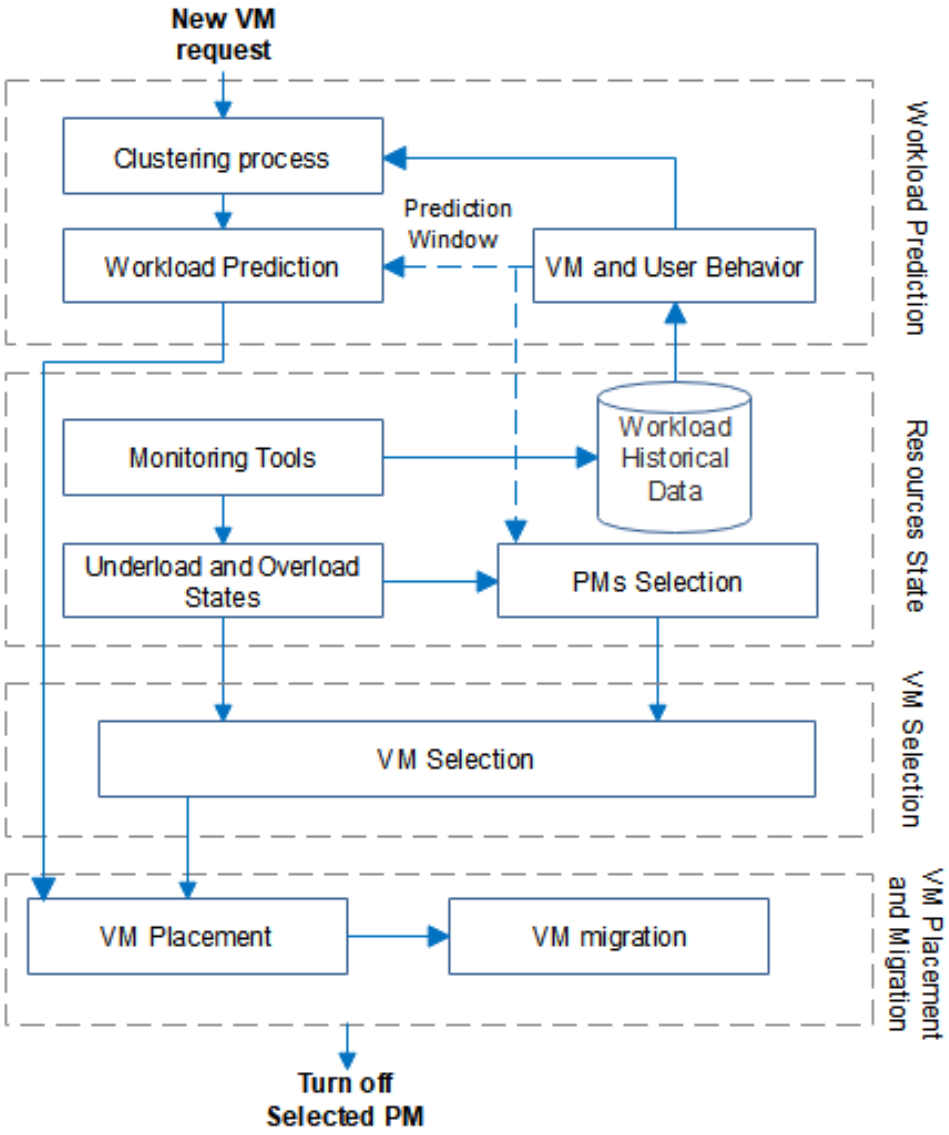


Figure 2.11: Proactive dynamic VM framework [129]

The components of the workload prediction domain are [129-130]:

- ***Clustering process***. During this stage, clustering techniques are used to enable efficient dynamic VM consolidation, where depending on the requests received, requests are shared in the appropriate clusters with different types of VMs. In the literature, there are different clustering algorithms and techniques that are used for the dynamic VM consolidation process.

- ***Prediction process***. To predict the future demand for resources based on historical workload data, accurate algorithms and techniques are needed. Several prediction techniques and algorithms are used, depending on the perspective of the researchers, but the most widely used are machine learning (ML) techniques [132].
- ***Prediction window size***. At this stage, the calculation of the time interval in the future is determined, for which the resource workload should be predicted. Based on this it can be decided whether a PM should switch to sleep mode, thus reducing energy consumption. In this case, the prediction process is highly dependent on the configuration mode at the data center with special emphasis on PMs' hardware. Depending on the prediction window size, the prediction and clustering techniques must also be defined and observed.
- ***VM and user behaviour***. This component analyzes the behavior of the user and the VM in real time when requesting the allocation of VM resources. Also, the analyses of the relationships and dependencies between users and VMs improves the overall prediction process, where based on comparative conditions unwanted users and VMs are excluded from the future step of the workload estimation process [131].

(b) **Resource State**. This is the state of all virtual and physical resources. This includes monitoring and tracking tools, techniques and algorithms that detect if a PM is overloaded or underloaded, and then facilitates the PM selection phase.

More specifically, the components within the resource state domain are:

- ***Monitoring tools***. The monitoring process in a data center facilitates the continuous monitoring of physical components and the accompanying infrastructure, as well as the measurement of performance in case of access to applications, in order to maintain a high level of reliability and quality of services. Thus, in the process of dynamic VM consolidation, monitoring tools provide the information about the state of the PMs and VMs, by analyzing the data generated by monitoring, select the main parameters that affect the reduction of the computation load, and also select to switch on or off a suitable PM.
- ***Overloaded and under-loaded PM detection***. The number of PMs in a data center is large and each PM has its VMs that execute different applications. Thus, it is a challenge to measure their loads.

To detect if any of the PMs is overloaded or underloaded, a continuous monitoring process by analyzing historical data of applications workload and resources usage should be provided. Based on historical data, the PM's workload several steps (some time intervals) in the future should be predicted and based on this the cloud provider should make a decision for the migration process. The ideal case to be achieved is that, based on long-term workload predictions, a decision for live migration of VMs before PM overload or underload happens is to be made. After that, it should be detected which of the PMs is overloaded or underloaded. Then, it must be decided which of its VMs must be migrated to other PMs.

The dynamic VM consolidation process is illustrated in the Figure 2.12 [120].

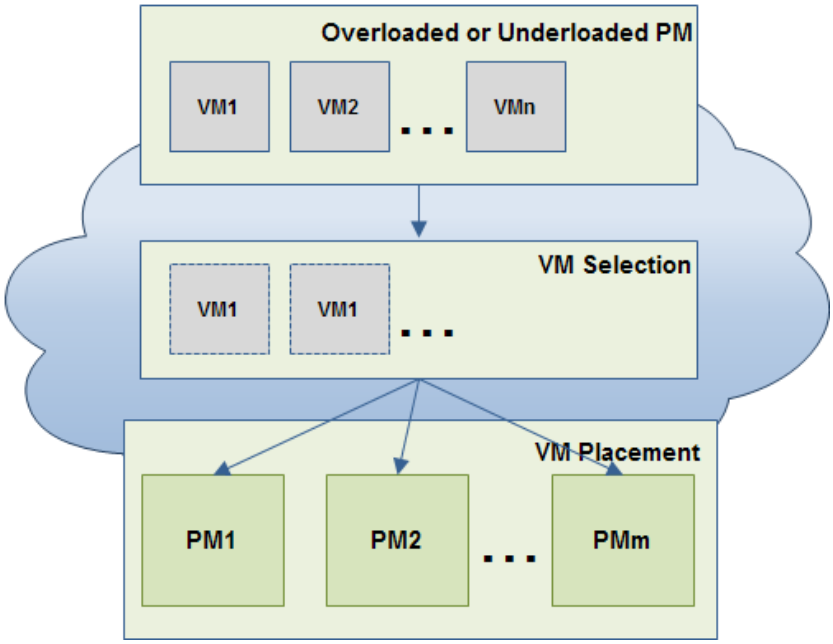


Figure 2.12: VM consolidation stages

(c) **VM Selection**. Another challenge in VM consolidation is which of the VMs from the overloaded or underloaded PM will migrate to other PMs. Potentially, there can be more than one VM that will be selected for migration. A suitable VM should be selected based on a monitoring process through the VMM (Virtual Machine Monitor) to convey the applications workload and the resource usage within the VM.

(d) **Destination PM Selection and VM Placement**. After selection of suitable VMs (one or more), another issue is to target a suitable destination PM. Potentially, more than one PM can be a destination for migrated VMs. Therefore, the state of PMs should be monitored to see if they

are overloaded or underloaded. Also, the state of the destination PM should be predicted exactly after VM placement in order to ensure that is not overloaded.

To explain all steps of VMC based on live migration, we present a flowchart in Figure 2.13 [120].

- ***Overloaded and Underloaded PM Detection.*** *A* major challenge for live migration of VMs is detecting when a PM is over-loaded or under-loaded. Problematic is the selection of the overload utilization threshold. Due to unpredictable and dynamic workload, a static overload utilization threshold is not suitable.

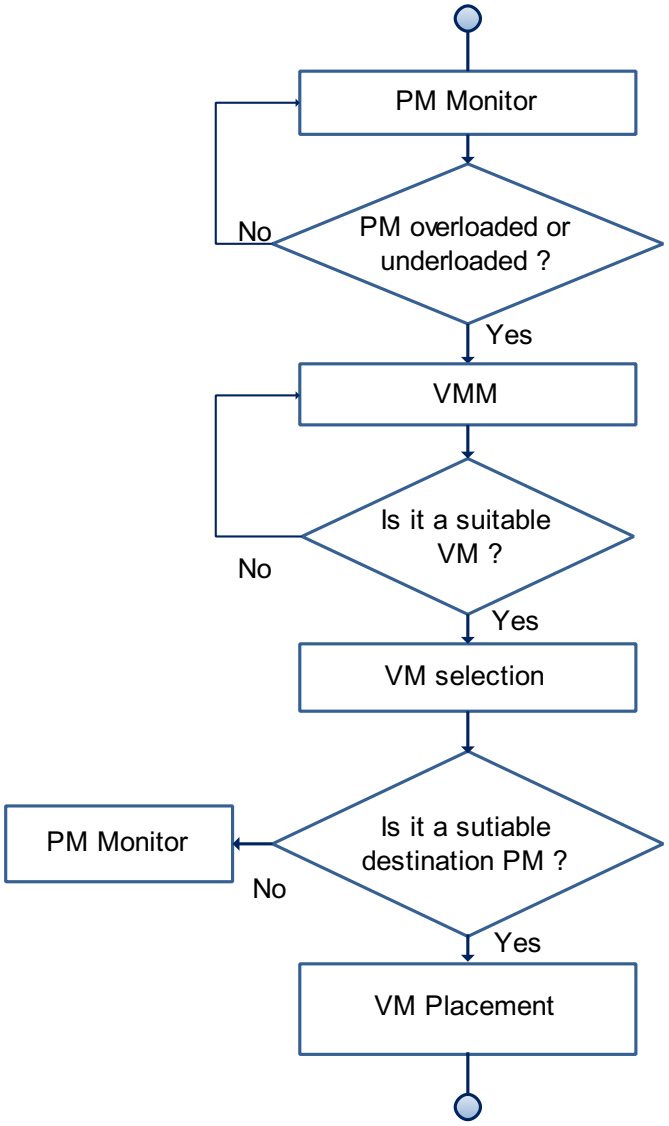


Figure 2.13: VM consolidation flowchart

In principle, a PM is considered as overloaded when during the resource usage monitoring process, the actual and predicted next value exceeds a specified upper utilization threshold. As well, a PM is

considered under-loaded then when actual and predicted next value exceeds a specified lower utilization threshold.

A heuristic approach for setting an upper and lower utilization threshold is proposed in [19].

Below are some techniques to identify the source PM selection and to detect if the PM is in an overloaded state [20-23].

- *Median Absolute Deviation (MAD)* uses median absolute deviation to assign an upper threshold of a PM to be marked as overloaded. The MAD is a measure of statistical dispersion, a robust statistic, being more resilient to outliers in a data set than the standard deviation.
- *Inter Quartile Range (IQR)* uses the interquartile range to decide the threshold of a PM to be marked as overloaded. For symmetric distribution, half of IQR is equal to MAD.
- *Threshold (TH)* provides the utilization when a PM must qualify as overloaded.
- *Local Regression (LR)* is based on the Loess method [21]. Local regression builds a curve that approximates original data by setting up the simple models to localized subset of data.
- *Local Robust Regression (LRR)* provides prediction for PM underload that is proposed as robust estimation method known as bisquare [24] that transform LR into iterative method.

- ***VM Selection.*** When a physical machine has more than one virtual machine running on it, then the challenge is how to select the virtual machine (VM) for migration when a physical machine is overloaded or under-loaded. In this case, a VM selection policy is needed that efficiently react in relation with assigned utilization threshold (for upper and lower threshold). Below some of the most common techniques for the process of VM selection are presented.
  - *Maximum Correlation (MC):* By the MC technique, the VM that has the maximum correlation coefficient compared to other VMs which are located on the same PM [25] [23] is migrated. This method is proposed by Verma at al. [26], based on the idea that the higher the correlation between the resource usages by applications running on an oversubscribed PM, the higher the probability of PM being overloaded.

- *Minimum Migration Time (MMT):* The MMT technique selects the VM that has the least memory, since it will be migrated faster [22]. The migration time is estimated as the amount of memory utilized by VMs divided by the network bandwidth available for a host [21].
- *Random Selection (RS):* The RS selects a VM for migration randomly from the VMs residing in the source PM [19].
- *Constant Fixed Selection (CFS):* CFS is similar to the Random Selection (RS) strategy. CPS selects from the VM list those that are in the first, middle, or last position to leave the overloaded PM [133].
- *High Potential Growth (HPG):* When the upper threshold is violated, the HPG policy migrates VMs that have the lowest usage of CPU depending on the CPU capacity that is defined by the VM parameters. This affects to minimize the potential increase of the PM's utilization and prevent an SLA violation [19].
- *Minimization of Migrations (MM):* This policy selects the minimum number of VMs needed to migrate from a PM in order to lower CPU utilization below the upper utilization threshold if the upper threshold is violated [19].
- *Minimum Utilization (MU):* This VM selection policy selects VMs that have the lowest CPU utilization, in order to reduce the processing overhead [25].
- *Multi-objective optimization:* This policy is suitable in dynamic environments. It is based on a multi-objective model where during the VM selection process it takes into account CPU parameters, such as temperature, resource use, and power consumption [134].
- *Fuzzy Q-Learning (FQL)*: Since the process of VM selection is a decision-making problem, this policy uses fuzzy logic, which integrates several VM selection criteria which then dynamically select the most suitable VM selection approach. Thus, this policy tends to choose a more optimal strategy that should be used in the VM selection process [135].
- *Fuzzy VM selection*: This VM selection policy is based on machine learning techniques, to select VMs from an overloaded PM. This policy integrates the migration control algorithm with the fuzzy logic VM selection strategy [136].

- ***Destination PM Selection / VM Placement:*** Another issue in dynamic VM consolidation after we have selected the right VM for migration is to select in which PM to place. In the VM placement phase the destination PM needs to analyse carefully whether it will be overloaded after the migration process. Many authors see the destination PM selection and VM placement as a bin-packing problem.

  The VM placement algorithms generally can be categorized into the following types [27]:

  (a) *Power-based*, the chosen VM migration to the target host should result in a system that is energy-efficient with utmost resource utilization.

  (b) *QoS-based*, the chosen VM migration to the target PM should ensure maximal fulfilment of quality of service requirements.

  Based on the literature, there are also more destination PM selection and VM placement schemes for the cloud infrastructure layer. Depending on the methods, they use some of VM placement schemes that can be classified as [28]: Graph theory-based, Genetic Algorithm-based, Automata-based, Greedy Algorithm-based, constraint programming-based, integer programming-based, ACO-based (Ant Colony), PSO-based (Particle Swarm Optimization), etc. Below some of the most common heuristic-based techniques for the destination PM selection and VM placement are presented.

  - *Random Choice (RC):* The RC policy randomly selects as PM that is available to migrate the selected VM. If there is no PM available for VM migration, then a new PM will start from a sleep state [119].
  - *First Fit (FF):* The available PMs are placed sequentially in an ordered list and for each VM that requires a destination PM then the first PM is selected from the list. If the first PM cannot accommodate a VM, then a second PM is checked, and it continues to other PMs until one of them has enough resource capacity to accommodate the VM [119].
  - *First Fit Decreasing (FFD):* In this policy, the VMs are sorted in decreasing order based on resource demand and for the VM with the highest resource demand using the First Fit policy will be required the destination PM [117].
  - *Next Fit (NF) / Round Robin (RR):* This is similar to the FF policy, but the NF or RR policy does not start the search of the destination PM from the first PM in the ordered list, but starts from the last PM selected in the previous VM placement [117, 119].

- *Best Fit (BF):* In this policy, the destination PM with the minimum residual resource is selected.
- *Best Fit Decreasing (BFD):* BFD is another heuristic policy where it first sorts the VMs in decreasing order based on resource demands and then allocates the VMs to the PM with resources that meet the VM requirements [118].
- *Power Aware Best Fit Decreasing Algorithm (PABFD):* This policy [19] first sorts the VMs according to their CPU utilization in descending order and then allocates VMs to the PM that provides the least increase of the power consumption.
- *Minimum Correlation Coefficient (MCC)*: In this technique, the correlation coefficient is used to represent the degree of association between a chosen VM and the target host [25]. During the chosen VM migration to the target host, if the correlation coefficient will increase, then MCC will increase the impact on the performance of the others VMs. Hence, to avoid performance degradation on others VMs a chosen VM will be migrated with the minimum correlation coefficient.
- *Heuristics, fuzzy logic and migration controls* [22] [29]: This approach combines the heuristics and migration controls for VM consolidation. The key metrics that are investigated are: SLA violation, number of VM migrations, and energy consumption.
- *Utility functions* [30]: This model addresses the VM placement problem by using utility functions, to maximize the profit of VM placement by minimizing energy consumption and SLA violations. The model is divided into the following categories: input, processing, and output.

In general, VMC techniques are classified into two major groups: Dynamic VMC (DVMC) Techniques and Static VMC (SVMC) techniques [117-118].

- *DVMC Techniques:* Based on workload variations as well as on specific time intervals, consolidation algorithms should make the decision to migrate VMs to the appropriate PMs. DVMC algorithms ensure the reallocating of VMs to a smaller number of PMs with the intent to reduce the number of active PMs.

- *SVMC Techniques:* In these consolidation techniques, the VM-to-PM mappings are not changed for a long time and if the workload changes during this time, no migration is

performed. SVMC algorithms that enable the solution of initial VM placement in the minimum number of active PMs will increase the energy-efficiency and resource utilization of the data center. These techniques do not provide the reallocation of VMs in new PMs considering current VM-to-PM mapping.

Figure 2.14 presents an overview of VMC techniques classification [117].

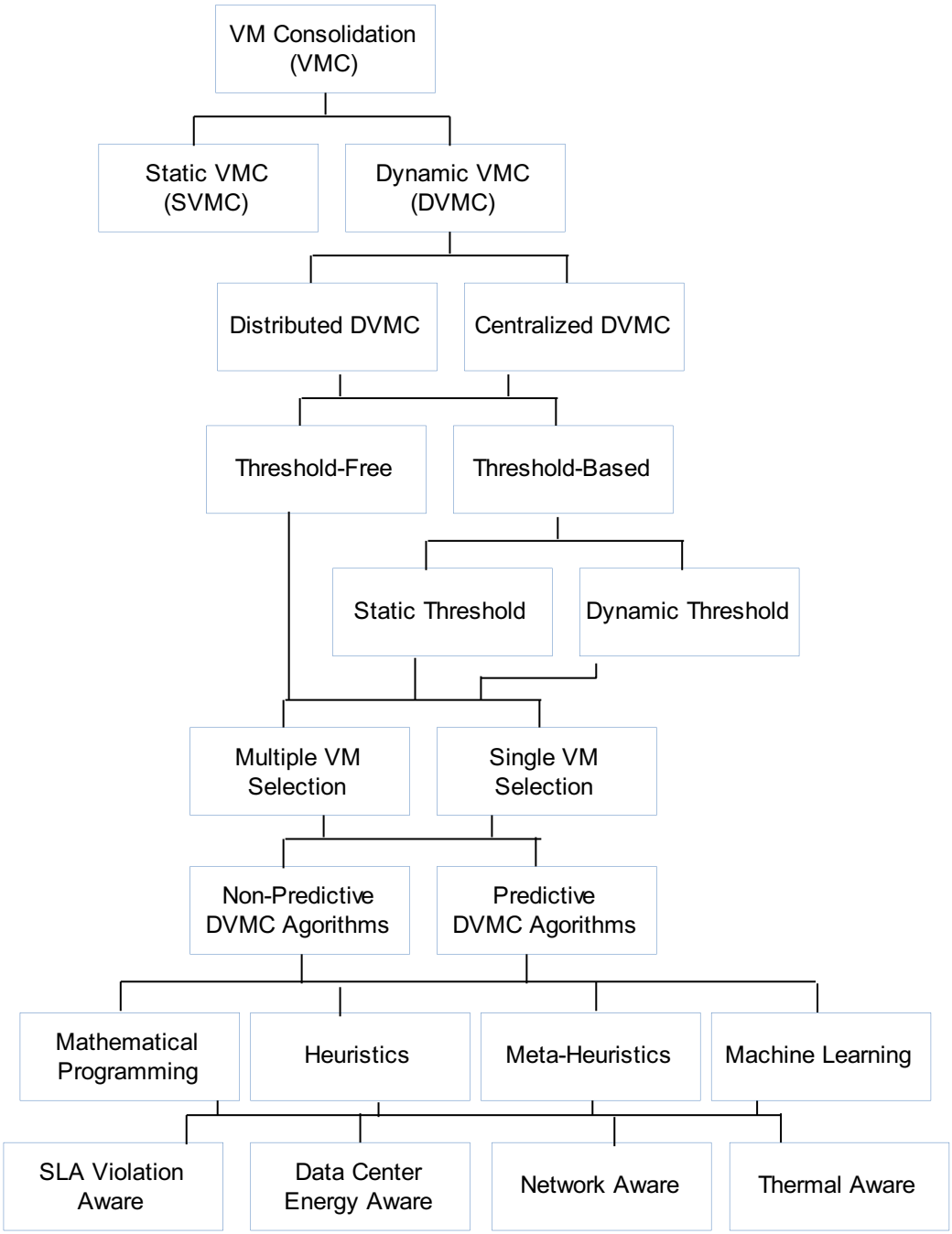


Figure 2.14: An overview of VMC techniques

DVMC algorithms through the live migration mechanism enable the running VM to migrate from one PM to another while it is running and providing service to its consumers.

DVMC techniques can be classified into two groups:

- *Centralized DVMC Techniques*: In centralized architectures, there is only a central controller that keeps the information on the capacity of available resources of all the PMs. The controller runs the centralized VMC algorithm that selects a destination PM for the selected VM migration, considering the resource availability of all PMs [117].
- *Distributed DVMC Techniques*: Unlike centralized architectures that have a centralized controller, in distributed architectures the PMs exchange information about their resource availability with their own neighbour PMs and vice versa. If a PM wants to migrate any of its VMs then it executes distributed VMC algorithm to select one of the neighbouring PMs as a destination PM to place the migrated VM.

To identify whether the source PM is an overloaded or underloaded state, there are two groups of DVMC techniques that tackle this issue:

- *Threshold-Based DVMC Techniques*: These algorithms are used to detect whether a PM is an overloaded or underloaded state upper and lower threshold values. In this case, the resource utilization ratio (Equation 2.1) of the PM_i should be compared with some static or adaptive threshold values, and if the resource utilization ratio exceeds the upper utilization threshold value, then the PM_i is considered as overloaded or over-utilized, and VMs of PM_i should be migrated out. On the other hand, if the resource utilization ratio is below the lower utilization threshold value, then the PM_i is considered as underloaded or under-utilized, and VMs of PM_i should be migrated out, so that PM_i to go into sleep mode [19].
- *Threshold-Free DVMC Techniques*: In threshold-free techniques, the resource utilization ratio of the PM is not comparable to any threshold value to detect if the PM as an overloaded or underloaded state [117]. In this case, the source PMs are selected randomly or depending on the algorithm some functions are applied that give priority to any PM in relation to others, based on the lower or higher resource utilization ratio.

From various research works related to DVMC techniques, threshold based DVMC techniques are classified into two groups [117]:

- *Static Threshold-Based Techniques*: To identify whether a PM as an overloaded or underloaded state, fixed or predefined values are used as upper and lower thresholds. Threshold values do not change over time, therefore referred to as static thresholds [117].
- *Dynamic Threshold-Based Techniques*: In these algorithms, to identify whether a PM as an overloaded or underloaded state, the values of the thresholds are dynamically assigned as the resource utilization ratio of the PM changes over time. Actually, the threshold values adapted depending on the changes in resource utilization [21].

In relation to the VM selection policy, the DVMC techniques are classified into two main categories:

- *Multiple/Clustered VM Selection*: In cases of multi-layered applications, where an application is located in one or more VMs, another application is in another VM, and then there is a functional dependency between applications and VMs. In the case of communication between applications in different VMs that are not hosted in nearby PMs, it may lead to communication difficulties and the degradation of application performance. In such cases, instead of migrating a single VM should be considered a group of VMs or clustered VMs for migration.
- *Single VM Selection*: In this category, the single VM selection algorithms select a single VM to migrate.

An important issue to be addressed in the dynamic consolidation of VMs is the future workload estimation of the PM. DVMC algorithms that make the decision for migration based on the prediction of future resource utilization show better performance compared to algorithms based on current resource utilization. In this sense, two groups of algorithms can be categories.

- *Predictive DVMC Algorithms*: These algorithms make the decision to migrate VMs from one PM to another based on the estimated future resource demand of VMs [117].
- *Non-predictive DVMC Algorithms*: The decision to migrate VMs from one PM to another based on the current resource demand of VMs [117].

To solve the problem of VM consolidation as a multi-objective optimization problem, different algorithms and approaches were proposed. Since it is known that VM consolidation is an NP-hard problem, it is not easy to find an optimal solution with a large number of PMs and VMs. The most

well-known algorithms that deal with VM consolidation problems are heuristics, meta-heuristics, mathematical programming and machine learning [63].

- **Heuristic algorithms** find the solution step-by-step by taking into consideration the best local decision. Some existing approaches to the VM consolidation problem have treated it as an optimization problem, to find a near optimal solution through heuristic algorithms. In this case, the heuristic algorithms can consolidate the workload in a multi-objective optimization problem. These algorithms are suitable to solve the VM consolidation as a bin-packing problem [128]. According to the bin-packing problem, each VM is considered an item and each PM is considered a bin.

  There are some heuristic algorithms that solve the bin-packing problem as they are: First-Fit (FF) algorithm, Best-Fit (BF) algorithm, Next-Fit (NF), Random-Fit (RF), First Fit Decreasing (FFD) and Best Fit Decreasing (BFD). As well, modified versions of these algorithms are used depending on the viewpoint of different authors.

- **Meta-heuristic algorithms** can compute near optimal solutions for complex multi-objective optimization problems. The well-known problems are ant colony optimization and genetic algorithms. Ant Colony Optimization (ACO) is a multi-agent approach (artificial ants) for complex combinatorial optimization problems, such as the Traveling Salesman Problem (TSP) and network routing [62]. In relation to VM consolidation problem, ant algorithms are used in terms of Ant System (AS), Max-Min AS (MMAS), and Ant Colony System (ACS).

  Genetic Algorithms (GA) have been shown to be successful in solving various optimization problems. GA is used to find the optimal solution to a given computational problem that maximizes or minimizes a particular problem. In addition, other well-known algorithms used to solve the VM consolidation problems are Simulated Annealing (SA), Particle Swarm Optimization (PSO), Tabu Search, and Hybrid Optimization algorithms.

- **Mathematical programming** uses a mathematical formulation to find an optimal solution by searching all the possible solutions. Well-known mathematical programming algorithms for VM consolidation are Stochastic Programming, Linear Programming (LP), Non-Linear Programming, Dynamic Programming, Constraint Programming (CP), Quadratic Programming, and Game Theory. Compared to heuristics and meta-heuristics, these algorithms provide good performance to compute the optimal solution.

- **Machine learning** is a computer science discipline that has the objective to develop learning capabilities in computer systems [64]. Several works use ML algorithms for workload prediction and power consumption modelling in a data center. The most commonly used ML techniques are Linear Regression (LR), K-Nearest Neighbour Regression (K-NNR), and Reinforcement Learning (RL).

2.5.2 Virtual Machine Live Migration Components

During the live migration process, it is important to define what should be migrated and which content should migrate. The migration process affects the CPU state, memory contents, and storage content.

- **CPU state**

  The CPU state of the VM needs to be context switched from one PM to another. This is a small amount of information to transfer and represents the lower bound to minimize the service downtime [122].

- **Memory Contents**

Memory migration is a process of moving the contents of VM's memory from one physical machine to another. The memory content that is subject to the migration process represents a larger amount of information, including the physical machine processes memory and guest OS memory within the virtual machine. The memory module that needs to migrate are [101]:

- *VM Configured Memory*: The amount of actual physical memory that is given to guest VM by the hypervisor.
- *Hypervisor Allocated Memory*: It is part of the VM configured memory, but with a smaller size then it does. If a VM tries to access this memory and free it, then the decision is taken from the hypervisor.
- *VM Used Memory:* It is the memory that is used by VM OS and all running processes. These memory pages keep track by the guest VM.
- *Application Requested Memory*: It is the amount of memory that is required for running an application, which is allocated by the guest VM OS.

- *Application Actively Dirtied Memory*: It is the part of the application requested memory, which is frequent access and modified by a running application.

The process of memory transfer can be divided into phases [102]:

- **Push phase:** The hypervisor transfers memory pages to the destination PM while VM on the source is still running. To maintain consistency, the pages that have been modified during the transfer process must be resent.
- **Stop-and-copy phase:** First, the source VM is stopped, pages are copied across to the destination VM and then start a new VM.
- **Pull phase:** If the new VM executes and tries to access a page that has not yet been copied, then this page is faulted in across the network from the source VM.

All the migration techniques try to reduce total migration time and down time. There are two main techniques in memory migrations: a) *pre-copy* and b) *post-copy*.

a) Pre-copy technique

At this stage, the pages are copied iteratively from the source to the destination PM while the VM continues to run. During the iteration procedure, some memory pages may be modified or dirtied, so they have to be re-sent to the destination PM in a future iteration [103] [126]. Then it is passed to the termination phase, which depends from the defined threshold, and if one of the three following conditions is met [101]: (1) the number of iterations exceeds pre-defined iterations, or (2) the total amount of memory that has been sent, or (3) the number of dirty pages in the previous round fall below the defined threshold.

Finally, in the stop-and-copy phase, the migration of the VM is suspended at the source host, after that move processors state and remaining dirty pages. When the VM migration process finishes correctly, then the hypervisor resumes the migrated VM on the destination PM. This technique is implemented in many kinds of hypervisors like Xen, VMware, and KVM.

The flowchart in Figure 2.15 illustrates the VM migration through pre-copy technique [101].

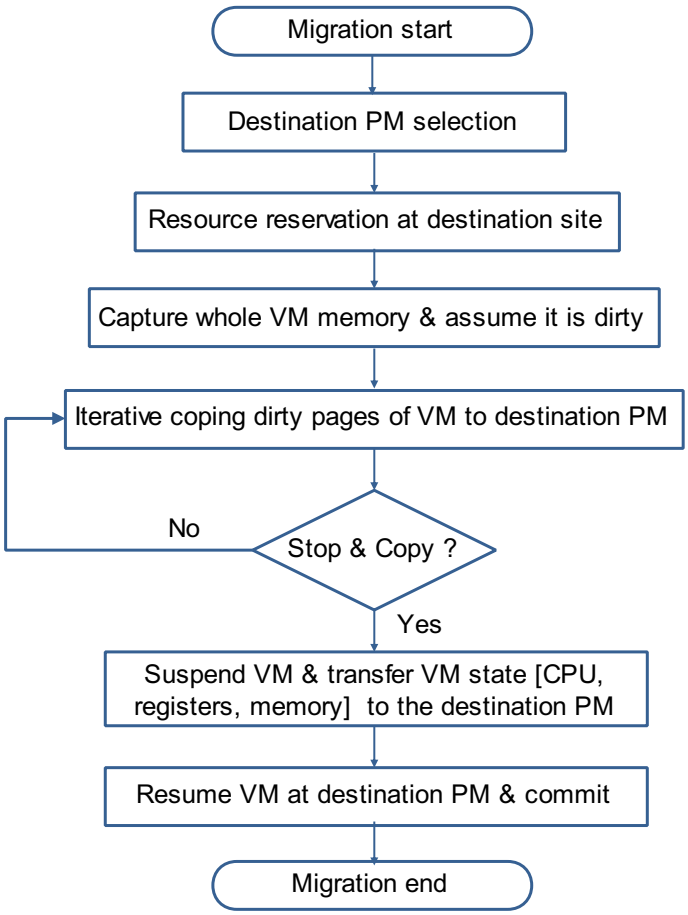


Figure 2.15: Pre-copy technique flowchart [101]

b) Post-copy technique

In the post-copy technique [104], the VM's memory content is transferred after the processor state has been sent to the destination host. According to this technique, first transmits all processor state to the destination, starts the VM at the destination, and then actively pushes memory pages from source to destination. Post-copy ensures that each memory page is transferred at most once, thus avoiding the duplicate transmission overhead pre-copy. Some modalities of the post-copy technique are as follows [101, 104] [126]:

- **Demand paging**: It ensures that each VM page is sent only once over the network, unlike in the pre-copy technique where dirtied pages can be resent multiple times. When VM resumes at the destination PM and requested memory pages results in page faults then the faulty pages are serviced by retransmission from the destination PM, so this leads to degradation of application performance. Therefore, demand paging provides the simplest and slowest variant.

- **Active Push**: In order to reduce the duration of residual dependency from the destination PM is to proactively push the VM pages from the source to the destination even the VM is running on the destination. If the page fault occurs at the destination VM then this situation is handled through the demand paging.
- **Pre-paging**: Through the pre-paging feature tends to avoid or mitigate the page fault rate and the reduction of major faults to be predicted in advance, and adapt the better page pushing sequence to access the VM's memory pages. This is done using the faulting addresses as hints to estimate the spatial locality of the VM's memory access pattern.
- **Dynamic Self-Ballooning (DSB):** DSB is a mechanism that is used to avoid the transfer of free memory pages. DSB enables the guest OS to reduce its memory footprint by releasing its free memory pages back to the hypervisor and this speedup the migration process. Therefore, the purpose is to avoid the sending of unused pages to the destination PM because this increases the total migration time.

The flowchart in Figure 2.16 illustrates the VM migration through the post-copy technique [101].

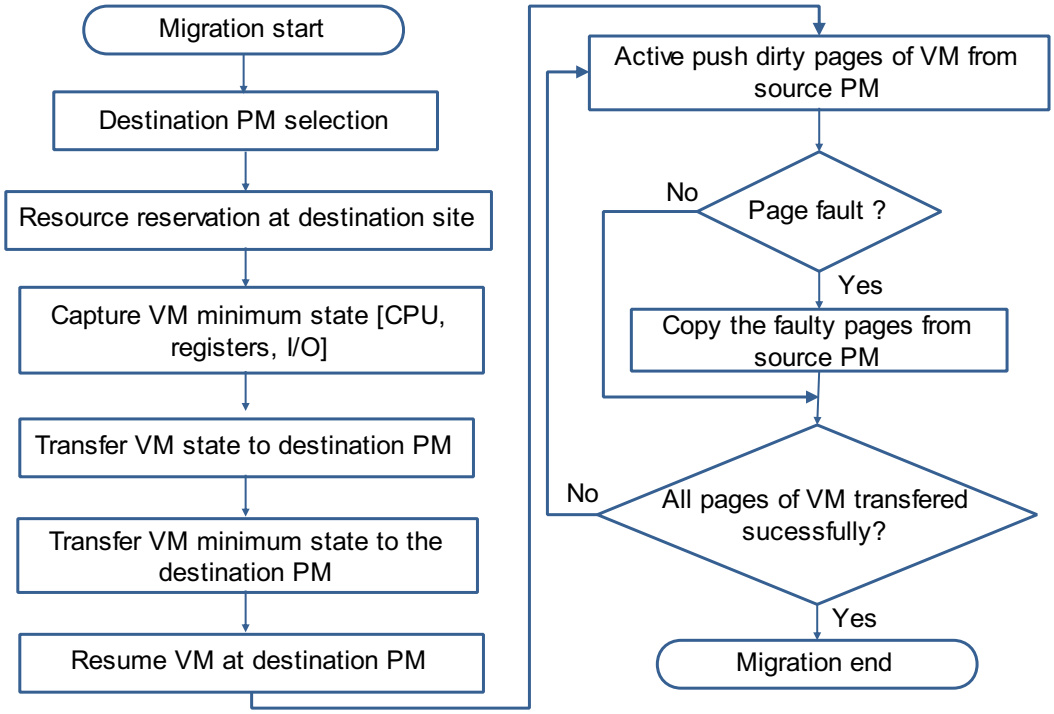


Figure 2.16: Post-copy technique flowchart [101]

c) Hybrid technique (pre- and post-copy)

The hybrid technique includes pre- and post-copy VM migration techniques with the intention to improve the total migration time and service downtime, as the most important parameters in the overall performance of the migration process. According to this approach [105] [126], in the first iteration, it works as a pre-copy technique while the VM is running on source PM. After the first iteration of the memory transfer, the VM is stopped and then resumes at the destination PM with its processor state and dirty pages. Then the remaining pages are transferred through the post-copy technique.

The flowchart in Figure 2.17 illustrates the VM migration through the hybrid technique [101].

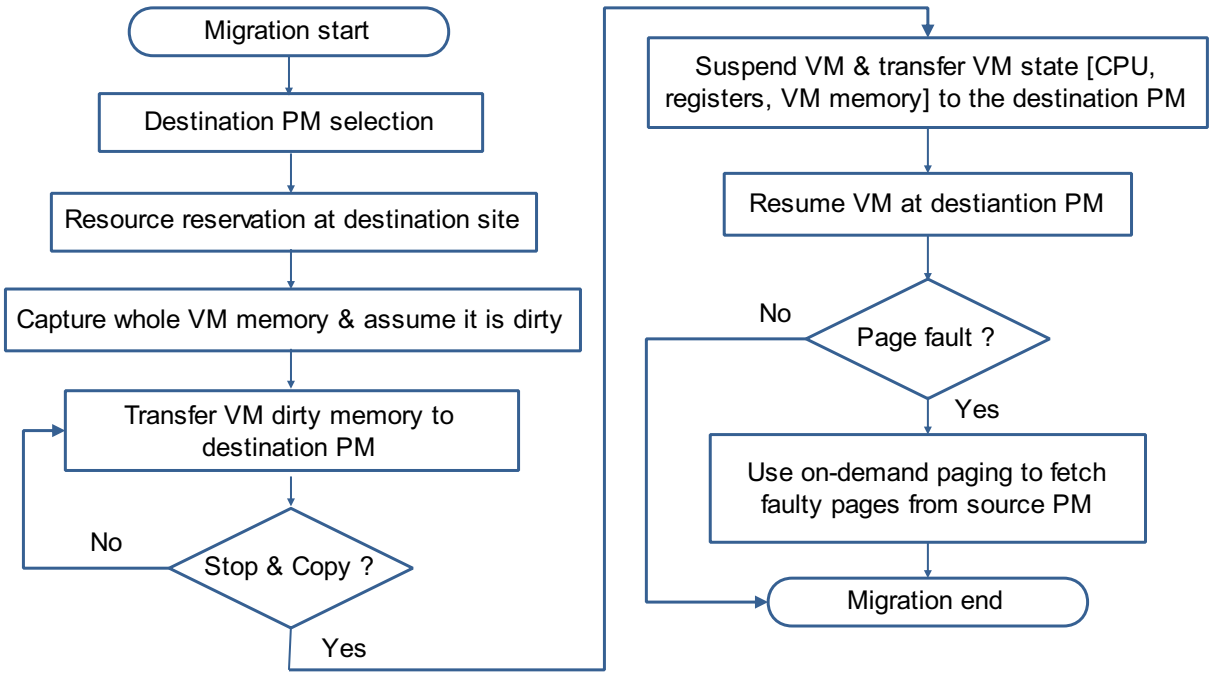


Figure 2.17: Hybrid technique flowchart [101]

- **Storage Content**

Storage content represents a large amount of information need to be transferred from the source to destination. The transfer of full disk image over the network takes a considerable time, so to reduce the transfer time and to avoid them transfer the hypervisor first identify the unnecessary contents and unused space. This reduces the migration time in total. The storage content that needs to be migrated is [101]:

- *Virtual Disk Size*: represents the disk size that is allocated for VM use, which is assigned when the VM is created.
- *VM Used Blocks*: are the system and user data blocks, which are stored in a VM image. These blocks are accessed and used by the guest VM OS.
- *Hypervisor Allocated Blocks*: represents the space allocated by the hypervisor to VM for data storage. The size of this space may be same as virtual disk size if pre-allocation is performed.

- **File System Migration**

To facilitate the migration process of VMs, the system should ensure that each VM with a consistent, location independent view of the file system that is available on all PMs [126]. A way to do this is to provide each VM with its own virtual disk, to which the file system is mapped, and transport the contents of this virtual disk along with the other states of the VM. Although seeing that in today's trends the capacity of the disks is higher, the migration of the contents of an entire disk over the network is not a proper choice. Another solution could be to have a global file system on each machine where e VM could be located. This excludes the possibility of copying files from one machine to another, while all files may be accessible through the network. Moreover, it is impractical to ensure a consistent global root file system across all machines [106].

2.6. Performance Metrics

Despite the fact that the VM live migration process has great benefits in data centers, it should not be ignored even its cost such as performance loss and energy overhead. This cost is also critical for businesses to achieve their goals.

To increase efficiency in VM consolidation and quality of services at an acceptable level, the overall performance should be evaluated through several metrics, like energy consumption, number of VM migrations, Service Level Agreement violations, performance degradation due to migration, energy consumption and SLA.

The live VM migration performance can be measured by the following metrics:

1. **Downtime:** This metric represents the time when a service is not available during the processor states migration process. The downtime metric T_{down} depends on the page dirty

rate *D*, page size *L*, duration T_n of the last pre-copy round *n*, and link speed *B*, and is defined as in Equation (2.1) [101], [103], [107]:

$$T_{down} = \frac{D \cdot L \cdot T_n}{B} \tag{2.1}$$

2. **Pages Transferred:** This metric indicates the number of pages transferred and duplicate pages during VM migration [101], [103], [107]. For a round *i*, the page transferred is calculated as in Equation (2.2):

$$V_i = \begin{cases} V_{mem}, & \text{if } i = 0; \\ D \cdot T_{i-1}, & \text{otherwise.} \end{cases} \tag{2.2}$$

where V_{mem} is the amount of VM memory; T_{i-1} is the time taken to migrate dirty memory pages, which is generated during just previous round.

The *elapsed time* of VM migration T_i at each round is defined in Equation (2.3):

$$T_i = \frac{V_{mem} \cdot D^i}{R^{i+1}} \tag{2.3}$$

Where *R* is memory transmission rate during VM migration.

The *network traffic* V_{mig} during VM migration is defined in Equation (2.4):

$$V_{mig} = \sum_{i=0}^{n} V_{mem} \left(\frac{D}{R} \right)^i \tag{2.4}$$

The *migration latency* T_{mig} is calculated as in Equation (2.5):

$$T_{mig} = \sum_{i=0}^{n} T_i \tag{2.5}$$

3. **Preparation Time:** This metric represents the time difference between the moment of initiation of the migration process and transferring the VM's state to the destination PM, without interrupting the execution and dirtying memory pages.

4. **Resume Time:** This is the time when the VM migration has finished and resumes the VM execution at the destination PM.

5. **Application Degradation:** This is a parameter that indicates the performance of an application that is interrupted or slow down services due to the migration process.

6. **Link speed:** This is also an important parameter that affects the performance of VM migration. Bandwidth allocation or capacity of the link is inversely proportional to downtime and total migration time. Faster transfer during the migration process requires more bandwidth, thus migration requires less time to complete [108].

7. **Page dirty rate:** This is the rate at which VM memory pages are modified by VM applications and this directly affects the number of pages that are transferred in each pre-copy iteration [108]. Higher page dirty rates cause increased data being sent per iteration and this leads in increasing total migration time and service downtime. Dirty page rate and migrating VM performance are in nonlinear relationships. If the rate of dirty pages is lower than link capacity, the migration process can transfer all modified pages at a frequent time then this leads in lower total migration time and downtime. Otherwise, the migration performance degrades.

8. **Energy consumption (E):** This is a key parameter since the target of VM consolidation is to reduce energy consumption. Energy consumption of the data center can be generated from various sources such as CPU, Memory, power supply units, disk storage boxes and cooling systems. Energy consumption is given in Equation (2.6) [66].

$$E = \int Power(u(t))dt \tag{2.6}$$

where, *E* is Energy Consumption, u(t) is the CPU usage, Power is Power Consumption, which is proportional to CPU usage as shown in Equation (2.7).

$$Power(u) = q \cdot P_{\max} + (1-q)P_{\max} \cdot u \tag{2.7}$$

where *q* is the fraction of energy consumed by the idle server, P_{max} is the maximum power consumption by utilized server, and *u* is the CPU utilization.

Energy consumption of the data center for the whole experimental time is measured in KWh.

9. **Service Level Agreement Violations (SLAV):** This is an important metric to measure the quality of service (QoS). Actually, SLA is the agreement between cloud provider and consumer in terms of maximum response time and minimum throughput. SLAV is proposed in [21] and

measure by the SLA violations due to over-utilization (SLAVO) and SLA violation due to migration (SLAVM). Performance degradation due to PM overloading and due to VM migrations metrics is shown in the Equation (2.8) [39], [66].

$$SLAV = SLAVO \cdot SLAVM \tag{2.8}$$

SLAVO indicates the percentage of time, during which active PMs have experienced the CPU or memory utilization of 100% as:

$$SLAVO = \frac{1}{M}\sum_{i=1}^{M}\frac{Ts_i}{Ta_i} \tag{2.9}$$

Where *M* is the number of PMs, Ts_i is the total time that the PM *i* has experienced the utilization level of 100% leading to an SLA violation. Ta_i is the total time period of the PM *i* during active state.

SLAVM shows the overall performance degradation as a result of live migration of VMs, as shown in Equation (2.10).

$$SLAVM = \frac{1}{N}\sum_{j=1}^{N}\frac{Cd_j}{Cr_j} \tag{2.10}$$

Where *N* is the number of VMs; Cd_j is the estimate of the performance degradation of the VMs *j* caused by migrations; Cr_j is the total CPU capacity demanded by the VM *j* during its lifetime.

10. **Number of VM migrations:** Live migration of virtual machines is a costly operation, considering some parameters like amount of CPU processing on a source PM, the network traffic between the source and destination PMs, downtime of the services and total migration time [33]. Therefore, a smaller number of the VM migrations means an efficient consolidation.

    The authors in [21] have defined the *total migration time* and *performance degradation* along VMs live migration, as shown in Equation (2.11) and (2.12), respectively.

$$T_{mj} = \frac{M_j}{B_j} \tag{2.11}$$

where T_{mj} is the total time for VM *j* migration; M_j is amount of memory used by VM *j*, and B_j is the available bandwidth to the VM *j*.

$$U_{dj} = 0.1 \cdot \int_{t_0}^{t_0+T_{mj}} u_j(t)dt \quad (2.12)$$

where U_{dj} is the *total performance degradation* by VM *j*, t_0 is the time when the migration starts, T_{mj} is the time taken to complete the migration, $u_j(t)$ is the CPU utilization by the VM *j*.

11. **Energy consumption and SLA violation (ESV):** A metric combines energy and SLA violations. Actually, if we try to reduce too much energy than the SLA violation will be increased, so need to find a trade-off that will consume less power and still incur a less SLA violation [22]. For this purpose, the ESV metric is defined, which is given by Equation (2.13).

$$ESV = E \cdot SLAV \quad (2.13)$$

where *E* is energy consumption and *SLAV* is the SLA violations of all VMs.

2.7. CloudSim

CloudSim is an extensible simulation framework that enables modelling, simulation and experimentation of cloud computing infrastructures and application services [42]. The main features of CloudSim are [43]:

- Modelling and simulation of large-scale cloud computing data centers.
- Modelling and simulation of virtualized server hosts.
- Energy-aware computational resources.
- Support for data center network topologies and message-passing applications.
- Modelling and simulation of federated clouds.
- Support for user-defined policies for allocation of hosts and host resources to virtual machines.

CloudSim enables researchers and experts from industry to perform tests and experiments, and to develop the best policies, based on specific scenarios and configuration in all the critical aspects related to cloud computing.

Figure 2.18 shows the layered implementation of the CloudSim architecture.

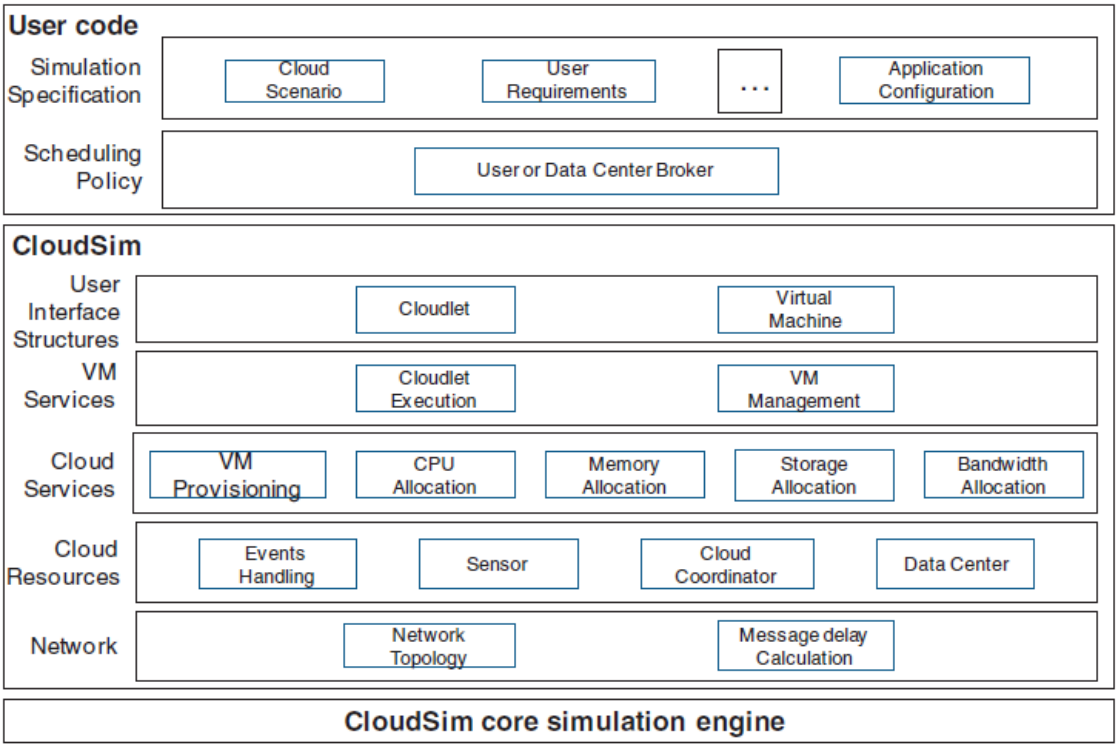


Figure 2.18: Layered CloudSim architecture [42]

The top layer of CloudSim is the User Code, which contains the basic entities such as the number of PMs with all their specifications, the number of tasks and the application's configuration, the specifications of VMs, the number of users and the types of their applications, and scheduling policies and strategies for the data center broker. Based on these modules, an application developer in the cloud can implement various functions, such as workload distribution with increasing demand, perform robust tests based on application configurations, and model reliable and customized techniques.

CloudSim layer enables the modelling of a virtual environment and has special management interface for different resources. This layer implements the basic functions, such as providing the appropriate PMs for VMs, application execution management and monitoring the dynamic state of the system. Therefore, to map the PMs to VMs, the cloud provider can implement the strategies and different techniques by adding the programming modules based on the functions of this layer.

2.8. Summary

In this chapter, we have given an overview of the key concepts and characteristics of cloud computing, starting from fundamental characteristics in cloud computing, cloud service models, deployment models and actors in the cloud. Virtualization has also been described as a fundamental and useful technique that enables efficient resource management in the cloud environment. The agreement between the cloud provider and the consumer, known as SLA, has been addressed, which defines the obligations between the parties and ensures that the requirements regarding the quality of services have been met.

We have provided an overview of dynamic resource allocation in cloud infrastructures with a focus on virtual machine consolidation (VMC) through the live migration mechanism. The components and possibilities of the live migration process were explained. To evaluate the overall performance in the cloud infrastructure are presented the fundamental metrics.

In addition, we have introduced the CloudSim simulator, which is a well-known and very useful simulator in both industry and academia for modelling cloud computing processes.

3

Related Work

3.1. Introduction

In this chapter, we analyze the research work related to resource allocation in cloud infrastructure. Section 3.2 presents and discusses VM consolidation approaches based on live migration process. Approaches that have dealt with dynamic VM consolidation based on hierarchical architectures are presented in Section 3.3. Section 3.5 addresses the problem of task scheduling and resource allocation in cloud environments. Section 3.4 summarizes this chapter.

3.2. VM Consolidation based on Live Migration

An important mechanism to allocate resources to the virtual machines (VMs) in a data center is live migration. Live migration is a costly operation that consumes network bandwidth and energy. The problem of VM consolidation and mapping with physical machines (PMs) need to address the issues: a) when to start the VM live migration process, b) which PM is targeted as a source for VM live migration, c) which VMs need to migrate from selected, which is the destination PM to placement the selected VMs. These issues that pose an optimization problem have been addressed from the different research works.

There are several approaches that have addressed the VM consolidation process through live migration.

Wood *et al*. [44] propose a system called Sandpiper to detect overloading physical machines and creates a new mapping of physical to virtual resources, resizing virtual machines, and initiating migrations. To detect overloaded physical machines, the Sandpiper collects the usage statistics for

VM and PM and based of them create a mirror of resource usage, and then applied the prediction techniques.

Beloglazov and Buyya [21], propose an adaptive heuristic for dynamic consolidation of VMs based on the statistical analyses of the historical data. To detect whether the physical machine is overloaded or underloaded they used lower and upper thresholds. Methods that are used for host overloading are Median Absolute Deviation, Interquartile Range, Local Regression and Robust Local Regression. To select VMs to migrate from overloaded and underloaded host they propose three policies: The Minimum Migration Time Policy, The Random Choice Policy, and The Maximum Correlation Policy. Whereas, for the VM placement is used the Power Aware Best Fit Decreasing (PABFD). This algorithm is a modification of the Best Fit Decreasing (BFD) algorithm. The primary focus of their work was only to reduce energy consumption and do not consider other performance metrics.

Murtazaev and Oh [33] propose an algorithm for server consolidation called Sercon to minimize the number of used servers and number of migrations. Sercon considers a threshold value so that the CPU's physical machine not to reach 100% of the utilization that leads to performance degradation. This algorithm migrates VMs from the least loaded nodes to the most loaded ones. Sercon inherits some characteristics of well-known heuristic algorithms for bin-packing problems, such as First-Fit and Best-Fit. From the results, it can be concluded that the proposed algorithm is scalable for middle-sized data centers.

Feller et al. [34] propose a fully decentralized dynamic VM consolidation scheme based on an unstructured peer-to-peer (P2P) network of physical machines to address the scalability and packing efficiency issues. This scheme uses a dynamic topology that is built by periodically and randomly exchanging neighbourhood information among physical machines. The VM consolidation operates only within the scope of the neighbourhoods where the system can scale with an increasing number of PMs and VMs, without having to require the global system knowledge. In this regard, another contribution is the modelling of a migration-cost aware ACO-based dynamic VM consolidation algorithm, which focuses on minimizing the number of PMs, and the number of migrations required to move from one machine to another.

Khanna *et al.* [46] propose an approach for dynamic consolidation of VMs based on live migration. Their approach for host overload detection is also based on resource usage exceeding a threshold value. Their goal is to minimize the number of hosts by maximizing the variance of resource capacity

residuals. This is achieved by ordering VMs in non-decreasing order of their resource usage and migrating the least loaded VM to the least residual resource capacity host.

Beloglazov *et al.* [19] propose energy-aware heuristic algorithms for dynamic allocation of VMs to hosts based on live migration. They decide on the overload or underload state of a host based on whether the CPU usage is higher or lower than the overload or underload thresholds, respectively. The authors apply a modified Best-Fit- Decreasing (BFD) heuristic to pack VMs to fewer hosts, which considers the power increase of hosts.

Gong and Gu [47] propose a dynamic consolidation approach called Pattern-driven Application Consolidation (PAC) based on extracting patterns of resource usage called signatures using signal processing approaches such as Fast Fourier Transform (FFT) and Dynamic Time Warping (DTW). Based on extracted signatures, they perform dynamic placements of VMs to the hosts that have the highest match between VM resource usage signature and host free capacity signature. Their work focuses on periodic global consolidation for VM resource usage patterns that show periodicity. The authors also consider on demand VM migrations for instantaneous overloads, but in contrast to our approach, they base overload detection on a single resource usage value exceeding a static threshold.

Andreolini *et al.* [48] propose an approach for host overload detection in which a host is declared as overloaded when there is a substantial change in the load trend of the host, as a result of applying the CUSUM algorithm. Their goal is similar to the goal of our work for providing a robust and stable approach avoiding unnecessary live migrations, but their load change point detection requires past history usage data to be available, at which point the SLA violations have already happened. In contrast, our approach applies long-term prediction to avoid violations before they happen.

Esfandiarpoor et al. [110] propose a VM consolidation approach for virtualized data centers in order to reduce energy consumption, by addressing structural features such as racks and network topology of the data center in cloud environments. Their model initially improved the existing VM placement algorithm that is known as Modified Best Fit Decreasing (MBFD) by proposing new algorithms named OBFD, Place VMs Rack by Rack (RBR), Place VMs in Non-Underutilized Rack (NUR), and Hybrid of Server and Rack Consolidation (HSRC). To minimize the energy consumption the intention is to turn off the routers with low traffic or idle routers and cooling equipment. From the experimental results, the proposed algorithms OBFD, RBR, NUR, and HSRC save up to 1.6%, 11.8%, 12.5% and 14.7% energy, respectively, compared to an existing MBFD algorithm.

Yadav *et al.* [112] propose an energy-aware dynamic VM selection algorithm to consolidate VMs from overloaded or underloaded physical machines in order to minimize the energy consumption, to reduce the SLAs violation and to maximize the Quality of Service (QoS). The algorithm named MuMs (Maximum Utilization Minimum Size) for the VM selection scheme, where selects VMs from overloaded or under-loaded PM and migrated to the other PM with sufficient capacity. Therefore, the VM with the highest utilization of the CPU is selected which is divided by the total size of the RAM allocated to this VM. To estimate the efficiency of the algorithm, several metrics are used: total energy consumption, SLA violation, number of migrations, and number of hosts shutdown. The MuMs algorithm is implemented in the CloudSim simulator and is compared with state-of-the-art algorithms such is Median Absolute Deviation (MAD), Linear Regression (LG), and Inter Quartile Range (IQR).

Yadav et al. [109] propose adaptive heuristic algorithms, named Least median square Regression (LmsReg) for overloaded PM detection and Minimum utilization Prediction (MuP) for VM selection of overloaded PMs. The LmsReg algorithm aims to minimize energy consumption and avoid SLA violation. The upper CPU utilization threshold is determined based statistical analyses of the past CPU utilization of the PMs. The variability of CPU utilization directly affects the upper CPU utilization threshold, and if this variability is small then CPU utilization reaches 100% utilization, therefore this leads to SLA violation. To find an optimal solution a robust regression technique is used, which is robust and reliable for dynamic environments. The MuP policy works in such a way that the selection of VM is determined depending on CPU utilization over the period of time. So, select a VM who's CPU utilization is less than the other VMs on same overloaded PM. This policy greatly reduces SLA violation and performance degradation at the migration process. The algorithms are evaluated with real CPU utilization data of heterogeneous PMs for metrics as energy consumption, SLA violation, the number of host shutdown, number of VMs migrations, and performance ratio metric. LmsReg and MuP are compared with other existing algorithms for overloaded PM detection such as Median Absolute Deviation (MAD), Linear Regression (LR), Inter Quartile Range (IQR) and VM selection, such as Minimum Migration Time (MMT), Maximum Correlation (MC) and Minimum Utilization (MU). The results are generated by the CloudSim simulator and show that the proposed algorithms perform better than other existing algorithms that are taken for comparison.

Fard *et al.* [114] developed a VM selection policy to decrease the number of migrations during the live migration process, in order to prevent performance degradation. The proposed VM selection

policy named Maximum Fit (MF) calculates the deviation between the utilization of overloaded PMs and its threshold and through binary search tends to find a VM on the PM in which the utilization is close to the deviation. The implementation of the algorithm is performed on the CloudSim simulator and PlanetLab data. They showed that their VM selection policy performs better than existing VM selection policies Minimum Migration Time (MMT) and Local Regression (LR).

Wang and Tianfield [115] propose two approaches regarding dynamic VM consolidation (DVMC) in order to reduce energy consumption without compromising the SLA. One proposed approach is a VM selection policy known as high CPU utilization-based migration VM selection (HS), and another approach is a VM placement policy named Space Aware Best Fit Decreasing (SABFD). The HS policy sort VMs based on their CPU utilization in decreasing order and the VM with the highest CPU utilization in the overloaded PM will be selected first to migrate. The migration of VMs with the highest CPU utilization will continue until the PM becomes non-overloaded. In addition, the SABFD policy first sorts VMs based on their CPU utilization in decreasing order. The PMs that have enough resources in MIPS (millions of instructions per second) will be estimated for the first VM. The PM with the minimal available resource in MIPS after the VM is placed in will be selected to migrate this VM to. The process repeats until all the VMs have migrated. This VM placement policy for migrating VMs to the destination PMs contributes to decreasing the number of migrations and PM shutdowns, and this leads to energy saving. The results show that through the HS policy the energy consumption is lesser than the well-known VM selection policy MMT (Minimum Migration Time) [21], while the SLA violation metric was higher than MMT. In addition, the results show that SABFD policy performs better than well-known VM placement policy PABFD (Power Aware Best Fit Decreasing) [21] on both energy consumption and SLA violation metrics.

Liu *et al.* [116] developed an approach based on Ant Colony System (ACS) algorithm for allocating the VMs in the minimum number of active PMs in order to reduce energy consumption in data centers. The authors also developed an order exchange and migration (OEM) mechanism for the ACS named OEMACS algorithm, to meet the needs of both homogeneous and heterogeneous physical machine environments. According to this approach, the structure of VMs is constructed by artificial ants based on global search information. OEMACS distributes pheromone between VM pairs that indicates a bond among the VMs on the same PM and record suitable VM groups through learning from historical experience. The OEMACS algorithm is efficient for large-scale problems and by experiments it is seen that this algorithm performs well in minimizing the number of active PMs,

improving the resource utilization, load balancing between resources, and reducing power consumption.

Moghaddam *et al.* [113] propose an energy-aware VM selection policy for CPU load balancing, in order to minimize the number of migration and to reduce the SLA violations. The VM selection policy using the hybrid load balancing model considers the CPU utilization of the VMs on each PMs and linear correlation between the CPU usage of the VMs. The intention is to design an optimal policy that selects the appropriate VMs for migration in order to decrease the time of migration, the number of overloaded and under-loaded PMs and to reduce the energy consumption and SLA violations. To evaluate the performance of the proposed VM selection policy several metrics are used, such is the total energy consumed by PMs, the total SLA violations, the ESV metric that is expressed as production between energy consumption and the SLA metric, and the total number of VM migrations.

There are several works that apply prediction techniques and algorithms for resource allocation in cloud infrastructure.

Bobroff *et al.* [50] proposes a dynamic server consolidation and migration algorithm by combining time series forecasting and bin packing heuristic techniques to minimize the number of physical machines. The proposed algorithm called Measure Forecast Remap (MFR) dynamically remaps VMs to PMs in order to minimize the number of PMs required to support a workload at a specific rate of SLA violations. From the experiments, it is evident that the proposed MFR algorithm achieves a significant reduction in resource consumption, up to 50% compared with static allocation approaches. However, the MFR algorithm does not treat the number of migrations required to a new VM placement.

Prevost *et al*. [68] propose a framework combining the load demand prediction and stochastic state transition models in order to optimize cloud resource allocation by minimizing energy consumed. They used neural network and autoregressive linear prediction algorithms to forecast loads in cloud data center applications. To predict the host utilization, they used statistical and neural network.

Di *et al.* [67] propose a prediction method based on Bayes model to predict a mean load over a long-term time interval. The prediction principle of the model combines the mean load over a future time interval, up to 16 hours and mean load over consecutive future intervals that are referred to as

a pattern. The Bayesian prediction is based on some important properties, which include the expectation, predictability, trends, and pattern of physical machine load. The improvement of the predictive power of a Bayesian model for physical machine load prediction is done by looking at whether the above properties are complementary to each other. They have evaluated their method using Google data centers traces for one month with thousands of machines, where the Bayes method outperforms other techniques that are taken for comparison by 5.6-50 % in long-term prediction.

Gong *et al.* [51] and later Shen *et al.* [52] proposes an approach for VM fine-grained resource allocation based on resource demand prediction. They base their resource demand prediction on two methods: a) Fast Fourier Transform to find periodicity or signature of resource demand and b) a state-based approach using Markov chains. If they predict a conflict, they apply a live migration action to resolve it, considering the migration penalty. As the authors point out, using a multi-step Markov model to predict further into the future lowers the prediction accuracy.

Islam *et al.* [53] proposes resource prediction approaches based on machine learning. More specifically, they propose and experiment with Linear Regression and an Error Correction Neural Network. They show experimentally the superiority of the neural network in making more accurate predictions, but they do not apply their techniques to host overload detection or in general for VM resource allocation.

Khatua *et al.* [54] proposes an approach for VM load prediction several time steps into the future by applying an Auto-regressive Integrated Moving Average (ARIMA) model. They apply their approach for horizontal scaling in cloud settings. If an overload situation is detected, based on some threshold value, then the number of VMs is increased. Also, their approach does not consider the uncertainty and prediction errors in their model of long-term prediction, which is important for increasing the quality of allocation decisions.

Qiu et al. [55] propose an approach for VM load prediction based on a deep learning prediction model. More specifically, this model is composed of two layers, the Deep Belief Network (DBN) and a regression layer. The DBN is used to extract the high-level workload features from the past VM resource utilizations, while the regression layer is used to predict the future load values. The authors evaluate experimentally only the prediction accuracy of the approach, but do not apply it on any VM resource allocation problem.

3.3. VM Consolidation based on Hierarchical Architectures

To increase the physical machine utilization and power efficiency of data centers, a hierarchical architecture is needed.

Some authors have investigated the use of hierarchical architecture for VM consolidation.

Jung et al. [58] have proposed a holistic controller framework called Mistral that optimizes power consumption and overall performance. The authors argue that Mistral can be configured as a multi-level hierarchical controller to allow the management of large-scale systems. This approach has dual objectives; power and performance, and to use both of them the framework uses a utility-based model. They assumed a set of distributed applications to be managed with multiple tiers of components. Each tier may have some replicas that reside inside VMs running on the PMs, with one replica for VM. Each application is associated with a set of transaction types, through which the users access its services. Mistral controllers are activated to determine which VM should reside on PM and how much CPU it should receive. In addition, a system configuration consists of the set of VMs, the PM on which VMs reside, and the CPU fraction allocated to them. According to the experiments, the authors conclude that Mistral provides better overall utility than existing controllers do. It is recommended that Mistral can be used as a multi-level hierarchical controller in large scale systems.

Nurmi et al. [45] have proposed Eucalyptus, an open source cloud computing framework for VM creation and resource control in a hierarchical manner. The Eucalyptus architecture is hierarchical and composes of four high level components.

Feller *et al.* [61] propose a scalable and fault tolerant VM management framework called Snooze. This framework uses a hierarchical architecture that is composed of three software components. A Local Controller (LC), who controls the physical machines. These local controllers are managed by a Group Manager (GM). Finally, at the high tier of the architecture is a Group Leader (GL), who distribute VM requests from the users between the GMs. As well, Snooze supports a power management and VM consolidation aspects.

Farahnakian *et al*. [49] propose a distributed controller to perform dynamic VM consolidation to improve the resource utilizations and to reduce the energy consumption. They used an ant colony system to optimize VM placement. The VM consolidation problem treated as one-dimensional bin packing problem.

Farahnakian et al. [56] have proposed an architecture based on multi-agent systems for dynamic VM consolidation. The authors split the problem of dynamic consolidation into two subproblems, namely host status detection and VM placement optimization. This two-level architecture uses a local agent for each host, which detects when the host is overloaded through a reinforcement learning (RL) approach. Another agent called global agent has a supervisory role. It receives information from the local agent and takes decisions for the migration of VMs.

Farahnakian *et al*. [57] propose a VM management framework based on multi-agent systems aimed to reduce SLA violations and power consumption. The agents, arranged in a three-level hierarchical architecture, are called global, cluster and local agents. A local agent is responsible for the resource usage of the host. To coordinate the local agents by respective clusters, a cluster agent is used, and a master node runs a global agent.

Hwang *et al.* [65] have proposed a hierarchical resource management architecture for VM consolidation in order to improve the energy efficiency. The resource demands are modelled as random variables. Hierarchical resource architecture uses two managers; a global manager assigns VMs into a cluster, while a local manager deploys the VMs to PMs in the cluster.

3.4. Task Scheduling and Resource Allocation in Cloud Environments

Several approaches have been presented to solve the problem of task scheduling in cloud environments. The general task-scheduling problem is NP-complete [76]. Thus, the research in this field focuses on finding low-complexity heuristics that perform well in the scheduling process.
The task scheduling problem is generally divided into two categories: static and dynamic scheduling. In the static scheduling category, all information related to tasks such as execution and communication cost as well as the relationship between tasks is known in advance. In the dynamic scheduling category, such information in relation to tasks is not available and decisions are made at runtime [72].

In general, static algorithms are grouped into two categories: Heuristic algorithms and Guided Random Search algorithms. Heuristic algorithms through polynomial time complexity produce approximate solutions, which are often good solutions. Guided Random Search algorithms also give approximate solutions, but here to improve the solution quality needs more iterations, so it makes it more costly than the heuristic algorithms [86].

The heuristic algorithms are grouped into three subcategories: list-scheduling algorithms,

clustering algorithms and duplication-based algorithms. Clustering algorithms are mostly used for homogeneous systems to form clusters of tasks that are assigned to processors.

The duplication-based algorithms have higher time complexity (i.e., cubic) and the duplication of the execution of tasks and this leads to the higher processor power. Therefore, these algorithms are considered as not very practical.

List-scheduling algorithms provide the most efficient schedules in relation to other categories. In this case, the scheduling algorithm has two phases: the *prioritizing phase,* in which priority is assigned to each task and a *processor selection* phase, in which the suitable processor is selected. If two or more tasks have equal priority, then a task is chosen randomly.

There are many approaches by different researchers, and we will present some of them.

El-Rewini and Lewis [87] propose a Mapping Heuristic (MH) scheduling algorithm that schedules program modules represented as nodes in a task graph with communication onto the target machine topology. MH performs ordering of tasks and then allocates them to the processor. MH handles the contention information, communication delay, the balance between computation and communication in multiprocessor systems. Compared with recently heuristic algorithms the MH algorithm has lower performance because MF only considers a processor ready when then it finishes the last task he has on the ordered list. The time complexity of the MH algorithm is $O(v^2 \cdot p)$, where *v* is the number of tasks and *p* is the number of processors.

Dynamic Level Scheduling (DLS) [88] is another compile-time scheduling heuristic algorithm, which considers inter-processor communication overhead when mapping precedence graphs onto heterogeneous processor architectures. DLS dynamically assigns task priority and match these tasks with processors at each step to eliminate shared resource contention. DLS estimates the availability of each processor if it is ready to perform any task and thus schedules a task to allocate to a currently busy processor. Processor selection is based on the Earliest Start Time (EST) parameter, which does not guarantee the minimum completion time for a specific task and this is a weakness of the algorithm. DLS does not address the idle time between two tasks that are scheduled to be processed on the same processor. The time complexity of the DLS algorithm is $O(v^3 \cdot p)$, where *v* is the number of tasks and *p* is the number of processors.

Iverson *et al.* [89] propose a heuristic algorithm called Levelized Min-Time (LMT) Algorithm. The approach is built in two phases. In the first phase, the problem of mapping and scheduling of the precedence constraints is divided into a series of non-precedence constraints sub-problems. This

process is known as level sorting. In the second phase, are treated individually sub-problems from the first phase. The approach that makes this process is called a Min-Time algorithm. The first and second phase together forms the LMT algorithm. The time complexity of the LMT algorithm is $O(v^2 \cdot p^2)$, where *v* is the number of tasks and *p* is the number of processors.

Another static scheduling heuristic algorithm for heterogeneous processors is called Best Imaginary Level (BIL) [90]. BIL defines a static level of a node incorporating the effect of inter-processor communication overhead and processor heterogeneity. The algorithm has the target to minimize the scheduling length (makespan) of the input task graph. The BIL algorithm offers an optimal solution for linear task graph. The time complexity of the BIL algorithm is $O(v^2 \cdot p \cdot \log p)$, where *v* is the number of tasks and *p* is the number of processors.

Radulescu and van Gemund [91] present two static list-scheduling approaches called Fast Load Balancing (FLB) and Fast Critical Path (FCP). The priority of tasks in these two approaches is assigned dynamically or statically. The weaknesses of the FLB and FCP algorithms are that they make poor scheduling for irregular task graphs and large processor speed variance.

Topcuoglu *et al*. [92] [86] have proposed two low-complexity heuristic algorithms for scheduling DAGs tasks on a bounded number of heterogeneous processors called Heterogeneous-Earliest-Finish-Time (HEFT) and Critical-Path-on-a-Processor (CPOP). The HEFT algorithm has two phases: a *task prioritizing phase,* which defines the priority of all tasks, and a processor selection phase which selects tasks depending on their priority and schedules them on a suitable processor. The CPOP algorithm has two phases: a *task prioritizing and processor selection* phase like the HEFT algorithm, but CPOP algorithm uses a different strategy to set the priority of tasks and to determine the suitable processor for each selected task. The time complexity for the HEFT and CPOP algorithms is $O(v^2 \cdot p)$, where *v* is the number of tasks and *p* is the number of processors. The HEFT algorithm is one the best algorithms in the group of list scheduling heuristic algorithms for task scheduling.

A list scheduling heuristic algorithm for a bounded number of heterogeneous processors is called Heterogeneous Critical Parent Trees (HCPT) [93]. The HCPT algorithm uses a mechanism to construct the scheduling list *L* instead of assigning priorities to the tasks. The algorithm divides the task graph into a set of unlisted-parent trees. The root of each unlisted-parent tree is a critical node (CN). HCPT consists of two phases: listing tasks and machine assignment. To evaluate the HCPT algorithm, a large set of application graphs are used that are generated randomly with varying characteristics based on

real world problems, such as Gaussian elimination, and molecular dynamic code. HCPT algorithm guarantees better scheduling results than FLB, DLS and CPOP, which are explained in this section. The time complexity of the HCPT algorithm is $O(v^2 \cdot p)$, where *v* is the number of tasks and *p* is the number of processors.

Ilavarasan *et al*. [94] propose a task scheduling algorithm for heterogeneous computing systems called High Performance task Scheduling (HPS). The HPS algorithm consists of three phases: *level sorting*, *task prioritization*, and *processor selection*. In the level sorting phase, the given task graph is traversed in a top-down fashion to sort task at each level in order to group the tasks that are independent of each other. So, tasks at the same level can be executed in parallel. In the *task prioritization* phase, for each task a priority is assigned and calculated through the attributes *Down Link Cost* (DLC), *Up Link Cost* (ULC) and *Link Cost* (LC) of the task. The DLC of a task represents the maximum communication cost among all the immediate predecessors of the task. The ULC of a task represents the maximum communication cost among all the immediate successors of the task. The LC of a task is the sum of DLC, ULC and maximum LC of all its immediate predecessor tasks. Based on LC values, at each level, the task with the highest LC value receives the highest priority, followed by the task with the next highest LC value and so on the same level. In the *processor selection* phase, the processor with minimum Earliest Finish Time (EFT) for a task is selected to execute the task [72]. Evaluation of HPS algorithm is performed for parameters such is makespan, speedup, efficiency and the scheduling time. The time complexity of the HPS algorithm is $O(v^2(p \cdot \log v))$, where *v* is the number of tasks and *p* is the number of processors.

Ilavarasan and Thambidurai [85] present another list scheduling for heterogeneous computing systems called low complexity Performance Effective Task Scheduling (PETS). The PETS algorithm as the HPS algorithm [94] explained above has three phases: *level sorting*, *task prioritization*, and *processor selection*. In the *level sorting* phase, the tasks are divided into levels and for each level, the tasks are independent, similar to HPS algorithm. In the *task prioritization* phase for each task is assigned and calculated priority through attributes Average Computation Cost (ACC), Data Transfer Cost (DTC) and the Rank of Predecessor Task (RPT). In the *processor selection* phase, the processor with minimum Earliest Finish Time (EFT) for a task is selected and assigned to execute the task. It also uses an insertion-based policy for a task scheduling in an idle slot between two previously scheduled task on a given processor [72]. The performance of the PETS algorithm has been evaluated based on well-known problems such as LU decomposition, Fast Fourier Transformation, and Molecular

Dynamics code, and is tested for average schedule length ratio, speedup, efficiency and running time metrics. The results have shown that the PETS algorithm performs better than LMT, CPOP and HEFT algorithms. The time complexity of the PETS algorithm is $O(v^2(p \cdot \log v))$, where *v* is the number of tasks and *p* is the number of processors.

Daoud and Kharma [95] present a static list-based scheduling algorithm for heterogeneous distributed computing systems called Longest Dynamic Critical Path (LDCP). LDCP addresses the fact that a single DAG may have more than one critical path, if scheduled on more than one non-identical processor. The LDCP algorithm consists of three phases: *task selection*, *processor selection* and *update status*. In the *task selection* phase, the algorithm first identifies a set of tasks that have a key role in determining the provisional schedule length, then for each processor at the beginning of the scheduling process has constructed a directed acyclic graph that corresponds to a processor. In the *processor selection* phase, using the insertion-based policy, the selected task is assigned to a processor in order to minimize its finish execution time. In the *update status* phase, when a task is scheduled on a given processor, the status of the system must be updated in order to reflect the new changes. During the calculation in the algorithm is neglected the communication cost overhead between the tasks that are scheduled on the same processor. The LDCP algorithm is compared with two other algorithms HEFT and DLS, which are explained in this section, and the results show that LDCP algorithm performs better than HEFT and DLS in terms of normalized schedule length and speedup. The time complexity of the LDCP algorithm is $O(v^3 \cdot p)$, where *v* is the number of tasks and *p* is the number of processors.

Bittencourt *et al*. [96] propose a scheduling algorithm an improvement version of the HEFT algorithm called *Lookahead*. The algorithm takes an improvement for HEFT to provide more information in the scheduling decision-making process before allocating each task. The first approach of the Lookahead algorithm is the use of lookahead information from the task graph in order to minimize the estimated finish time of the children of the task being scheduled. The second approach tackles the priority task list, changing the order of the scheduled tasks in order to see which order gives a better-estimated finish time. The Lookahead algorithm has the same structure as HEFT but computes the estimated finish time metric for each child of the current task. In addition, the Lookahead algorithm satisfies improvements in cases where the communication cost is higher with respect to computation. From simulated experiments of the Lookahead algorithm, it is evident that the algorithm provides shorter makespan up to 20% on average. The time complexity of the

Lookahead algorithm is $O(v^4 \cdot p^3)$ where *v* is the number of tasks and *p* is the number of processors.

Arabnejad and Barbosa [72] propose a list-scheduling algorithm for heterogeneous computing systems called *Predict Earliest Finish Time (PEFT)*. PEFT algorithm compared to state-of-the-art algorithms has a look ahead attribute which does not increase the time complexity. The algorithm is based on an Optimistic Cost Table (OCT) which is used for rank tasks and for processor selection. The OCT is the matrix where the rows indicate the number of tasks and the columns indicates the number of processors. Values from the cost table are used in the processor selection phase. PEFT adds to Earliest Finish Time (EFT) the processing time stored in the cost table for the pair (task, processor). All the processors are tested, and the one that has the minimum value is selected. To set the task priority must be computed the average OCT for each task over all processors. The results show that the PEFT algorithm performed better than state-of-the-art quadratic algorithms in terms of the schedule length ratio, efficiency and frequency of the best results, and offers the lowest quadratic time complexity. The time complexity of the PEFT algorithm is $O(v^2 \cdot p)$, where *v* is the number of tasks and *p* is the number of processors.

Parsa and Entezari-Maleki [77] propose a task-scheduling algorithm called RASA (Resource Aware Scheduling Algorithm) that takes the scalability characteristics of resources into account. RASA is compared with two well-known scheduling algorithms, Max-Min and Min-Min, making use of their advantages and avoiding their disadvantages. Initially, RASA estimates the completion time of the task on each of the available computing resources and then applies Min-Min and Max-Min algorithms. For executing small tasks before large tasks, the RASA uses Min-Min strategy, and to avoid delays in the execution of large tasks and to ensure concurrency in the execution small and large tasks, RASA uses Max-Min strategy. RASA is more efficient in task scheduling and achieves better load balancing.

To achieve better results than RASA, an improved version of the Max-Min algorithm has been proposed by Elzeki *et al.* [78]. Their improved Max-Min algorithm is based on the expected execution time as a basis for selecting tasks instead of completion time. This approach has resulted in better load balancing and smaller makespan than other algorithms used for comparison.

Canon et al. [84] have analyzed 20 static makespan-centric Directed Acyclic Graph (DAG) scheduling heuristics by investigating how robustness and makespan are correlated. The authors have addressed the issue whether dynamically changing the order of the tasks on their processors

can improve robustness. The twenty heuristic algorithms that are analysed, in alphabetical order, are BIL, CPOP, DPS, Duplex, FCP, FLB, GDL, HBMCT, HCPT, HEFT, k-DLA, LMT, MaxMin, MCT, MET, MinMin, MSBC, OLB, PCT, WBA. From the evaluation of the algorithms above are derived some conclusions such as: it is better to respect the static order of the tasks on the processors than to change this order of the tasks dynamically; the robustness and makespan attributes are correlated, and in this case static scheduling tends to be the most robust; algorithms such as HEFT, HBMCT, GDL, PCT, are among the best for makespan and robustness.

To schedule large-scale workflows with various QoS parameters, Chen and Zhang [81] have proposed an Ant Colony Optimization (ACO) algorithm. According to this approach, the users can specify their QoS preferences and determine the minimum QoS thresholds for a given application. The basic parameters of QoS, which are addressed in this model, are reliability, time and cost. The objective of the proposed ACO algorithm is to find a suitable scheduling plan that satisfies all user defined QoS parameters. The model consists of seven instance-based heuristics that guide the search behaviour of ants, and an adaptive scheme to manage these heuristics.

Hu *et al.* [79] have proposed a probability dependent priority algorithm to determine the allocation strategy that requires the smallest number of physical machines to execute tasks. The model considers the processing of interactive jobs only, where jobs usually have small processing requirements and needed good response time performance. The number of physical machines required is affected by the resource allocation and job scheduling strategy within the application environment. In this model, the Service Level Agreement (SLA) is based on response time distribution that is more relevant than the mean response time in terms of performance requirements of interactive applications.

Pandey *et al.* [80] have proposed a scheduling strategy based on a Particle Swarm Optimization (PSO) algorithm to schedule applications to cloud resources that tackle both computation cost and data transmission. They used heuristics to minimize the total cost of execution of application on cloud environments. The algorithm was compared with the existing heuristic algorithm 'Best Resource Selection' (BRS) where PSO can achieve three times cost savings compared to BRS, and the best distribution of the workload to resources. This approach can be used for a variety of tasks and resources by increasing the dimension of the particles and the number of resources.

Byun *et al.* [82] have proposed an architecture for the automatic execution of large-scale

workflow applications on dynamically and elastically provisioned computing resources. A heuristic algorithm named Partitioned Balanced Time Scheduling (PBTS) is proposed that estimates the optimal number of resources to execute a workflow within a user-specified finish time. The algorithm also generates a task to resource mapping and is designed to run online. This approach treats the elasticity of the cloud resources but does not consider the heterogeneity of computing resources by assuming there is only one type of VM available.

Malawski *et al.* [69] have addressed the issue of efficient management under budget and deadline constraints on Infrastructure-as-a-Service (IaaS) clouds. They propose various static and dynamic strategies for both task scheduling and resource provisioning. The three algorithms that are proposed are: a) *Dynamic Provisioning Dynamic Scheduling (DPDS)*, which is an online algorithm that provisions resources and schedules task runtime. DPDS consists of two main phases: a provisioning procedure that based on resource utilization, and a scheduling procedure; b) *Workflow-Aware DPDS (WA-DPDS)*, which is an extended version of the DPDS algorithm by introducing a workflow admission procedure. WA-DPDS compares the current cost and remaining budget, tackles the cost of the currently running VMs, and the cost of workflows that have been admitted; c) *Static Provisioning Static Scheduling (SPSS)*, in contrast to the above algorithms, the SPSS algorithm creates a provisioning and scheduling strategy before running any workflow tasks. From the results, it is evident that an admission procedure based on workflow structure and the task's estimated execution time can improve the quality and performance. Their work considers only a single type of virtual machines (VM) and does not treat heterogeneity of IaaS clouds.

Rodriguez and Buyya [83] have proposed a resource provisioning and scheduling strategy for scientific workflows in cloud infrastructures, specifically in the Infrastructure-as-a-Service (IaaS) model. The authors have modelled this strategy through a static cost-optimization, meta-heuristic optimization technique, Particle Swarm Optimization (PSO), to optimize the total execution cost while meeting deadline constraints. The algorithm considers the fundamental principles of the IaaS cloud such as pay-as-you-go model, heterogeneity, elasticity, dynamic provisioning of computing resources, performance variations, and VM boot time parameters.

Mittal and Katal [98] have proposed a task-scheduling algorithm that builds upon the advantages of the state-of-the-art algorithms considering the distribution and scalability characteristics of cloud resources, named as Optimized Task Scheduling Algorithm (OTSA). The algorithm distributes the tasks over the resources in an appropriate manner to gain the lower value of the makespan metric.

The OTSA is compared to some of the existing algorithms such as Min-Min, Max-Min, RASA, Improved Max-Min, and Enhanced Max-Min, and from the results show that the OTSA algorithm in most cases performs better than other algorithms.

Liu *et al.* [99] have proposed a task-scheduling algorithm based on Genetic and Ant Colony Optimization algorithm in the cloud environments. This approach is a combination of Genetic and ACO algorithms (GA-ACO) in order to get the best result of task scheduling and takes less time. The focus of the GA-ACO algorithm is as follows. At the beginning of the task scheduling process, it takes advantage of a genetic algorithm's global search ability, and forms chromosome by indirect encoding. Then, through the fitness function, the reciprocal of the task completion time is chosen. After the selection, crossover and mutation, the optimal solution is generated and convert this solution into ACO's initial pheromone, and in this way, the optimal solution of the task scheduling is generated. From the simulation results, it is seen that the integration of GA and ACO is useful to solve the task scheduling problems in cloud environments, and efficiently improves the searching of the algorithm.

Cui and Xiaoqing [97] have proposed a workflow task-scheduling algorithm in cloud environments based on genetic algorithm. In this algorithm, the priority of each task is assigned by an up-down levelling method, where all workflow tasks are divided into the different levels that enable the parallel execution of workflow tasks. The task-scheduling problem is addressed in two dimensions. First, a new genetic crossover is designed, and secondly a mutation operation to produce new different offspring for increasing the population diversity. The evaluation of the individual fitness of the population realized through the fitness function synchronously considering the scheduling time and the scheduling cost. The simulation results show that the proposed algorithm offers better performance in reducing the workflow scheduling cost.

Gawali and Shinde [100] proposed a heuristic approach for task scheduling and resource allocation in cloud computing environments. Their approach combines the Modified Analytic Hierarchy Process (MAHP), Bandwidth Aware divisible Scheduling (BATS) + BAR optimization, Longest Expected Processing Time pre-emption (LEPT), and divide-and-conquer methods. Through the MAHP process, each task is processed before its actual allocation to cloud resources. The resources are allocated using the combined BATS plus Bar optimization methods and consider the bandwidth and load of cloud resources as constraints. The proposed model moreover pre-empts resource intensive tasks using LEPT pre-emption. To improve the model more, the divide-and-conquer approach is used, where the improvement it is seen from the experiments by comparing with existing BATS and

Improved Differential Evolution Algorithm (IDEA) frameworks applied to performance metrics as turnaround time and response time. Finally, the proposed approach in terms of resource utilization enables efficient resource allocation with high utility. Maximum utilization is achieved for computing resources such as CPU, memory, and bandwidth, where it differs from other existing approaches that consider only the CPU and memory.

Bryk *et al.* [111] addressed the area of workflow ensemble scheduling algorithms under cost and deadline constraints in IaaS clouds, with a focus on file transfers between workflow tasks which have a large impact on workflow ensemble execution. They developed and implemented a global storage model for transferring files between tasks. The model enables to calculate the bandwidth dynamically and supports a configurable number of replicas, allowing to be tested for various levels of congestion in the system. The paper also addresses the issue of how file transfers affect the execution of scientific applications. It is evident that some applications, for example, the Google Cloud Storage may spend up to 90% of their execution time on file transfers. Also, caching files in local VM storage should be considered where some applications indicate caching ratios greater than 50%.

Tsai *et al.* [137] provided an approach to optimize the problem of task scheduling and resource allocation using a differential evolution algorithm. The proposed algorithm, called improved differential evolution algorithm (IDEA), focuses on the cost and time model in the cloud environment. In the cost model, the costs of processing and receiving subtasks are calculated, whereas for the time model the time for waiting, receiving, and processing are included, excluding the variations of tasks which are not covered.

Maguluri and Srikant [138] proposed a throughput optimal load balancing and scheduling algorithm for a cloud data center with the assumption that the job sizes are unknown in the beginning, however, more information becomes available later. Knowing that each job requests a certain amount of resources, such as CPU, memory, disk space and more, these jobs need to be scheduled non-preemptively on physical machines. However, although the job sizes are unknown, the algorithm does not waste the resources, only in cases when the job sizes have high variability, then the resource wastage is high. The algorithm works when the job sizes are not bounded, when they are geometrically distributed. The proposed algorithm is non-preemptive, so these types of algorithms are more difficult to address because the state of the system for different time intervals is coupled.

Cheng and Wang [139] proposed an energy-saving task scheduling algorithm for cloud environments, which is based on the vacation queuing model. The authors have used the vacation queuing model to schedule tasks in heterogeneous cloud environments, taking into account the change in the state of a compute node, latency during the process of transition of states, and the different energy consumption parameters. The algorithm is based on similar tasks. However, this approach does not promise to ensure proper utilization of resources.

Lin *et al.* [140] proposed a task scheduling algorithm considering the bandwidth resource, based on a nonlinear programming model. Algorithm named as Bandwidth-Aware Task-Scheduling (BATS) is a heuristic algorithm for divisible load scheduling to solve the bounded multi-port model. The model allocates an appropriate number of tasks to each VM including CPU, memory, and network devices. The task scheduling problem based only on CPU and memory resources without the bandwidth resource is not a sustainable solution, because due to the insufficiency of the network bandwidth it can result in waste of resources. Based on experimental results the BATS algorithm performs well in decreasing the execution time and is convenient for scheduling task in bandwidth-bounded cloud environments.

Liu *et al.* [141] proposed a parallel task scheduling algorithm which is an extension of the first-come-first-serve (FCFS) technique, named aggressive-consolidation-based first-come first-serve (ACFCFS) algorithm. The algorithm uses the parallel workload consolidation which includes parallel workloads of different PMs from the set of PMs in order to improve resource utilization. To organize VMs, the method of two-tier processor partition for parallel workload consolidation is used. This two-tier method divides the CPU into two priorities, one with high CPU priority and the other with low CPU priority, and both VM groups are mapped to one processor.

The performance of tasks that run on VMs with high CPU priority is close to the tasks that run on dedicate processors, whereas the idle CPU cycles perform well on tasks that run on VMs with low CPU priority. In the ACFCFS algorithm, users must specify a task's process number and the CPU utilization values. Also, as an extended version of the existing FCFS algorithm, the ACFCFS algorithm retains all the advantages of FCFS, such as no requirements for a task's runtime estimation, no starvation, no task migration, and is easy to implement.

Experimental results show that this algorithm is robust in terms of evaluating CPU usage in parallel processes and CPU cycles.

Keshk *et al.* [142] proposed a task scheduling policy as an improved adaptation of the ant colony technique for the problem of load balancing. The Ant Colony Optimization (ACO) technique uses random optimization search, so it is a reasonable technique for a cloud environment for allocating the incoming tasks to VMs. This algorithm known as a Modified Ant Colony Optimization for Load Balancing (MACOLB) aims to balance the system workload and to minimize the makespan for tasks in the given set. The load balancing factor in MACOLB is a key feature to ensure a lower degree of imbalance in the system, hence it contributes to overall performance gain. However, the proposed approach does not address the availability of resources and the weight of tasks.

Shamsollah *et al.* [143] proposed a model for scheduling physical machine load based on a multi-criteria approach. The model takes into account several criteria with different priorities for allocating processor load fractions. This approach is built on Analytical Hierarchy Process theory (AHP), which is recognized as an adequate method for the scheduling problem, as a problem based on priorities and with variable parameters over time. AHP as a decision-making method and with a multi-criteria attributes consists of three levels: objective level, attributes level, and alternatives level. However, the proposed approach does not provide satisfactory optimization because makespan is defined under priority conditions and is different over time.

Goudarzi *et al.* [144] proposed a resource allocation approach in order to reduce power consumption and migration cost in a cloud infrastructure, assuring that SLA criteria are met at the client-level under probability considerations. A penalty is charged to the provider's system if the client's requirements are not fulfilled, limited to a specific upper ceiling to perform the service in accordance with the terms of the SLA. Addressing the resource allocation problem specifically in VM placement, an algorithm based on convex optimization and dynamic programming is used. Experimental evaluation shows that incorporating the SLA in an effective VM placement phase results in lowering of the operating costs in a cloud environment.

Ghanbari *et al.* [145] addressed the load scheduling problem through a multi-objective optimization of divisible load to increase performance. The method enables to estimate the current computation rates of worker processors.

The proposed approach based on a multi-level tree network-topology explores the effect of the multi-criteria method regarding payment, makespan, and utility, where the results show that the approach offered reduces the makespan, increase the utility, and optimizes the process of scheduling tasks in a cloud environment.

Radojevic and Zagar [146] proposed a model for load balancing in a cloud infrastructure which automates the scheduling of tasks and minimizes human intervention. The model incorporates virtualized resources and the experience of the end users in order to influence the decision for load balancing proactively. The model continuously monitors computing resources, including load balancers and applications in the physical machines, and based on the collected information, the decisions will be directed to the load balancers. However, the model has some drawbacks, such as the lack of an analysis of capabilities of nodes and the configuration parameters. Also, the system does not provide the backup process, which can result in a single point of failure.

Zhu *et al.* [147] addressed the real-time task scheduling problem through the rolling-horizon architecture. In their approach, an energy-aware scheduling algorithm named as EARH is modeled for real-time, aperiodic, and independent tasks, in which the authors have incorporated rolling-horizon strategy. The EARH algorithm has integrated resource scaling up and scaling down strategies which adjust the active PM's scale in order to meet the requirements of the task in real time and save energy. Experimental results show that the proposed approach improves the quality of scheduling for different workloads and aims to save energy in a cloud environment.

3.5. Summary

This chapter has presented and discussed related work on cloud resource allocation. Research works were analyzed in these directions; the state of the art on the VM consolidation through live migration based on centralized architectures; the VM consolidation based on hierarchical architectures; and research work in task scheduling and resource allocation in cloud environments.
By considering these approaches, we have identified the requirements that should be fulfilled in this thesis.

Part II.

Long-term Predictions and Task Scheduling

4

Long-Term Predictions for Host Overload and Underload Detection in Cloud Infrastructures

4.1. Introduction

One of the key mechanisms for dynamic resource allocation is live migration of VMs. Using live migration makes it possible to manage cloud resources efficiently by adapting resource allocation to VM loads, keeping VM performance levels according to SLAs and lowering energy consumption of the infrastructure. However, one problem to address in the context of live migration is to detect when a PM is overloaded or underloaded.

Most of the existing approaches that address the problem of live migration are based on monitoring resource usage, and if the actual or the predicted next value exceeds a specified threshold, then a host is declared as overloaded. A problem with the existing approaches lies in that decisions about when a PM is overloaded or underloaded are made from a single resource usage value or a few future values, so this leads to improper decisions, unnecessary live migration overhead and stability issues. However, it should be noted that live migration is an expensive action which can lead to VM performance violations. A more promising approach is to base live migration decisions on resource usage predictions several steps ahead in the future. This increases stability by performing migration actions only when the load persists for several time intervals, but also allows cloud providers to predict overload states before they happen [38]. However, one should keep in mind that predicting further into the future increases the prediction error and the uncertainty and, in this case, it violates the advantages of long-term predictions.

In this chapter, a new approach for PM overload and underload detection based on long-term resource usage predictions is presented. The following issues are specifically addressed:

- A new approach of dynamic resource allocation of VMs in cloud infrastructure is presented. It combines local and global VM resource allocations. Local resource allocation enables allocating CPU resource shares to VMs according to the current load, while global resource allocation enables live migration actions when a PM is overloaded and underloaded in order to mitigate VM performance violations and to reduce energy consumption.
- Another issue that is presented is based on long-term resource usage predictions to detect when a PM is overloaded or underloaded.
- In relation to the new approach of long-term resource usage predictions the uncertainty and VM live migration overheads are considered.

4.2. Resource Manager Architecture

The resource manager architecture works on the principle of managing an IaaS cloud in which several VMs run on the physical machine. The overall architecture of the resource manager is shown in Figure 4.1, consisting of a *VM Agent*, *Host Agent* and *Global Agent*.

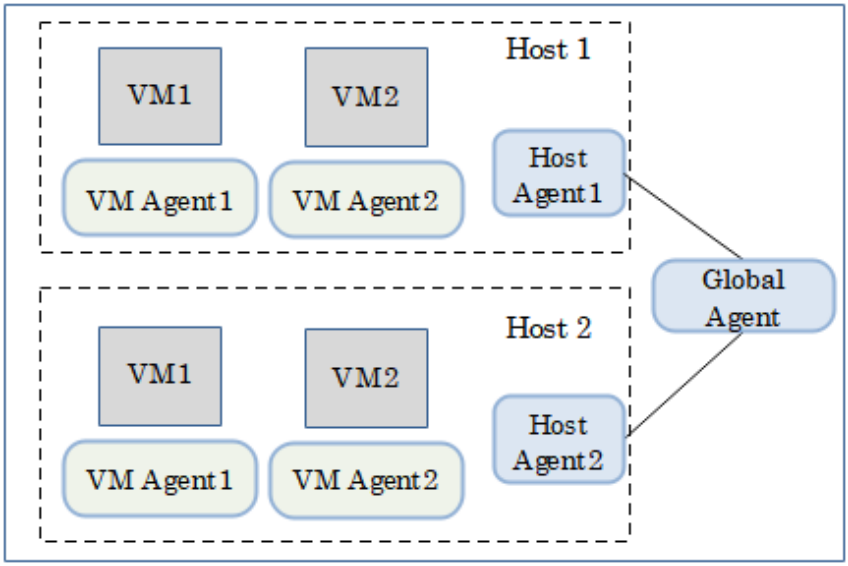


Figure 4.1: Resource manager architecture

The tasks of each agent are described below [38]:

- **VM Agent**: for each VM there is a *VM Agent* which is responsible for local resource allocation decisions by dynamically determining the resource shares to be allocated to its own VM. Allocation decisions are made in discrete time intervals where in each interval the resource share to be given in the next time interval is determined. In our approach, the time interval is

set to 10 seconds in order to adapt quickly to load changes. The time interval is not set to less than 10 seconds, since in long-term prediction this would increase the number of time steps to predict into the future, lowering the prediction accuracy. On the other side, setting an interval larger than 10 seconds can lead to inefficiencies and SLA violations due to the lack of quick adaptation to the load changes. This way of dynamic resource allocation enables the cloud provider to adapt the resources given to each VM according to the current load, thus keeping the required performance level with the minimum resource costs.

- **Host Agent:** for each host (or physical machine) there is a *Host Agent* that receives the resource allocation decisions of all *VM Agents* and determines the final allocations by resolving any possible conflicts. The *Host Agent* decides about the final CPU allocations for all VMs. The possibility of any conflict can arise when the CPU requirements of all VMs exceed the total CPU capacity, but if there is no conflict, then the final CPU allocation is the same as the allocations requested by the VM agents. If there is a conflict, the *Host Agent* computes the final CPU allocations according to the following equation:

$$\text{FinalAlloc} = \frac{\text{ReqAlloc}}{\text{Sum\_ReqAlloc}} \cdot \text{TotalCap} \tag{4.1}$$

  where *FinalAlloc* is the final allocation, *ReqAlloc* is the required allocation, *Sum_ReqAlloc* is the sum of all VMs' requested allocations and *TotalCap* is the total CPU capacity. Another important function of the *Host Agent* is to detect whether the host (PM) is overloaded or underloaded. Then, this information is passed to the Global Agent that then initiates live migration actions for moving VMs away from overloaded or underloaded hosts according to the global allocation algorithm.

- **Global Agent:** has the duty to make global resource allocation decisions to initiate live migration actions of VMs from overloaded or underloaded hosts to the other hosts in order to reduce SLA violations and energy consumption. The *Global Agent* receives information from the Host Agent if a host will be overloaded or underloaded in the future, and based on this information performs the appropriate VM live migration action if it is worth the cost. The *Global Agent* is based on the resource allocation algorithm used in previous work [21] for global VM resource allocation and the Power Aware Best Fit Decreasing (PABFD) [21] algorithm for *VM placement*. In our approach, we have modified these techniques to apply

them in the long-term prediction with uncertainty. In the *VM selection* stage is used the Minimum Migration Time (MMT) [21] policy, but with the modification that only one VM is selected for migration in each decision round even if the host can possibly remain overloaded after migration. This reduces the number of simultaneous VM live migration, and the overhead derived from these actions. For the consolidation process, our approach focuses on the underloaded hosts that are detected by the proposed long-term prediction techniques. From the list of hosts identified as underloaded, the ones that have lower average CPU usage of previous historical values are considered first. Applying the proposed long-term prediction techniques, the hosts that are not underloaded are chosen as the VM live migration destination.

4.2.1 Host Overload Detection

To detect if a host is overloaded, a long-term time series prediction approach is used. In our approach, long-term prediction means predicting 7-time intervals ahead into the future. A host is considered overloaded if the actual and the predicted total CPU usage of 7-time intervals ahead into the future exceed an overload threshold. The predicted total CPU usage of a time interval into the future is estimated by summing up the predicted CPU usage values of all VMs of the corresponding time interval [38].

The value of predicting 7-time intervals into the future is chosen such that it is greater than the estimated average live migration time (around 4-time intervals). The average live migration time is assumed to be known and its value of 4-time intervals is estimated by averaging over all VM live migration times over several simulation experiments. In real world scenarios, this value is not known in advance, but it can be estimated based on the previous history of live migration times. On the other side, having a larger value than 7-time intervals is not useful because some overload states that do not last long can be skipped.

Based on some experiments, increasing the number of prediction time intervals further into the future does not increase the stability and performance.

The overload threshold value is determined dynamically based on the number of VMs and relates to the VM SLA violation metric, as explained in the next sections.

4.2.2 Host Underload Detection

Since in the above section it is the duty of the host agent to detect whether a host is underloaded in order to apply dynamic consolidation by live migrating all its VMs to other hosts and turning off the host to save energy. Also, the long-term time series predictions of CPU usage are used. A host is considered underloaded if the actual and the predicted total CPU usage of 7-time intervals ahead into the future are less than an underload threshold. Hence, the value of 7-time intervals is long enough to skip short-term underload states, but not too long as to miss any opportunity for consolidation [38].

The underload threshold value is a constant value, and it is set to 10% of the CPU capacity, but it can be configured by the administrator according to his or her preferences for consolidation aggressiveness.

4.2.3 Host Not-Overload Detection

To make the decision to initiate the live migration process, the global agent needs to know the hosts that are not overloaded in order to use them as destination hosts for VM live migrations.

A host is declared as not overloaded if the actual and the predicted total CPU usage of 7 time intervals ahead into the future is less than the overload threshold [38]. The actual and the predicted total CPU usage of any time interval is estimated by summing up the actual and predicted CPU usage of all existing VMs plus the actual and the predicted CPU usage of the VM to be migrated. The purpose is to check whether the destination host remains not overloaded after the VM has been migrated.

4.2.4 Uncertainty in Long-Term Predictions

The process of detecting the host if it is overloaded or overloaded based on long- term predictions carries with it the uncertainty of correct predictions, which can lead to erroneous decisions. To take into account the uncertainty of long-term predictions, a probabilistic distribution model of the prediction error is used. The probability density function of the prediction error for every prediction time interval is first computed. Since the probability distribution of the prediction error is not known in advance and different workloads can have different distributions, a non-parametric method to build the density function online is required. Therefore, in our approach, a non-parametric method

for probability density function estimation based on kernel density estimation [123] is used. It estimates the probability density function of the prediction error every time interval based on a history of previous prediction errors [38]. In this case, the probability density function of the absolute value of the prediction error is used. Since there are 7-time interval predictions into the future, 7 different prediction error probability density functions are built online.

4.2.5 Probabilistic Overload Detection

To detect if a host is overloaded, i.e., if the future total CPU usage will be greater than the overload threshold, we use the probability density function of the prediction error for each predicted time interval. This is defined in Algorithm 1 that returns true or false with some probability whether the future CPU usage will be greater than the overload threshold [38].

First, the algorithm finds the probability that the future CPU usage will be greater than the overload threshold. If the predicted CPU usage is greater than the overload threshold, the difference, called max_error, between the predicted CPU usage and overload threshold, is found. For the future CPU usage to be greater than the overload threshold, the absolute value of the error (i.e., the difference between predicted and future value) should be less than *max_error*. Based on a cumulative distribution function of the prediction error, the probability that the prediction error is less than *max_error*, i.e., the future CPU usage is greater than the overload threshold, is found. Since it can happen that the future CPU usage will be greater than the overload threshold, and also that the prediction error will be greater than *max_error*, the probability that this happens, given as *(1-probability)/2*, is added to the calculated probability to yield the final probability *(probability+1)/2*.

Algorithm 1: **Overload Detection**

```
1:  if Pred_Total_Util >= OverThreshold then
2:      max_error=Pred_Total_Util - OverThreshold
3:      probability=CumulativeProbability(max_error)
4:      probability=(probability+1)/2
5:  end
6:  else
7:      max_error=OverThreshold - Pred_Total_Util
8:      probability=CumulativeProbability(max_error)
9:      probability=(probability+1)/2
10:     probability=1-probability
11: end
```

```
12:  probability=(probability)*100
13:  randnum=rand.nextInt(100)
14:  if randnum < probability then
15:      return true
16:  end
17:  else
18:      return false
19   end
```

If the predicted CPU usage is less than the overload threshold, by the same approach, first, the probability that the future CPU usage will be less than the overload threshold is found. Then, the probability that the future CPU usage will be greater than the overload threshold is given as (1-probability). Finally, the algorithm returns true with the estimated probability.

Algorithm 1 returns the overload condition probabilistically only for a single prediction time interval. Therefore, to declare the host as overloaded, the actual CPU usage should exceed the overload threshold, and the algorithm should return true for all 7 prediction time intervals in the future.

The interpretation of taking into account prediction uncertainty in overload detection is as follows. Although CPU prediction can lead to values above the overload threshold, there is some probability, due to the uncertainty of prediction, that the CPU utilization will be lower than the threshold. This means that for some fraction of the time the host will not be considered as overloaded. This increases the stability of the approach, as shown by the lower number of live migrations for the probabilistic overload detection approach, compared to other approaches.

Furthermore, when CPU prediction is lower than the overload threshold, there is some probability that the CPU utilization will be greater than the threshold. This means that for some fraction of the time the host will be considered as overloaded. In summary, we can say that the host is considered as overloaded or not in proportion to the uncertainty of prediction, which is the right thing to do, as supported by our experimental results compared to approaches that do not take prediction uncertainty into account.

4.2.6 Probabilistic Not-Overload Detection

To take into account the uncertainty of long-term predictions in detecting whether a host is not overloaded, Algorithm 2 is proposed. It returns true, with some probability, if the future CPU usage of some prediction time interval will be less than the overload threshold. The host is declared as not

overloaded if the actual CPU usage is less than the overload threshold, and Algorithm 2 returns true for all 7 prediction time intervals in the future.

Algorithm 2: **Not-Overload Detection**

```
1:  if Pred_Total_Util >= OverThreshold then
2:      max_error=Pred_Total_Util - OverThreshold
3:      probability=CumulativeProbability(max_error)
4:      probability=(probability+1)/2
5:      probability=1-probability
6:  end
7:  else
8:      max_error=OverThreshold - Pred_Total_Util
9:      probability=CumulativeProbability(max_error)
10:     probability=(probability+1)/2
11: end
12: probability=(probability)*100
13: randnum=rand.nextInt(100)
14: if randnum < probability then
15:     return true
16: end
17: else
18:     return false
19: end
```

4.2.7 Probabilistic Underload Detection

To detect whether a host is underloaded, Algorithm 3 is proposed. It returns true, with some probability, if the future CPU usage of some prediction time interval will be less than the underload threshold. The host is declared as underloaded if the actual CPU usage is less than the underload threshold, and Algorithm 3 returns true for all 7 prediction time intervals into the future.

Algorithm 3: **Underload Detection**

```
1:  if Pred_Total_Util >= UnderThreshold then
2:      max_error=Pred_Total_Util - UnderThreshold
3:      probability=CumulativeProbability(max_error)
4:      probability=(probability+1)/2
5:      probability=1-probability
6:  end
```

```
7:    else
8:        max_error=UnderThreshold - Pred_Total_Util
9:        probability=CumulativeProbability(max_error)
10:       probability=(probability+1)/2
11:   end
12:   probability=(probability)*100
13:   randnum=rand.nextInt(100)
14:   if randnum < probability then
15:       return true
16:   end
17:   else
18:       return false
19:   end
```

4.3. Experimental Results

An experimental evaluation of the proposed approach is done through the CloudSim [42] simulator. It is a well-known simulator that permits the simulation of dynamic VM resource allocation and energy consumption in virtualized environments. We have made modifications and extensions to the simulator to integrate the proposed approach and to provide support for setting the CPU CAP to VMs for local resource allocation.

In our experiments, a virtualized data center with 100 heterogeneous hosts is simulated. Two types of hosts are simulated, each with 2 CPU cores. One host has CPU cores with 2,100 MIPS and the other one has CPU cores with 2,000 MIPS, while both have 8 GB of RAM. One host simulates the power model of the HpPro-LiantMl110G4 Xeon3040 computer, and the other one simulates the power model of the HpProLiantMl110G5 Xeon3075. On each host are scheduled 3 VMs (in total 300 VMs). Four types of VMs are used, and each VM requires one VCPU. Three VMs require a maximum VCPU capacity of 1000 MIPS, while the other one requires 500 MIPS. Two VMs require 1740 MB of RAM, one requires 870 MB, and the last one requires 613 MB.

To test realistic workloads, the CPU usage data of real VMs running on the PlanetLab [37] infrastructure are chosen to simulate VM workloads. Each VM runs one application (cloudlet in CloudSim terminology) and the cloudlet length, given as the total number of instructions, is set to a large value in order to prohibit cloudlets to finish before the experiment ends. The experiment is run for 116-time intervals, and the duration of a time interval is set to 10 seconds.

We also use WEKA [124], a machine learning framework with Gaussian Processes for regression through its Java API for long-term time series prediction. A history of previous CPU usage data with a length of 20 samples is used for prediction and forecasting model training. To keep the simulation time to acceptable levels, the forecasting model is trained every 5 time intervals with new CPU usage data. For kernel density estimation, the empirical probability distribution implementing the Variable Kernel Method with Gaussian Smoothing of the Apache Commons Math 3.6 API [125] is used. A history of previous prediction errors with a length of 30 samples is used for probability density function model training, which is done in each time interval.

The experimental results are generated by comparing six different approaches, as follows:

a. **No-Migrations (NOM)**: This approach allocates CPU resources locally to VMs but does not perform live migration actions.

b. **Short-Term Detection (SHT-D)**: Represents the detection of whether a host is overloaded, not-overloaded, or underloaded based on short-term CPU usage predictions. Thus, this technique detects an overload state if the actual and the predicted CPU usage values of the next two-time intervals in the future are above the overload threshold. The same applies to not-overload and underload states where the actual and predicted CPU values for the next two-time intervals into the future are used.

c. **Long-Term Detection (LT-D)**: This approach represents overload, underload and not-overload detections on long term CPU usage predictions of the next 7 control intervals into the future.

d. **Long-Term Probabilistic Detection (LT-PD)**: Bases overload, underload and not-overload detections on long term CPU usage predictions of the next 7 control intervals into the future but considers prediction uncertainty through prediction error probability distribution modelling.

e. **Local Regression Detection (LR-D)**: This approach uses the local regression technique to predict the resource usage in the future. We have chosen this state-of-the-art technique since it achieves the best performance as shown by the authors [21] compared to other techniques that use static or adaptive utilization thresholds.

We have defined five performance metrics for evaluation of the proposed approach, as discussed below:

a. **VM SLA Violation (VSV)**: This metric represents the penalty of the cloud provider for violating the performance of the VMs of the cloud consumer. The performance of an application running inside a VM is at an acceptable level if the required VM resource usage is less than the resource share allocated. A VM SLA violation is defined to happen if the difference between the allocated CPU share and CPU usage of a VM is less than 5% of the CPU capacity for 4 consecutive time intervals. The idea is that application performance degraded if the required CPU usage is near to the allocated CPU share. The penalty of a VM SLA violation is the CPU share by which the actual CPU usage exceeds the 5% threshold difference from the allocated CPU, for all 4 consecutive time intervals. In this case it is the goal of the global agent to mitigate VM SLA violations by providing sufficient free CPU capacity through VM live migration, in order to have the CPU share allocation above the required usage by more than 5% for each VM. Through experiments the VM SLA Violation metric if defined to overload states of hosts. It is calculated dynamically based on the number of VMs. Let us define N as the number of VMs on a host. To avoid a VM SLA violation, each VM should have more than 5% capacity above CPU usage, so the total free CPU capacity of the host should be more than $N * 5\%$. Based on this, the overload threshold is calculated as the total CPU capacity (100%) minus $N*5\%$. This means that the overload threshold represents the CPU usage level above which some VMs will have SLA violations.

b. **Energy Consumption (E)**: This metric computes energy consumption in a data center measured in KWh, for the whole experimental time.

c. **Number of VM Migration (NM)**: This metric represents the number of VM migrations for the whole experimental time.

d. **Energy and VM SLA Violations (ESV)**: This metric combines energy consumption (E) and cumulative VM SLA Violation (CVSV), as given in the formula below:

$$ESV = E \cdot CVSV \qquad (4.2)$$

where E is energy consumption and $CVSV$ is the cumulative VSV value of all VMs for the entire experimental time.

The simulation experiment is run for two different load levels called LOW and HIGH and three different VM live migration SLA violation penalties, mp=2%, mp=4% and mp=6% (MP2, MP4, MP6). The load level represents the CPU usage consumed by each VM. The load levels (Low and High) are taken by multiplying the PlanetLab CPU usage values for each time interval with a constant value of 8 and 14, respectively.

The experiment is repeated five times for each combination of approach and load level.

To see the effect the load level has on the VM SLA violation, Figure 4.2 presents the cumulative VSV value for each approach averaged over all combinations of load levels and migration penalties The cumulative VSV value is the sum of VSV values of all VMs for the whole experimental time. The graph shows that the LT-PD technique achieve lower VM SLA violation levels than the other approaches because it considers the prediction uncertainty. It is also evident that the LR-D technique approximately like LT-D perform better than SHT-D approach because both techniques apply prediction of resource usage into the future, but without taking prediction uncertainty into account. Therefore, taking into consideration long-term prediction uncertainty in decision-making is useful for lowering VM SLA violations.

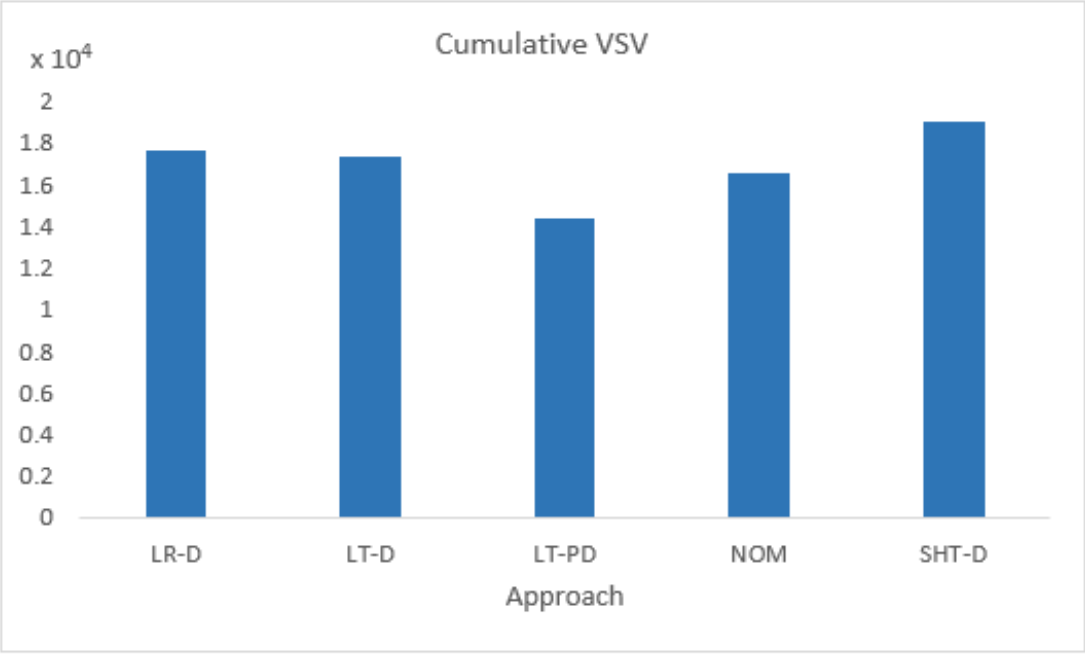


Figure 4.2: Cumulative VSV over all loads and migration penalties

To see the effect the load level has on VM SLA violations, in Figure 4.3 the cumulative VSV value is shown, averaged over all migration penalties, for each approach and the two load levels. From the graph can be observed that in all approaches, increasing the load increases the VM SLA violations, which is expected since there is more contention for resources. It is also seen that for both load levels,

the LT-PD technique achieves the lowest VSV value compared to the other approaches.

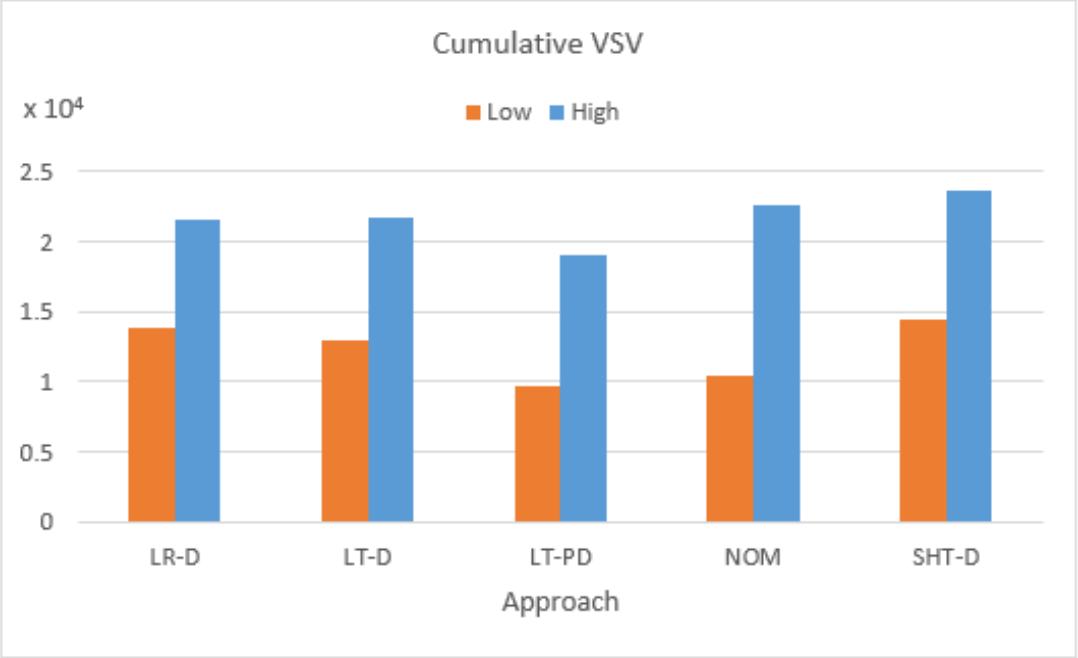


Figure 4.3: Cumulative VSV over all migration penalties and two load levels

In Figure 4.4, the number of VM live migrations for each approach averaged over all combinations of load levels and migration penalties is shown. From the graph it can be observed that the LT-PD approach achieves the smallest number of live migrations compared to other approaches.

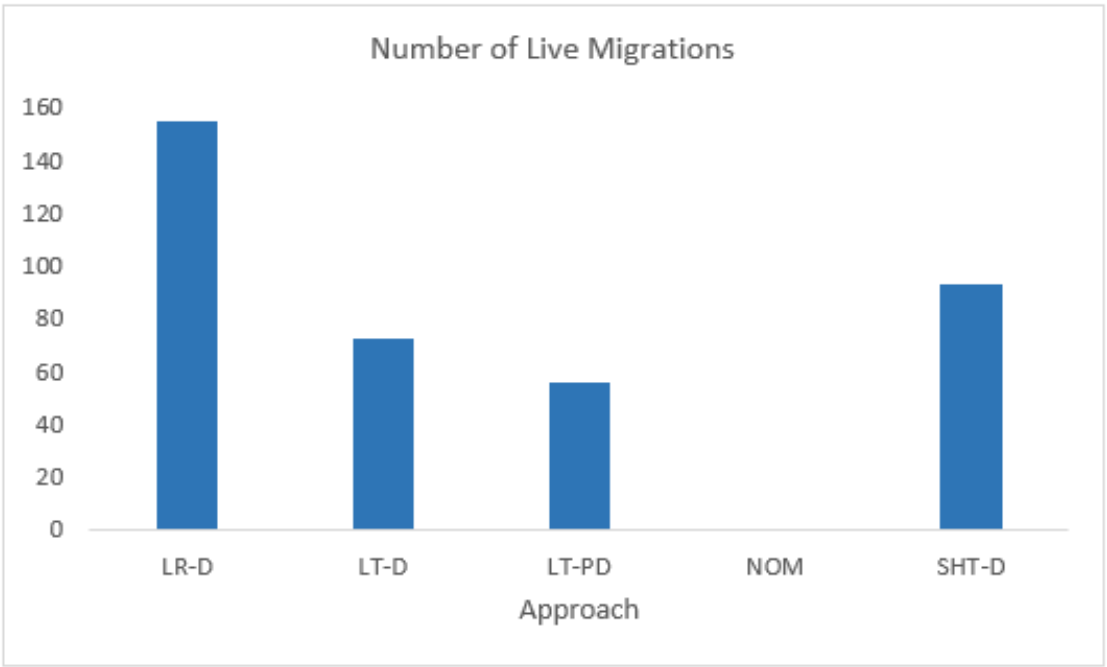


Figure 4.4: Number of live migrations over all loads and migration penalties

The results show that the transition from short-term prediction to long-term prediction increases the stability of the approach thus reducing the number of live migrations. It is also evident that considering uncertainty of long-term predictions and live migration penalties increases stability and reduces the number of live migrations further. Another notable case is the LR-D approach, which has the highest number of VM live migrations compared to other approaches for the fact that the LR-D approach takes live migration actions if only one predicted usage point in the future is above the threshold, while the other approaches check several points into the future.

In Figure 4.5 it is shown for each approach how the number of live migrations is affected by the load level. It can be noticed that the number of live migrations of the LT-PD approach is significantly smaller than for other approaches.

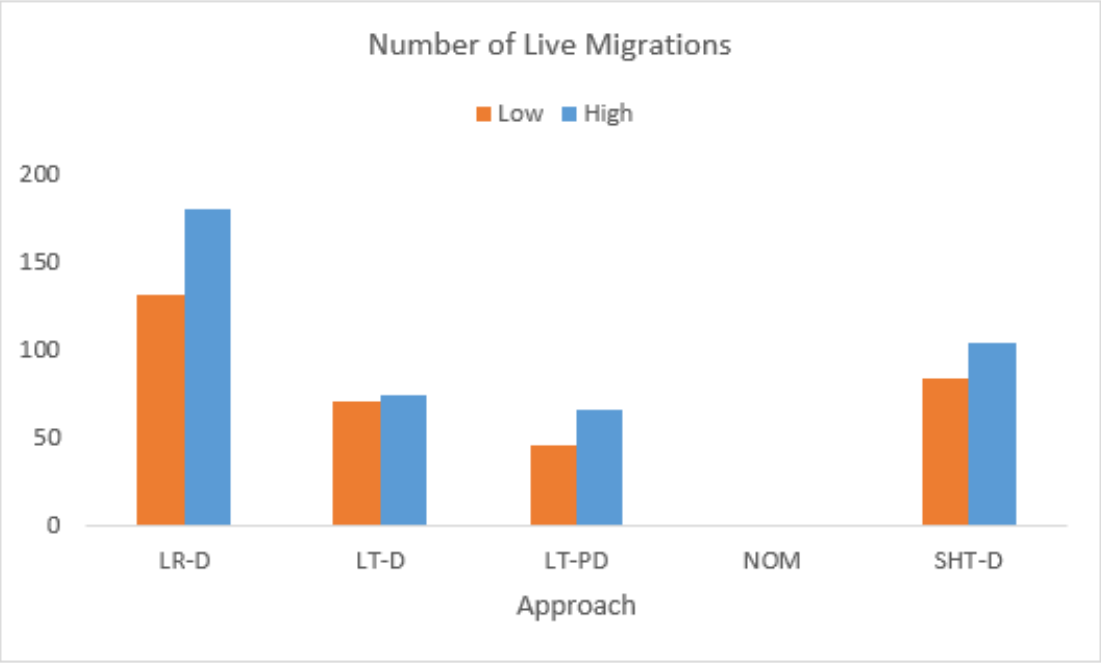


Figure 4.5: Number of live migrations over all migration penalties for two load levels

Figure 4.6 shows the energy consumption of the data center for the whole experimental time for each approach over all combinations of load levels and migration penalties.

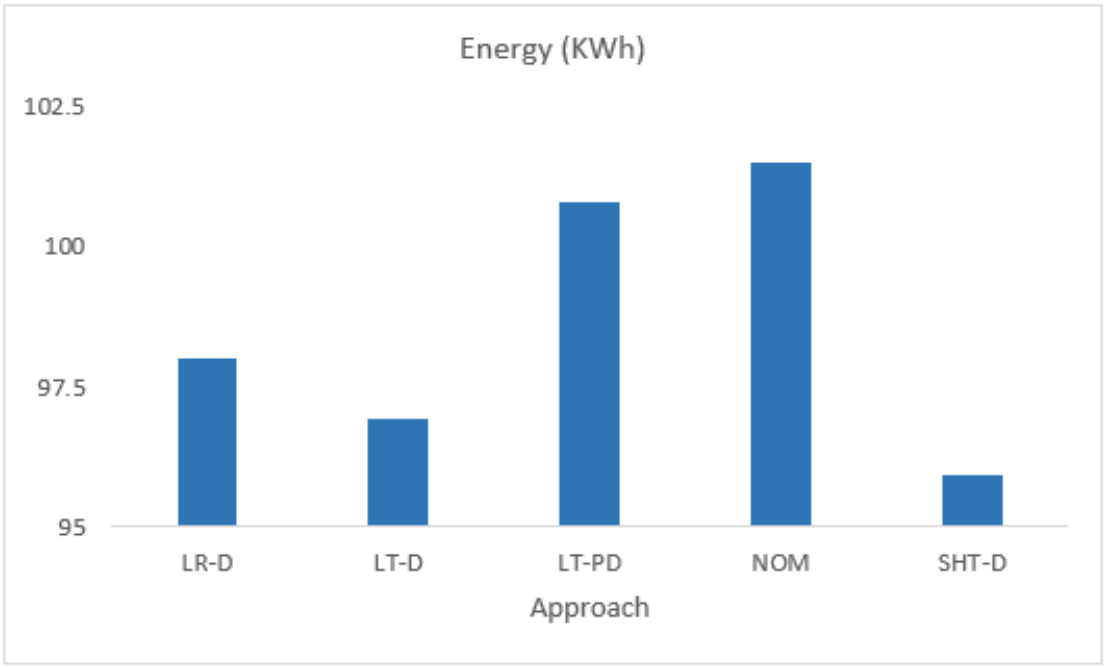


Figure 4.6: Energy over all loads and migration penalties

In Figure 4.7, we show for each approach how the energy consumption is affected by the load level. It can be observed that that increasing the load increases the energy consumption for all approaches. Decreased energy consumption with a decrease in the load level can be explained by the fact that low load creates more opportunities for consolidation and turning off hosts.

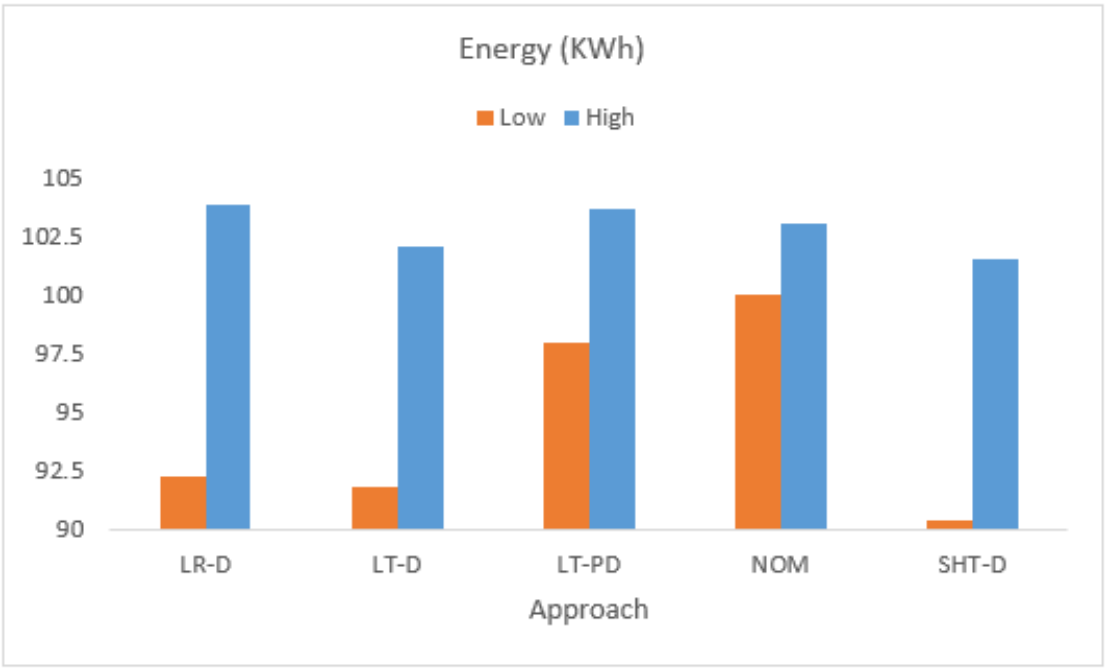


Figure 4.7: Energy over all migration penalties for two load levels

Figure 4.8 shows the ESV value for the whole experimental time for each approach over all

combinations of load levels and migration penalties. The graph shows that LT-PD has lower ESV value than other approaches.

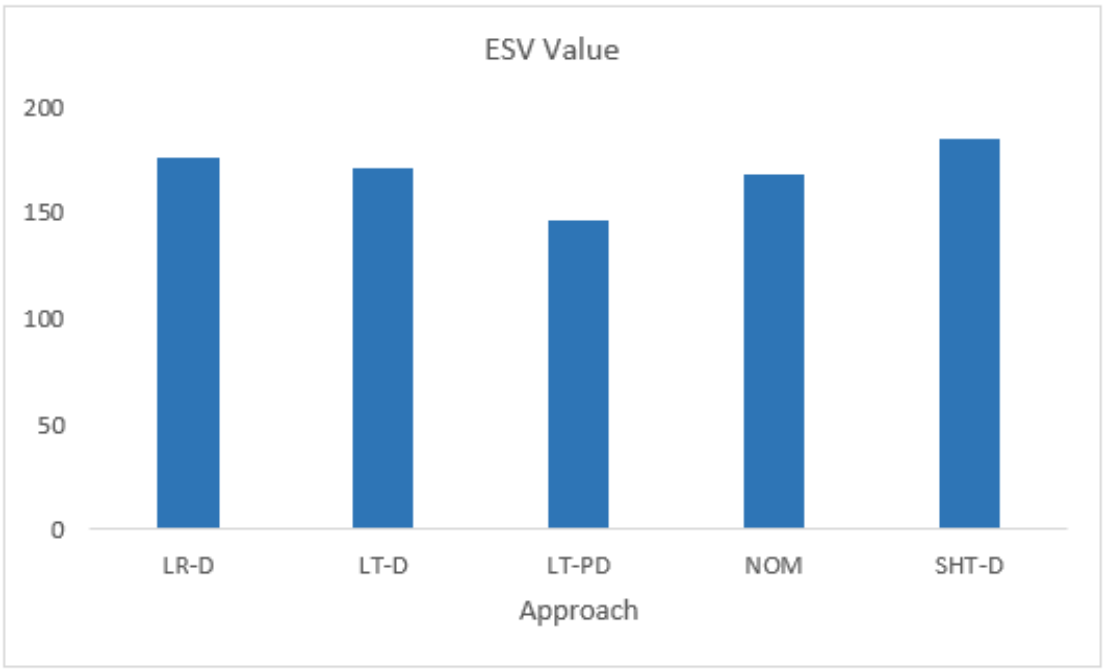


Figure 4.8: ESV value over all loads and migration penalties

In Figure 4.9, we show for each approach how the ESV metric is affected by the load level. It is observed that the ESV metric is lower for the LT-PD approach than the other approaches for each load level.

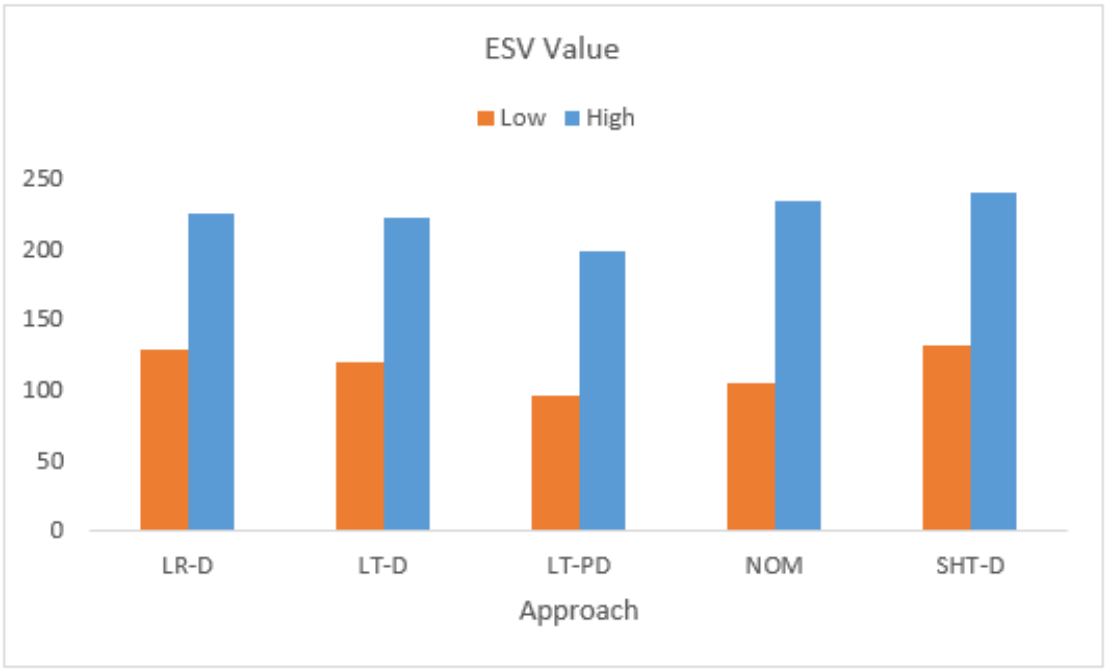


Figure 4.9: ESV value over all loads and migration penalties

4.4. Summary

In this chapter, we have presented a VM resource allocation approach in a cloud infrastructure environment. It allocates resources locally by changing the CPU share given to VMs according to the current load. While global resource allocation is done by migrating VMs from overloaded or underloaded hosts to other hosts to reduce VM SLA violations and energy consumption. Long-term predictions of resource usage are used to detect if a host is overloaded or underloaded, based on Gaussian processes as a machine learning approach for time series forecasting.

We have also considered the prediction uncertainty through a probability distribution model of the prediction error, based on the kernel density estimation method.

Based on the results of the experiments, we can draw conclusions in two directions. First, making long-term predictions of resource demand can increase stability and overall performance of a cloud. Second, making overload detection decisions proportional to the uncertainty of predictions increases the overall performance of the VM migrations in the cloud infrastructure.

5

The Experiential Heterogeneous Earliest Finish Time Algorithm for Task Scheduling in Clouds

5.1. Introduction

Task scheduling in cloud environments is the problem of assigning and executing computational tasks on the available cloud resources. Effective task scheduling approaches reduce the task completion time, increase the efficiency of resource utilization, and improve the quality of service and the overall performance of the system. In this chapter, we present a novel task scheduling algorithm for cloud environments based on the Heterogeneous Earliest Finish Time (HEFT) algorithm, called experiential HEFT. It considers experiences with previous executions of tasks to determine the workload of resources. To realize the experiential HEFT algorithm, we propose a novel way of HEFT rank calculation to specify the minimum average execution time of previous runs of a task on all relevant resources. Experimental results indicate that the proposed experiential HEFT algorithm performs better than HEFT and the popular Critical-Path-on-a-Processor (CPOP) algorithm considered in our comparison.

The Infrastructure-as-a-Service (IaaS) service model in cloud computing can be used to adjust the capacity of cloud resources depending on changing demands of applications. This feature is known as auto-scaling [69].

Task scheduling in cloud infrastructures is the problem of assigning tasks to appropriate resources [70]. Task scheduling can have a significant impact on the performance of the system and is particularly challenging when the cloud resources are heterogeneous in terms of their computation, memory, and communication characteristics, due to different execution speeds, memory capacities, and communication rates between processors.

Typically, the scheduling process in the cloud consists of several phases [75]: resource discovery and filtering, where a broker discovers the resources in the network and collects their status information; resource selection, where the target resources are selected, based on the main parameters of the task and the resources; task submission, where tasks are submitted to selected resources.

Task scheduling algorithms select and allocate suitable resources to tasks such that the overall execution can be completed to satisfy objective functions specified by users or cloud providers [73-74]. For example, to improve Quality of Service (QoS) for users and maximize profit for cloud providers, parameters such as resource utilization, throughput, performance, execution times, computational cost, bandwidth, energy consumption, and Service Level Agreements (SLAs) may be considered [71]. The task-scheduling problem can be classified into static and dynamic scheduling. In static scheduling, all information about tasks such as execution and communication costs for each task and the relationship with other tasks are known in advance. In dynamic scheduling, there is no prior information, i.e., decisions are made at runtime [72].

In Figure 5.1, we present a system model for the workflow scheduling problem in cloud environments. There are three layers: the task graph layer is composed of tasks with precedence constraints, the resource graph layer which represents a network of VMs, and the cloud infrastructure layer as a set of data centers connected by network links [127].

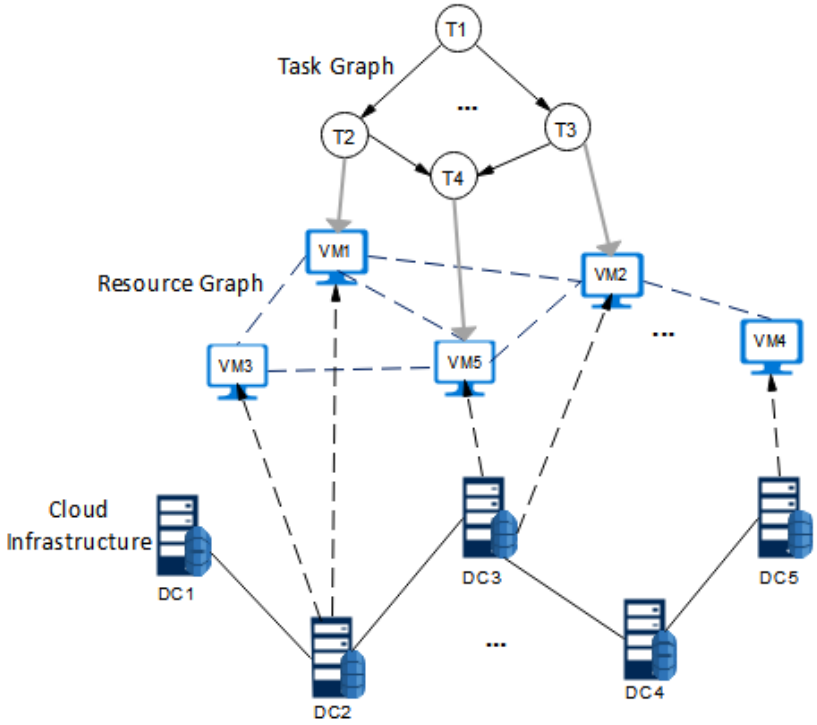


Figure 5.1: A system model of workflow application scheduling in a cloud environment

In this chapter, we present a novel dynamic task scheduling algorithm for cloud environments with heterogeneous resources. It extends the Heterogeneous Earliest Finishing Time (HEFT) algorithm by utilizing past experiences with task executions; hence we call it the *experiential HEFT* (EHEFT) algorithm. It uses an additional parameter that calculates the minimum average execution time of previous runs of a task on all relevant resources. This parameter equips the proposed EHEFT with the ability to take the workload and processing power of resources into account when assigning a task to a processor. It gives priority to a resource that in the past has executed the task faster than others. Experimental results show that our EHEFT algorithm performs better and is more efficient than other than the original HEFT and the popular Critical-Path-on-a-Processor (CPOP) algorithm considered in our comparison.

This chapter is organized as follows. Section 5.2 describes the task scheduling problem formulation. HEFT and CPOP are described in Section 5.3. Our novel EHEFT algorithm is introduced in Section 5.4. Experimental results are presented in Section 5.5. Section 6 concludes the paper and outlines areas for future work.

5.2. Task Scheduling Problem Description

To split an application into tasks with appropriate sizes, we use DAGs. Each task of a DAG corresponds to the sequence of operations and a directed edge represents the dependency between the tasks.

More precisely, a DAG is represented by the graph *G = (V, E)*, where *V* is the set of *v* tasks and *E* is the set of *e* edges between the tasks. Each edge *(i, j)*$\in$*E* represents the dependency such that task n_i should complete its execution before task n_j starts. If a task has no a parent task, this task is defined as the *entry task* of a workflow of tasks. If a task has no a child, this task is defined as the *exit task* of a workflow of tasks.

From the DAG, we derive a matrix *W* that is a *v* x *p* computation cost matrix, where *v* is the number of tasks and *p* is the number of processors; $w_{i,j}$ represents the estimated execution time to complete task v_i on processor p_j. The average execution time of task v_i is defined in Equation (5.1) [85] [72]:

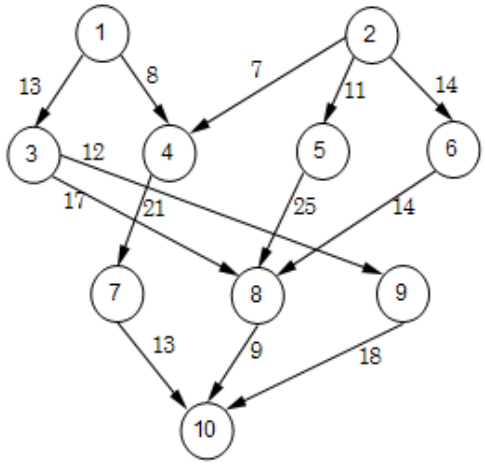
$$\overline{w}_i = \frac{\sum_{j \in P} w_{i,j}}{p} \qquad (5.1)$$

Each edge *(i, j)* $\in E$ is associated with a non-negative weight $c_{i,j}$ which represents the communication cost between the task v_i and v_j. The average communication cost of an edge *(i, j)* is defined by Equation (5.2):

$$\overline{c}_{i,j} = \overline{L} + \frac{data_{i,j}}{\overline{B}} \qquad (5.2)$$

$\overline{L}$ is the average communication startup time and $\overline{B}$ is the average transfer rate among the processors; $data_{i,j}$ is an amount of data required to be transmitted from task v_i to task v_j. In cases when tasks v_i and v_j are scheduled to run on the same processor, the communication cost is considered to be zero, because the intra-processor communication cost is negligible compared to the inter-processor communication cost.

A task workflow example and a computation cost matrix of tasks 1-10 for the resources R1, R2, R3 is shown in Figure 5.2.



Task	R1	R2	R3
1	11	12	17
2	18	21	15
3	8	10	13
4	17	11	19
5	11	5	14
6	21	12	7
7	18	20	12
8	15	8	13
9	11	19	15
10	23	9	14

Figure 5.2: An example of task graph and computation time matrix of the tasks in each processor

A popular metric in task scheduling is the *makespan* or schedule length, which defines the finish time of the last task in the given DAG. The *makespan* is defined by Equation (5.3):

$$makespan = \max\{AFT(n_{exit})\} \qquad (5.3)$$

where $AFT(n_{exit})$ represents the Actual Finish Time of the exit node.

Furthermore, the Earliest Start Time $EST(n_i, p_j)$ of a node n_i on a processor p_j, which is defined in Equation (5.4):

$$EST(n_i, p_j) = \max\{T_{avail}(p_j), \max_{n_m \in pred(n_i)}(AFT(n_m) + c_{m,i})\} \quad (5.4)$$

where T_{avail} is the earliest time at which processor p_j is ready to execute the task. *pred(n_i)* is the set of immediate predecessor tasks of task n_i. The inner *max* block in the EST equation denotes the time at which all data needed by n_i arrive at processor p_j. The communication cost $c_{m,i}$ is zero if the predecessor node n_m is assigned to processor p_j.

Finally, *EFT(n_i, p_j)* defines the *Earliest Finish Time* of a node n_i on a processor p_j, which is defined in Equation (5.5):

$$EFT(n_i, p_j) = EST(n_i, p_j) + w_{i,j} \quad (5.5)$$

5.3. CPOP and HEFT

In this section, we describe two popular algorithms for task scheduling, namely the Critical-Path-on-a-Processor (CPOP) and Heterogeneous-Earliest-Finish-Time (HEFT) algorithms [72] [86].

Canon et al [84] have compared 20 scheduling algorithms and have concluded that both algorithms perform well, but the HEFT algorithm is the algorithm in terms of makespan.

Topcuoglu *et al.* [86] also consider the HEFT algorithm among the best list-based heuristic algorithms, and use the CPOP algorithm, among others, for comparison.

In both algorithms, the tasks are ordered based on a scheduling priority defined by a ranking function. The rank value for an exit task n_i is:

$$rank(n_i) = \overline{w}_i \quad (5.6)$$

For other tasks, the rank values are computed recursively based on the Equations (5.1), (5.2) and (5.6), as defined in Equation (5.7):

$$rank_u(n_i) = \overline{w}_i + \max_{n_j \in succ(n_i)}(\overline{c}_{i,j} + rank_u(n_j)) \quad (5.7)$$

where *succ(n_i)* is the set of immediate successors of task n_i, $\overline{c}_{i,j}$ is the average communication cost of edge *(i, j)*, and $\overline{w}_i$ is the average execution time of task n_i.

5.3.1 CPOP

The CPOP algorithm consists of two phases: task prioritization and processor selection.

The task prioritization phase assigns the priority of each task by computing the rank values for all tasks. In CPOP, for a given application the graph uses a critical path, where the length of this path is the sum of the communication costs of the tasks on the path and the communication costs between the tasks along the path.

The sum of rank values set the priority of each task. Initially, the entry task is the selected task and marked as a critical path task. An immediate successor (of the selected task) that has the highest priority value is selected and is marked as a critical path. This process is repeated until the exit node is reached [86].

In the processor selection phase, the task that has the highest priority is selected for execution. If the selected task is on the critical path, it will be scheduled on the critical path, it will be scheduled on the critical path processor. Otherwise, the task is assigned to a processor that minimizes the earliest execution finish time.

The CPOP algorithm is shown in Algorithm 1 [86].

Algorithm 1 CPOP Algorithm

1: Set the computation costs of tasks and communication costs of edges with mean values.

2: Compute $rank_u$, starting from the exit task.

3: Compute $rank_d$ of tasks, starting from the entry task.

4: Compute $priority(n_i)$ for each task n_i in the graph.

5: $|CP| = priority(n_{entry})$, where n_{entry} is the entry task.

6: $SET_{CP} = \{n_{entry}\}$, where SET_{CP} is the set of tasks on the critical path.

7: $n_k \leftarrow n_{entry}$

8: **while** n_k is not the exit task **do**

9: Select n_j where (($n_j \in succ(n_k)$) and ($priority(n_j) == |CP|$)).

10: $SET_{CP} = SET_{CP} \cup \{n_j\}$.

11: $n_k \leftarrow n_j$

12: **end while**

13: Select the critical path processor.

14: Initialize the priority queue with the entry task.

15: **while** there is an unscheduled task in the priority queue **do**

16: Select the highest priority task n_i from priority queue.

17: **if** $n_i \in SET_{CP}$ **then**

18: Assign the task n_i on critical path processor

19: else

20: Assign the task n_i to the processor p_j which minimized the $EFT(n_i, p_j)$.

21: Update the priority queue with the successors of n_i.

22: **end while**

The CPOP algorithm has $O(v^2 \times p)$ time complexity, where v is the number of tasks and p is the number of processors [72] [85].

5.3.2 HEFT

Similarly, the HEFT algorithm also has the same two phases: task prioritization and a processor selection [72] [86].

In the task prioritization phase, HEFT assigns the priorities of all tasks by computing the rank for each task, which is based on mean computation time and mean communication cost. The task list is ordered by decreasing of their rank values.

The processor selection phase schedules the tasks on the processors that give the Earliest Finish Time (EFT) for the task. The algorithm uses an insertion policy that tries to insert a task at the earliest idle time between two already scheduled tasks on a processor. The slot should have enough capacity to accommodate the task.

The HEFT algorithm also has $O(v^2 \times p)$ time complexity, where v is the number of tasks and p is the number of processors [72] [85] [86].

The HEFT algorithm is shown in Algorithm 2 [86].

Algorithm 2 HEFT Algorithm

1: Set the computation costs of tasks and communication costs of edges with mean values.

2: Compute $rank_u$ for all tasks, starting from the exit task.

3: Sort the tasks in a scheduling list by decreasing order of $rank_u$ values.

4: **while** there are unscheduled tasks in the list **do**

5: Select the first task n_i, from the list for scheduling.

6: **for** each processor *m* **do**

7: Compute *EFT(i, m)* value using insertion-based scheduling policy

8: Assign task n_i to the processor p_j that minimized *EFT* of task n_i.

9: **end while**

5.4. Experiential HEFT

We now present a novel task-scheduling algorithm, called experiential HEFT (EHEFT), which gives the original HEFT algorithm the ability to take the workload and computational power of resources into account when assigning a task to processor. In the EHEFT algorithm, the average execution time of a task is calculated by the definition given in Equation (5.1). Furthermore, the calculation of the average communication cost is performed according to Equation (5.2). As an extension of Equation (5.7), we have added a parameter that calculates the rank by considering the minimum average execution time of the task on each relevant resource. This novel rank calculation is shown in Equation (5.8) [121]:

$$rank_u(n_i) = \overline{w}_i + \max_{n_j \in succ(n_i)} (\overline{c}_{i,j} + rank_u(n_j)) + \min_{j \in R} \frac{\sum_{i=0}^{n_j} w_{i,j}}{n_j} \qquad (5.8)$$

where *R* represents the set of processors; *j* is a processor of the set of processors. The execution time of the task *i* on processor *j* is defined by $w_{i,j}$, while the number of previous executions of the task in processor *j* is defined by n_j.

The proposed EHEFT algorithm is shown in Algorithm 3 [121].

Algorithm 3 Experiential HEFT Algorithm

1: Compute the computation cost for each task according to Equation (5.1)

2: Compute the communication cost of edges according to Equation (5.2)

3: Compute the average execution time of previous runs:

for each task

for each machine **do**

sum up the time of the task's previous executions in the assigned processor.

end for

4: Calculate the minimum as the proportion of the sum from Step 3 and the number of executions of a task in the assigned processor.

5: **end for**

6: Compute the rank value for each task according to Equation (5.8)

7: Sort the tasks in a scheduling list by decreasing order of task *rank* values

8: **while** there are unscheduled tasks in the list

9: select the first task *i* from the list

for each processor *m* **do**

Compute the *EFT(i, m)* value

end for

Assign task *i* to processor *m* that minimized *EFT* of task *i*.

10: **end while**

To prioritize processors that have executed a given task in a more efficient manner in the past, a sum and a count of previous execution times of tasks for each of the resources in the cloud is stored. When there is no such data, the EHEFT algorithm performs exactly as the HEFT algorithm itself. Therefore, EHEFT algorithm we propose is highly dependent on the values of past execution times of tasks.

Assuming a high heterogeneity between cloud resources, variable processing powers, and workloads, as well as considering that some tasks are better suited for a particular processor architecture than others, by including the minimum average execution time of previous runs of a task in the resources of the cloud, the EHEFT algorithm gives precedence to a processor that has performed better in executing a given task in the past.

5.5. Experimental Results

In this section, we present experimental results of our proposed EHEFT algorithm compared to the existing HEFT and CPOP algorithms. The tests were conducted on an Intel Core i7-6500U CPU with a 2.50 GHz × 4 speed, 16 GB of RAM, on Ubuntu 16.04 LTS.

To evaluate the performance of the EHEFT algorithm the application graphs that are generated randomly, are considered. We have implemented and simulated three algorithms using the Python programming language. Our simulator has five input parameters: the number of resources (i.e., processors) in the cloud, the number of DAG nodes (i.e., tasks), connections between tasks, resource heterogeneity, and previous run statistics for each task

The input parameters that are defined to build the weighted DAG are:

- Number of computation nodes in the DAG (Number of DAG Tasks).
- β (Range percentage of computation costs on processors). It is the heterogeneity factor for the processor speeds. A high β value causes higher heterogeneity and different computation costs among processors, and a low β value indicates that the computation costs for a given task are nearly equal among processors. The average computation cost of each task n_i in a given graph $\overline{w_i}$ is selected randomly from a uniform distribution with range $\left[0,2 \cdot \overline{w_{DAG}}\right]$, where $\overline{w_{DAG}}$ is the average computation cost of a given graph that is set randomly in the algorithm. The computation cost of each task n_i on each processor p_j is randomly set from the following range.

$$\overline{w_i} \cdot \left(1 - \frac{\beta}{2}\right) \leq w_{i,j} \leq \overline{w_i} \cdot \left(1 + \frac{\beta}{2}\right) \tag{5.9}$$

- CCR (Communication to Computation Ratio). This metric defines the ratio of the sum of the edge weights to the sum of the node weights for a given DAG. If a CCR value for a given DAG is very low, then it can be considered as a computation intensive application.

For simplicity, constant values are set for the average computational and communication costs. The simulator defines a set of virtual resources with heterogeneous processing powers, as well as current computational workloads and communication costs for the given input DAG. The simulator is fed with

different input values to test variance of the algorithm in terms of makespan and runtime under different conditions.

In our experiment, the following parameters with defined values are used:

- *β* = {0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
- *Number of DAG Tasks* = {5, 10, 15, 20, 25}
- *Connectivity* = {0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0}
- *Number of Processors* = {2, 3, 4, 5, 6}

Our experiments are run for each of the tasks in each of the processors. Such test runs were performed to collect statistics for the execution times that were then used to calculate the minimum average execution time for past runs of a given task on all cloud resources.

To avoid that this additional parameter biases the ranking function, a scaling parameter is used. This scaling parameter determines the weight of the minimum average execution time for past runs in the overall calculation of rank. For all compared algorithms, the simulation conditions were the same.

We use the following performance metrics for our evaluation of the proposed approach.

5.5.1 Scheduling Length Ratio (SLR)

To evaluate a schedule for a single DAG, the most commonly used metric is the makespan. The makespan represents the finish time of the last task in the scheduled DAG, as shown in Equation (5.3).

Considering that a large set of task graphs that have different properties is used, then the schedule length should be normalized to a lower bound, which is known as the Schedule Length Ratio (SLR), defined in Equation (5.10).

$$SLR = \frac{makespan}{\sum_{n_i \in CP_{MIN}} \min_{p_j \in Q} \{w_{i,j}\}} \qquad (5.10)$$

The denominator in SLR metric is the minimum computation cost of the critical path tasks, represented as CP_{MIN}.

Figures 5.3 - 5.6 show the makespan of the three algorithms calculated by the example task graph and computation time matrix of the tasks in each processor of Figure 5.2. The calculation of the makespan is performed for (a) Number of Tasks (Figure 5.3), (b) Connectivity (Figure 5.4), (c) Number of Processors (Figure 5.5), and (d) Processor Range (Figure 5.6).

In Figure 5.3, the simulations are run for five different DAG nodes, with an increasing number of nodes. As expected, the makespan increases with the number of nodes for each of the algorithms we evaluated. EHEFT performs better than the other algorithms because it considers the heterogeneity of resources when calculating the rank for a task. It assigns the execution of a task to a resource that not only has the best present conditions to achieve the earliest finish time, but that has also shown to do so in the past.

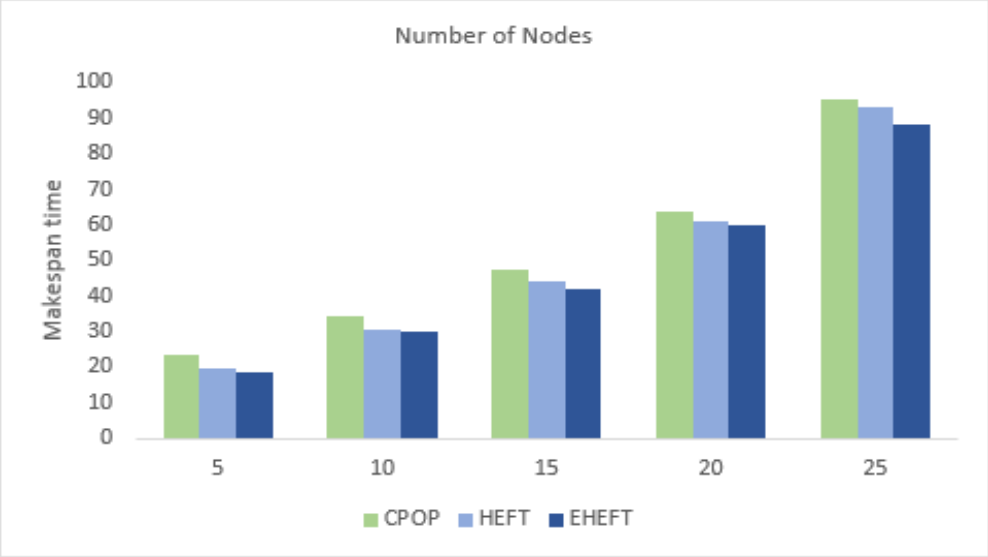


Figure 5.3: Makespan for number of tasks

Figure 5.4 shows that increasing the connectivity between nodes of the input DAG also increases the makespan of the algorithms linearly. The higher the number of dependent tasks on the graph, the more time it takes for the algorithm to assign and execute the tasks. Therefore, it is important to assign tasks that are part of critical paths to resources that can execute them in the fastest manner. In our simulations, we have put more load on the tasks in the critical path. Thus, the results indicate the ability of EHEFT to assign such tasks to resources with highest processing power.

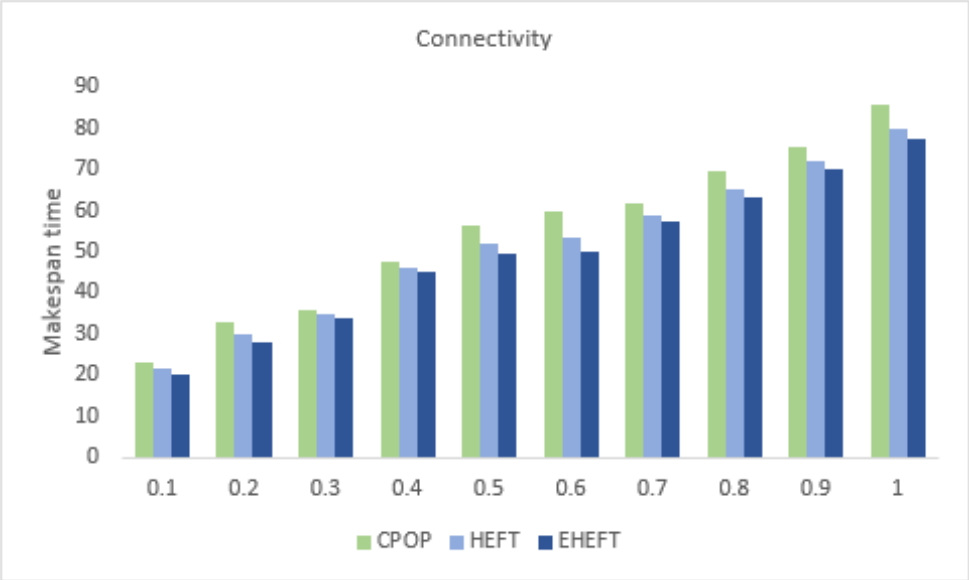


Figure 5.4: Makespan for connectivity

Figure 5.5 shows that increasing the number of processors and a constant node number for the input DAG decreases the makespan. The improvement in performance of the EHEFT algorithm is due to the variance in the calculation of the rank that the statistics of previous runs provide.

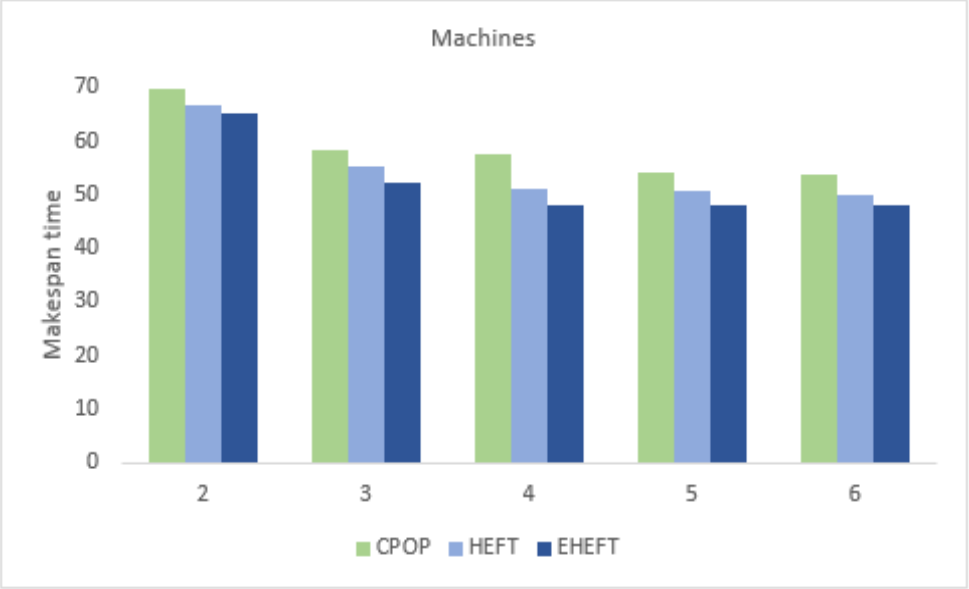


Figure 5.5: Makespan for number of processors

The main advantage of EHEFT over HEFT and CPOP is the processor range parameter *Beta*, as shown in Figure 5.6. EHEFT considers the processing efficiency of a resource for a task, given its previous run statistics. The performance of EHEFT improves with the increase of the processor range, because it is

the factor that makes the highest difference with respect to the average past execution time parameter.

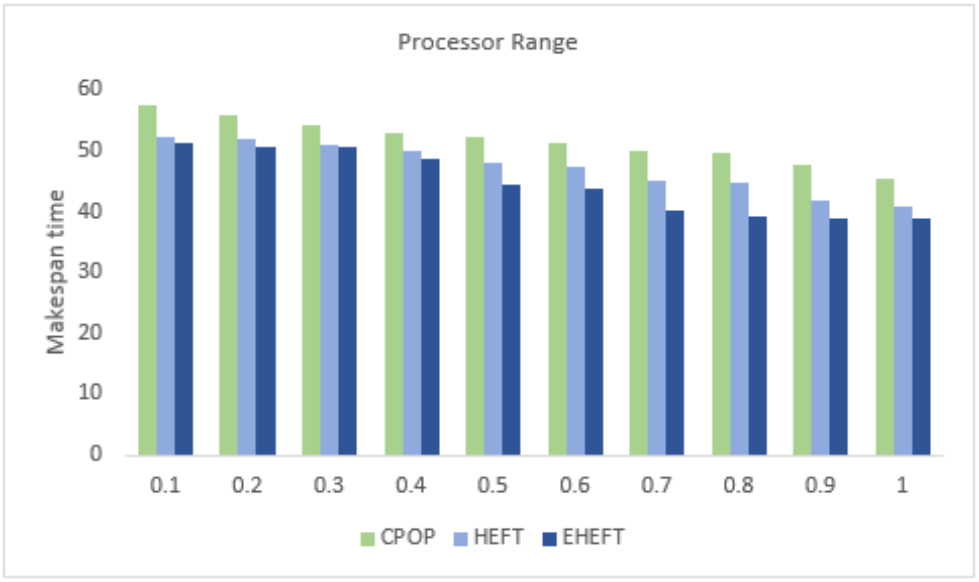


Figure 5.6: Makespan for processor range

In Figure 5.7, we present the SLR value for each of algorithms calculated from the number of nodes. The graph shows that the SLR value for EHEFT is lower than for the HEFT and CPOP algorithms.

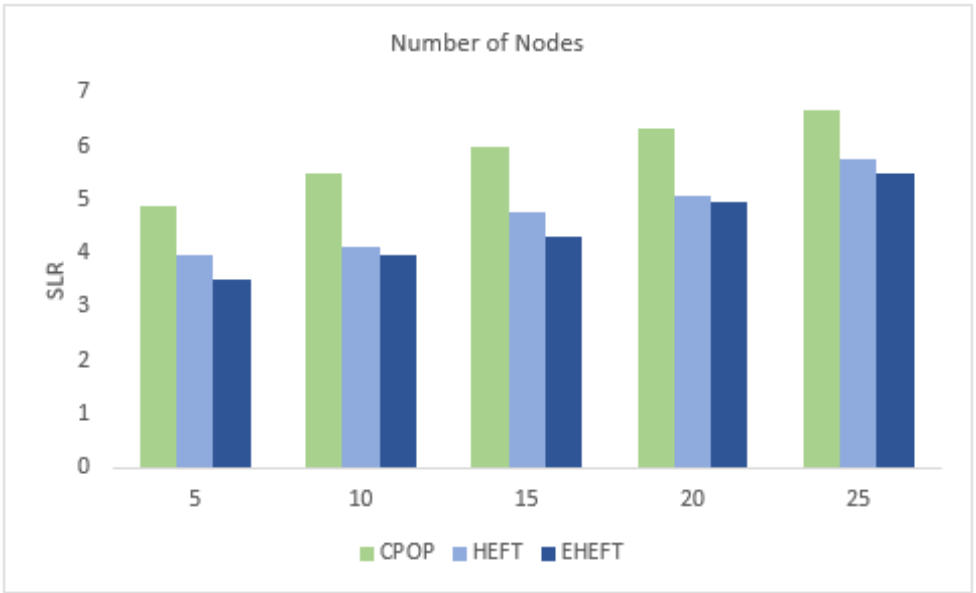


Figure 5.7: Scheduling length ratio

5.5.2 Runtime

The runtime metric represents the execution time to obtain the output of schedule for a given task graph. The results for the runtimes of our performed experiments are shown in the graphs below.

Figures 5.8 – 5.11 show the runtimes of each of the algorithms calculated for the parameters (a) Number of Tasks (Figure 5.8), (b) Connectivity (Figure 5.9), (c) Number of Processors (Figure 5.10), and (d) Processor Range (Figure 5.11).

Figure 5.8 shows that for small numbers of DAG nodes the difference in runtime between EHEFT and HEFT is small. The improvement in performance that the minimum average execution time of the task for each resource gives is only apparent when the number of nodes in the input DAG increases.

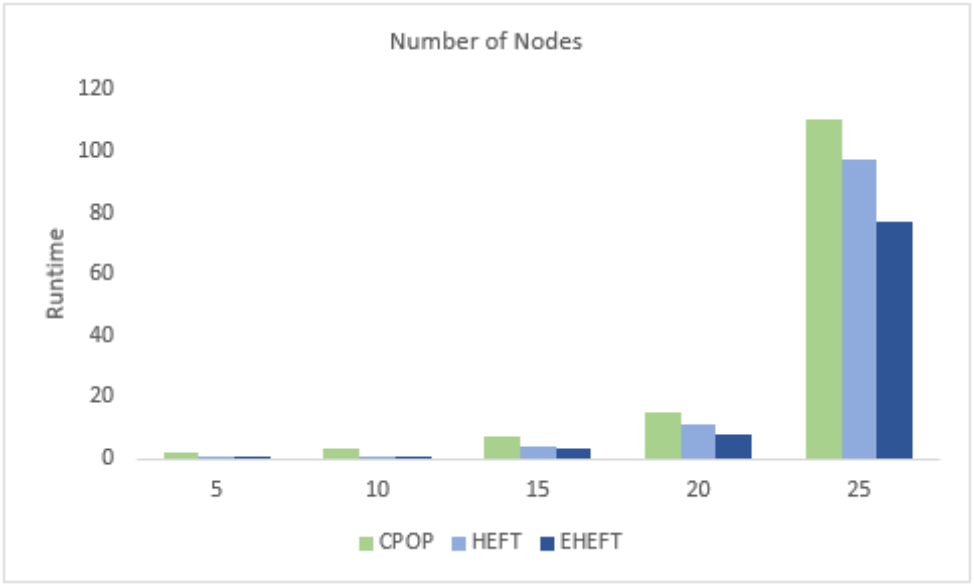


Figure 5.8: Runtime for number of tasks

Figure 5.9 shows that increasing the connectivity between the nodes in the graph results in an increased runtime for the algorithms. The small difference in performance between EHEFT and HEFT is due to the difference in the handling of critical path tasks in the overall runtime. The CPOP algorithm takes more time to execute due to its two-phase rank calculation.

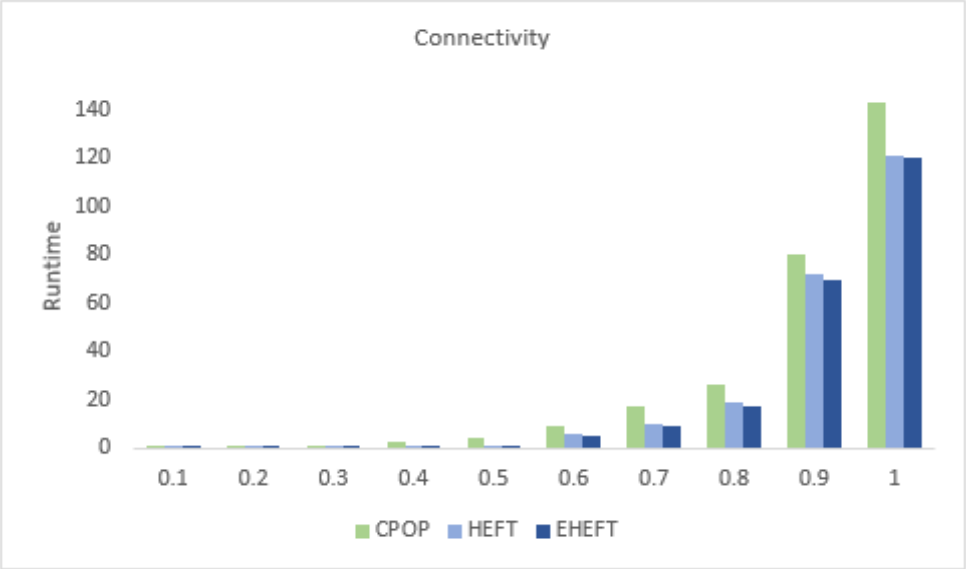


Figure 5.9: Runtime for connectivity

As shown in Figure 5.10, task assignment and execution are performed in a faster manner with an increasing number of simulated virtual resources, as indicated by the decrease in the runtimes of the algorithms. The EHEFT algorithm gains an edge in performance due to its ability to assign the heavy loaded tasks and the ones in the critical path to better performing resources.

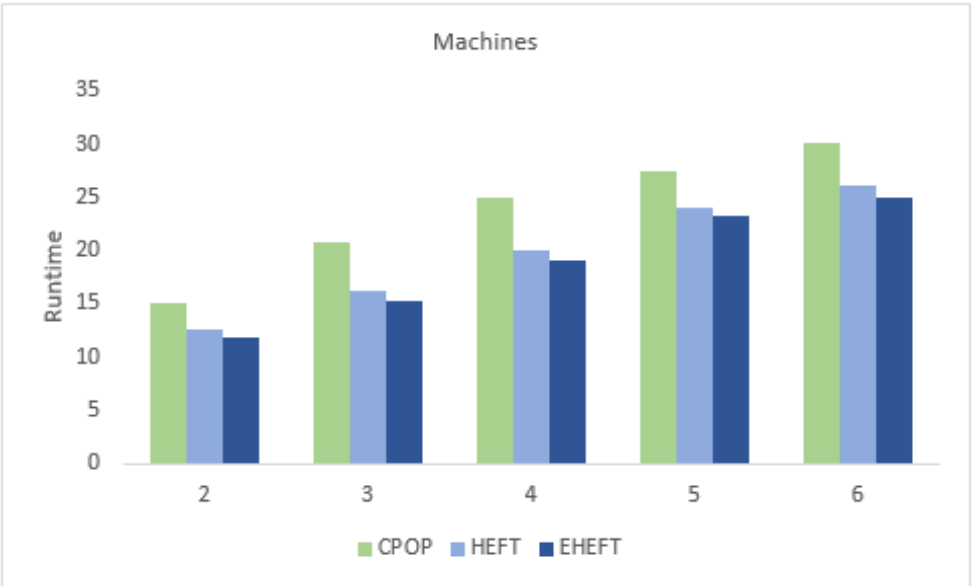


Figure 5.10: Runtime for number of processors

Figure 5.11 shows that increasing the processor range improves the performance of all the algorithms that we evaluated. Since the CPOP algorithm assigns only tasks on the critical path to critical path processors, the EHEFT and HEFT algorithms show better results due to the fact that not

only critical path tasks may be assigned to high performance resources. Tasks that take more time to process and that are not on the critical path of the graph are better assigned to resources through the HEFT and EHEFT algorithms. The difference in performance between the EHEFT and HEFT algorithm lies in the fact that EHEFT assigns tasks to resources that were better suited to execute such tasks in the past.

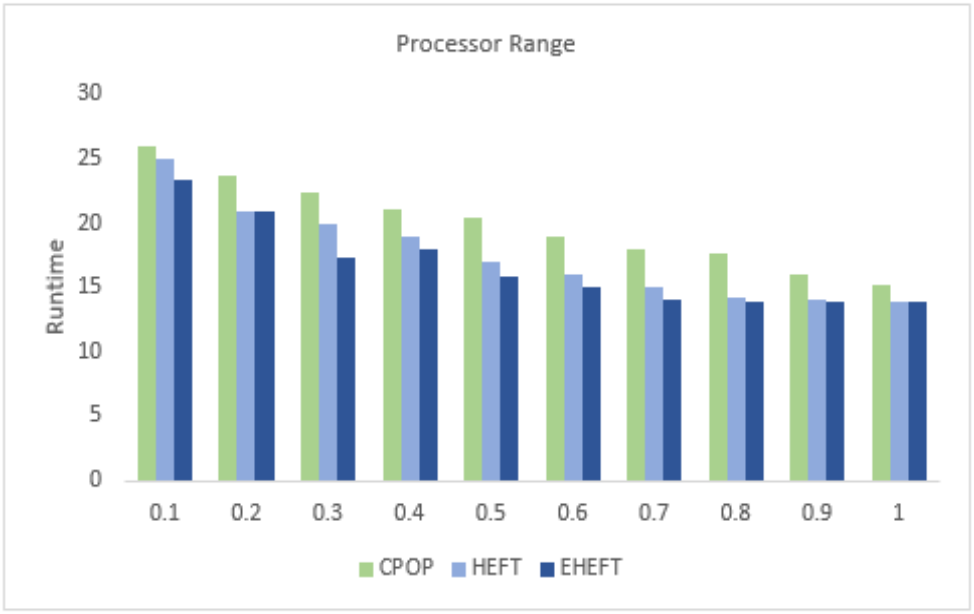


Figure **5**.11: Runtime for processor range

5.6. Summary

In this chapter, we have presented a novel task-scheduling algorithm for cloud environments, called Experiential Heterogeneous Earliest Finish Time (EHEFT) algorithm. In EHEFT, we have modified the rank calculation of the original HEFT algorithm by adding a parameter that specifies the minimum average execution time of a task on each relevant resource. The EHEFT algorithm performs better than the original HEFT and CPOP algorithms in terms of scheduling length ratio and runtime.

Part III.

Distributed Resource Allocation

6

Distributed Resource Allocation in Cloud Computing using Multi-Agent Systems

6.1. Introduction

The virtualized infrastructure is a key component that enables a data center to serve multiple users in an efficient, flexible and secure way. The cloud infrastructure must accommodate varying demands within certain time constraints; hence, it requires powerful dynamic resource allocation methods.

The Infrastructure-as-a-Service (IaaS) model of cloud computing allocates resources in the form of VMs that can be resized and live migrated at runtime [33].

The rapidly growing demand from hundreds of millions of end users for the use of Internet-scale applications has caused cloud providers (such as Google, Amazon, and Microsoft) to operate large-scale data centers around the world. These large-scale data centers consume a large amount of energy. A large energy consumption leads to high costs and to high carbon emissions. Currently, data centers that power Internet-scale applications consume about 1.3% of the worldwide electricity supply, and this fraction is expected to grow to 8% by 2020 [31].

In recent years, a primary focus of research in the field of cloud infrastructures is to reduce energy consumption and service level agreement (SLA) violations for efficiently managing resources.

Most of existing state-of-art VM resource allocation approaches are centralized, but a centralized controller does not scale well for large cloud infrastructures, might represent a communication bottleneck, and is a single point of failure in terms of reliability [57].

As well, some of the existing approaches for VM consolidation [32-35] have as their main objective the energy efficiency and the reduction of SLA violations. For VM consolidation, these approaches

use live migration actions. Some authors have treated the VM consolidation process as an optimization problem [36], taking into account constraints such as data center capacity and SLAs.

In contrast to the existing dynamic VM consolidation approaches presented above, in this chapter we propose a distributed resource allocation approach based on multi-agent systems. Our approach does not use static or adaptive thresholds, but it is based on the utility function model based on host CPU utilization. Basing resource allocation decision on utility function optimization offers a flexible resource allocation policy that is not present in the threshold- and rule-based policies. This means that the core allocation algorithm, namely the utility function optimization mechanism, does not need to change. Changing the form of the utility function can easily change the resource allocation policy.

6.2. System Architecture

In this section, we discuss large-scale data center architectures consisting of *m* hosts and *n* virtual machines running on each host. Since the workload and the CPU usage change over time, an efficient approach for dynamic VM consolidation is needed.

Our work [40] focuses on two models of architectures that will be presented below. Figure 6.1 depicts a two-level centralized architecture, consisting of a local agent called Host Agent and a central controller called Global Agent. The tasks of each agent are described below:

- **Host Agent (HA)**: is responsible for continuously monitoring the host's CPU utilization and to determine whether the host is in an overloaded or underloaded state. This information is passed to the global agent that initiates live migration actions if needed. It is also responsible for initiating local allocation actions by deciding about the CPU capacity (CAP) allocation to each VM and resolving conflicts when the sum of the CAP values for all VMs is greater than the total CPU capacity.

- **Global Agent (GA)**: makes global resource allocation decisions to optimize the VM placement in order to reduce the SLA violations and energy consumption. It gets the information from the HA for a host's status data, available CPU capacity, used CPU capacity, and the predicted CPU utilization, and performs the appropriate live migration actions.

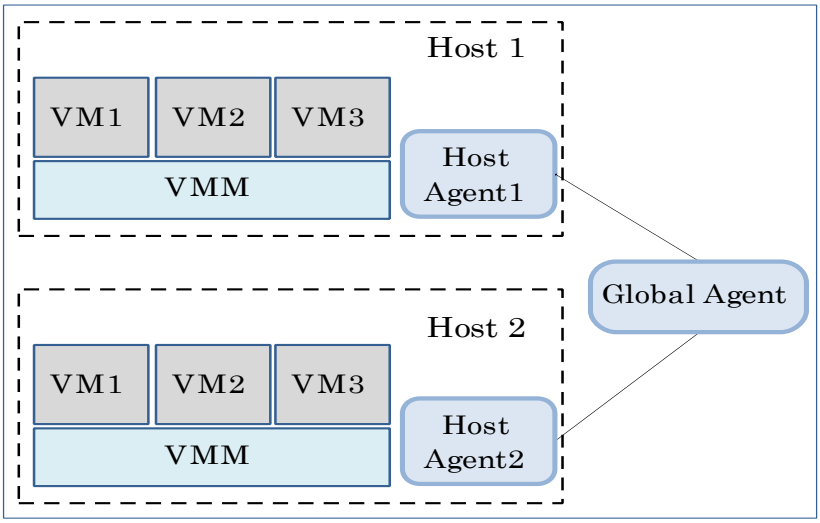


Figure 6.1: Centralized allocation architecture

Figure 6.2 shows a distributed architecture where the communication is performed between the HAs.

In this case, a HA decides to perform live migrations without activating any central controller or GA. If the HA detects an overload or underload situation, it forwards a live migration request to another randomly selected HA to find a host as a possible destination for the VM placement.

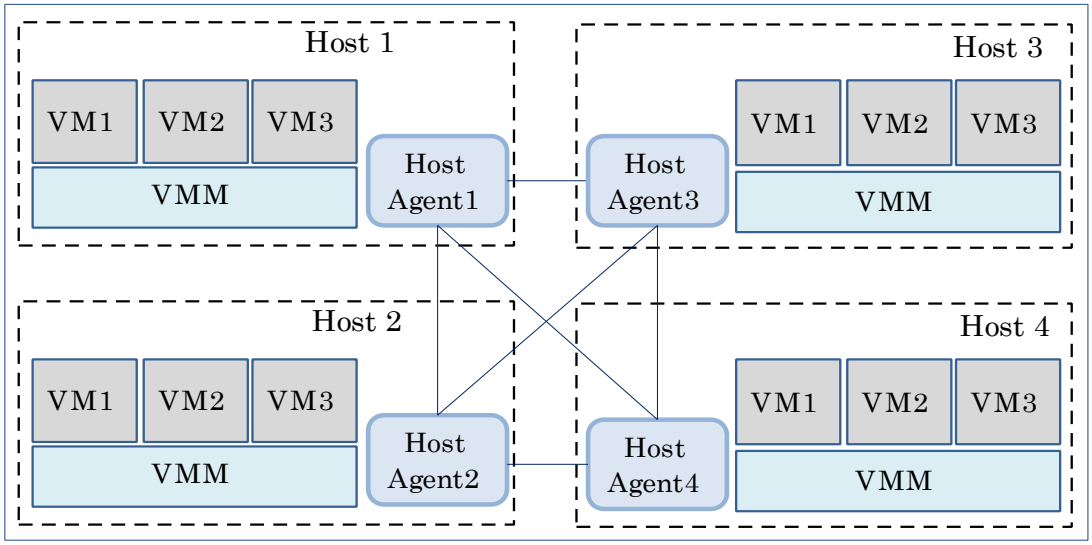


Figure 6.2: Distributed allocation architecture

6.3. Centralized and Threshold-Based Distributed Allocation Approaches

In this section, we describe a centralized resource allocation approach and a threshold-based distributed resource allocation approach.

6.3.1 Centralized Allocation

The centralized allocation (CA) approach is based on the architecture shown in Figure 4.1, where the communication is performed between the GA and the HAs. The HA uses historical data to detect whether the host is in an overloaded or in an underloaded state. A host is considered as overloaded, if the actual and the past three host CPU usage values exceed an upper threshold. A host is considered as underloaded if the actual and the past three host CPU usage values are less than a lower threshold. If a host is in one of these situations, then the GA takes a decision for VM live migration.

In a host overload situation, the VM that has the maximum average CPU utilization is selected. After selecting the first VM, the host is checked again if it is still overloaded in order to proceed with the selection of another VM. This process continues until the host is no longer overloaded. In a host underload situation, all of its VMs are selected for migration in order to turn off the host.

The VM placement algorithm that allocates the VMs to hosts is based on the Best First Decreasing (BFD) algorithm [12], a heuristic algorithm known for solving bin-packing problems.

6.3.2 Threshold-Based Distributed Allocation

A distributed resource allocation approach is suitable for large data centers where centralized optimization complexity and single point of failure are important factors to consider. This approach is based on the architecture shown in Figure 4.2 where each HA makes live migration decisions in cooperation with other HAs.

To determine the host's overloaded or underloaded state, upper and lower thresholds are used. In this work, the lower threshold is set to 10% of the CPU capacity, and the upper threshold is calculated as the total CPU capacity (100%) minus N*5%, where N is the number of VMs.

When the HA detects a host in an overloaded or under-loaded state, it makes a live migration request to a randomly selected host, to serve as a destination for the VM placement. If the request is rejected due to insufficient resource capacity, the HA randomly tries another host. If it fails to find a suitable host, after trying a predefined number of times, it powers on a new host. The VM selection process is the same as in the centralized approach.

6.4. Utility-Based Distributed Allocation Approach

To increase flexibility and to achieve a better overall performance in a data center, we propose a novel Utility-based Distributed Allocation (UDA) approach. This approach is based on the architecture shown in Figure 6.2.

Unlike the DA, to make VM live migration decisions and to detect a host's overloaded or underloaded state, the UDA approach is based on a utility function model. In Figure 4.3, the host utility function model used in our approach is shown, based on host CPU utilization ranging from 0 to 100%. From the graph it is evident that the best value (max of 1) of the utility function is at host CPU utilizations of 70% and 0%. The 70% of CPU utilization is optimal, since the host is fully utilized but not overloaded. The utility value is set to 1 also when the CPU utilization is 0%, with the idea to promote the removal of VMs and host shutdown when the load is low. The goal of the HA is to initiate VM live migration actions in order to maximize the utility function, resulting in optimal resource utilization.

According to the UDA approach during the monitoring process, a HA considers taking VM live migration actions if the utility value is lower than 0.8, for 4 consecutive time intervals.

In this work, the value of 4 consecutive time intervals is the average VM live migration time. It is estimated by averaging over all VM live migration times over several simulation experiments. The reason is that in order to increase the stability of the approach, live migration actions are not started immediately but only if low utility states persist longer than the migration time.
In this case, the HA should make a VM live migration request to a peer HA selected randomly.

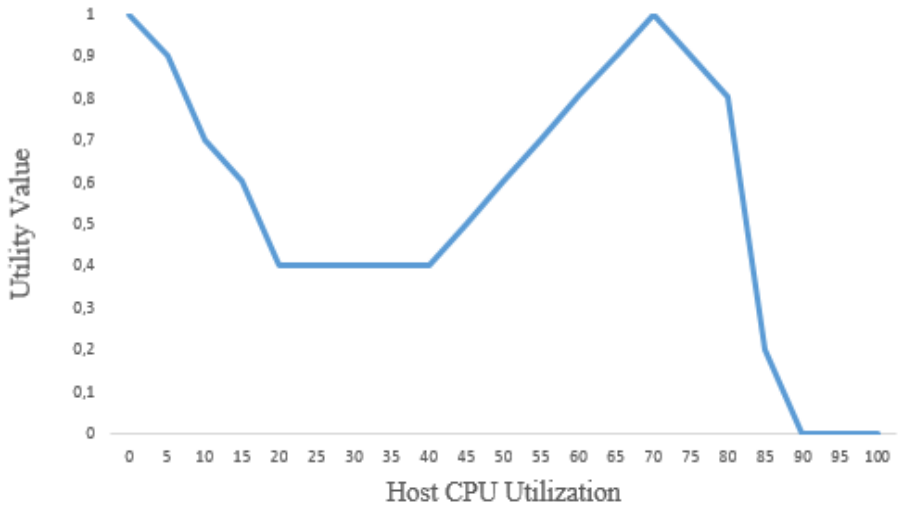


Figure 6.3: The utility function model

In Figure 6.4, the communication scheme between source and destination host is illustrated.

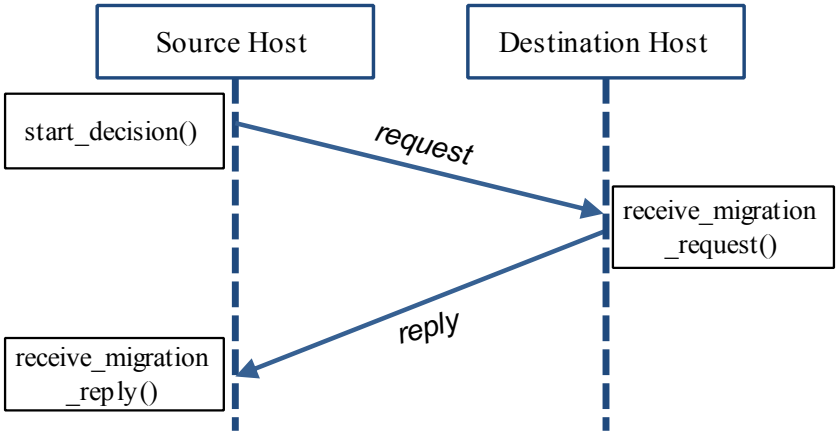


Figure 6.4: Communication scheme between source and destination host

The communication proceeds as follows. When the source HA senses that there is a low utility state, it calls the *start_decision()* procedure. This procedure selects a destination host randomly and sends a migration request message. The message contains all information needed by the destination HA to make migration decisions, such as a list of VMs with their CPU utilization, free RAM capacity etc.

To mitigate any race conditions of receiving migration requests by other hosts, this procedure also sets the *busy_with_migration* variable to true. After receiving a migration request, if *busy_with_migration* is false, the receiving host sets it to true and calls the *receive_migration_request()* procedure that is explained in Algorithm 1. This procedure makes migration decision of migrating one VM from source to destination host or vice versa and notifying the source host for the decision with a reply message. If the destination host is busy with processing another migration request, it send a reply indicating the reason of request rejection as busy.

The HA uses two threads, one for accepting requests and one for making migration decisions. This is done for not blocking the sending HA for a reply. After receiving the reply, the source HA calls *receive_migration_reply()* that is explained in Algorithm 2. Algorithm 1 is executed on the destination host in response to a VM live migration request. This algorithm estimate, which VM should be migrated and in which direction the maximum utility increase should be given.

Algorithm 1 receive_migration_request()

```
1:  push_p = DO_NOTHING
2:  utility_before = get_utility(source_h) + get_utility(dest_h)
3:  for each VM in  source_h.VM_list  do
```

```
4:      utility_after = get_utility(source_h_util -  vm.get_avrg_util())+
          get_utility(dest_h.get_avrg_util(h) + vm.get_avrg_util());
5:      utility_increase = utility_after - utility_before;
6:      if (utility_increase > max_util_increase) &
          (dest_h.get_free_ram() >= vm.ram)  then
7:        max_util_increase = utility_increase;
8:        max_migrating_vm = vm;
9:        push_p = PUSH_VM;
10:     end if
11:  end for
12:  for each VM in dest_h.getVMList() do
13:     utility_after = get_utility(source_h_util + vm.get_avrg_util())+
        get_utility(dest_h.get_avrg_util(h) - vm.get_avrg_util());
14:      utility_increase = utility_after - utility_before;
15:     if (utility_increase > max_util_increase) &
          (source_h.get_free_ram() >= vm.ram)  then
16:       max_util_increase = utility_increase;
17:       max_migrating_vm = vm;
18:       push_p = PULL_VM;
19:     end if
20:  end for
21:  if  max_util_increase > utility_diff_thr  then
22:     send.Reply(push_p, migrating_vm);
23:  end if
24:  else
25:      send.Reply(DO_NOTHING, null);
26       busy_with_migration = false;
27:  end
```

Utility_before is the sum of the source and destination host utility values before VM live migration. *Utility_after* is the sum of the source and destination host utility values after VM live migration. The increase of the host utility value as a result of VM live migration is defined through *utility_increase*. The *get.avrg_util()* method gives the average CPU utilization of VMs, source and destination hosts, calculated for the past 4 consecutive intervals. The utility increase should be greater than a predefined *utility_diff_thr* threshold in order to take a VM migration action. This is done to increase the stability of the approach and reduce unnecessary VM live migrations. The variable *busy_with_migration* also is set to false on both source and destination hosts, when the VM live migration process is finished.

Algorithm 2 is executed on the source host after receiving the response from the destination host.

In Algorithm 2, the *push_pull* variable indicates the direction of VM live migration. If its value is DO_NOTHING, there is no live migration action because there is no increase in utility function, or the destination host is busy with another migration process.

To differentiate between the cases, the variable reject is tested to check if the destination host is busy with another migration. The *busy_counter* variable limits how many times to try other hosts if previous hosts are busy. The *overload_counter* limits how many times to try other hosts if there are no increases in utility.

Algorithm 2 receive_migration_reply()

```
1:  if (push_pull == DO_NOTHING) & (reject == BUSY)  then
2:      if  busy_counter != 0  then
3:          start_decision();
4:          busy_counter --;
5:      end if
6:      else
7:          busy_counter = busy_counter_thr;
8:          busy_with_migration = false;
9:      end
10: else if  push_pull == DO_NOTHING  then
11:     if  overload_counter != 0  then
12:         start_decision();
13:         overload_counter --;
14:     end if
15:    else if (source_h.get_avrg_util(h) >
       (source_h.getUpperThr(h))
16:          new_h = host_power_on();
17:       migrate_vm_to_host(source_h, new_h,
          source_h.selectVM());
18:     end if
19:     else
20:       busy_with_migration = false;
21:       overload_counter = overload_counter_thr;
22:     end
23: end if
24: else if  push_pull == PUSH_VM   then
25:     migrate_vm_to_host(source_h, dest_h, migrating_vm);
26: end if
27: else if push_pull == PULL_VM   then
```

```
28:     migrate_vm_to_host(dest_h, source_h, migrating_vm);
29:   end if
```

In both cases, the *start_decision()* is called to make a new migration request to another randomly selected host. If a suitable host is not found for a number of trials, because there is no increase in utility (the *overload_counter* variable reaches a threshold) and the source host is in an overloaded state, then a new host is powered on. In this case, *selectVM()* is called to select the VM that should be migrated from the source host to the new host.

If the value of the *push_pull* variable is *PUSH_VM* or *PULL_VM*, then it indicates a VM live migration action from source to destination host or from destination to source host, respectively.

6.5. Experimental Results

In this section, we present experimental results of our proposed UDA approach compared to three other approaches. The first one, called No Migration (NOM) approach, allocates CPU resources to VMs locally, but does not perform live migration actions.

The second one is the Centralized Approach (CA), and the last approach is Distributed Allocation (DA), as described in Section IV.

To have a controlled experimental environment and the possibility of repeated experimental runs, we have developed a simple event-based cloud simulator. We consider data centers of different sizes with the number of hosts varying from 100 to 700 and an initial allocation of 3 VMs per host. To simulate VM workloads, CPU usage data of real VMs running in PlanetLab [37], is used.

Each VM runs one application with a variable workload. The experiment is run for 570-time intervals, and the duration of a time interval is set to 5 seconds. The simulation experiment is run for four different load levels called VLOW, LOW, HIGH and VHIGH. The load level represents the CPU usage consumed by each VM. The load levels taken by multiplying the PlanetLab CPU usage values for each time interval with a constant value are as follows: VLOW with 0.2, LOW with 1, HIGH with 2 and VHIGH with 3. The experiment is repeated ten times for each combination of approach and load level. We have defined four performance metrics for evaluation of the proposed approach, as discussed below.

6.5.1 VM Sla Violation (VSV)

This metric represents the penalty of the cloud provider for violating the performance of the cloud consumer VMs. The VM SLA violation happens if the difference between allocated CPU and CPU usage of the VM is less than 5% of the CPU capacity for four consecutive intervals. The reason for this metric is that the performance of an application is poor if the required CPU usage is near the allocated CPU share.

Considering the VM SLA violation metric above, we have defined an upper threshold for host overload detection, which is calculated dynamically depending on the number of VMs. To avoid a VM SLA violation, each host must have more than 5% capacity above the CPU usage for each VM, so the total free CPU capacity of the host should be more than N*5%, where N is the number of VMs. The overload threshold is calculated as the total CPU capacity (100%) minus N* 5% [14]. For the underload threshold, we used a fixed value of 10% of the CPU capacity.
In the results, we show the cumulative VSV (CVSV) value that is estimated as the sum of VSV values of all VMs for the whole experimental time.

6.5.2 Energy Consumption (E)

This is an important metric, since the target of server consolidation in a data center is to reduce energy consumption. The total energy consumption of the data center, measured in KWh, for the whole experimental time is shown in the experimental results.

6.5.3 Number of VM Migrations

The process of live migration is costly because it takes a significant quantity of CPU processing on the source host, traffic load during the communication between the source and destination host [39] and can cause VM SLA violations.

6.5.4 Energy and VM Sla Violations (ESV)

This metric combines energy consumption (E) and cumulative VM SLA violation (CVSV) value in a single metric:

$$ESV = E \times CVSV \qquad (6.1)$$

To see the effect the load level has on the VM SLA violation, Figure 6.5 presents the cumulative VSV

value for each approach and the four load levels. For each approach, we notice that while the load increases, the SLA violations are increased. From the results, we can see that for all load levels, the UDA approach achieves the lowest CVSV value compared to the other approaches.

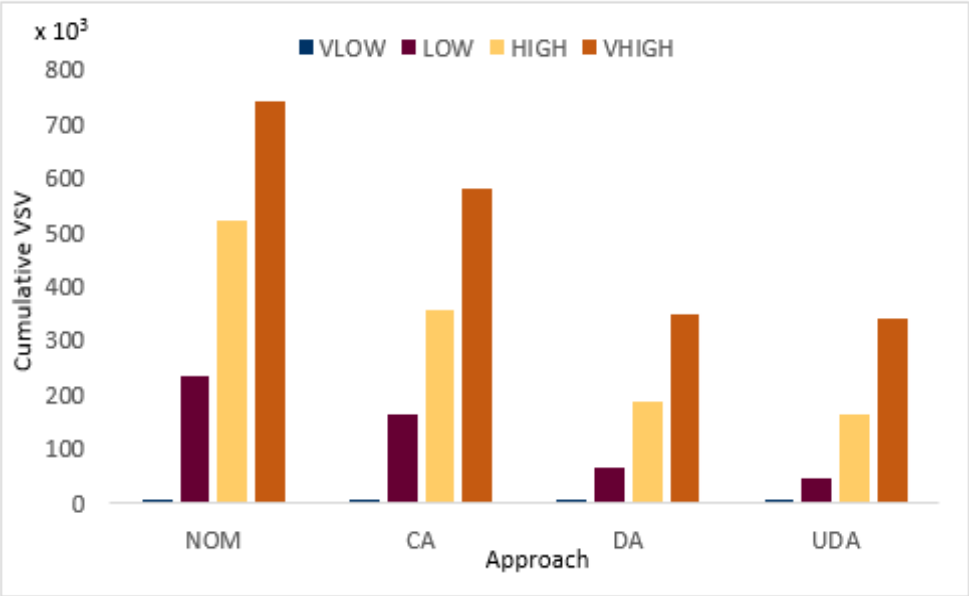


Figure 6.5: Cumulative VSV over all load levels

Figure 6.6 shows the energy consumption of the data center for the whole experimental time for each approach over all load levels. It is evident that by increasing the load, the energy consumption is increased for all approaches.

The UDA approach, despite consuming more energy than the CA and DA approaches, has smaller SLA violations and therefore a lower ESV metric than other approaches, as shown in Figure 6.8.

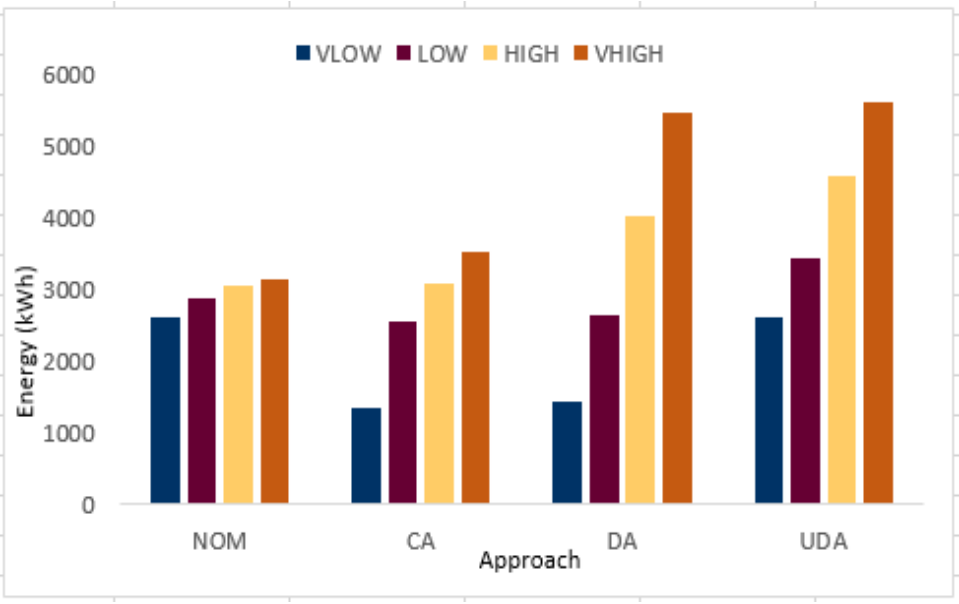


Figure 6.6: Energy over all load levels

Figure 6.7 shows how the load levels affect the number of live migrations.

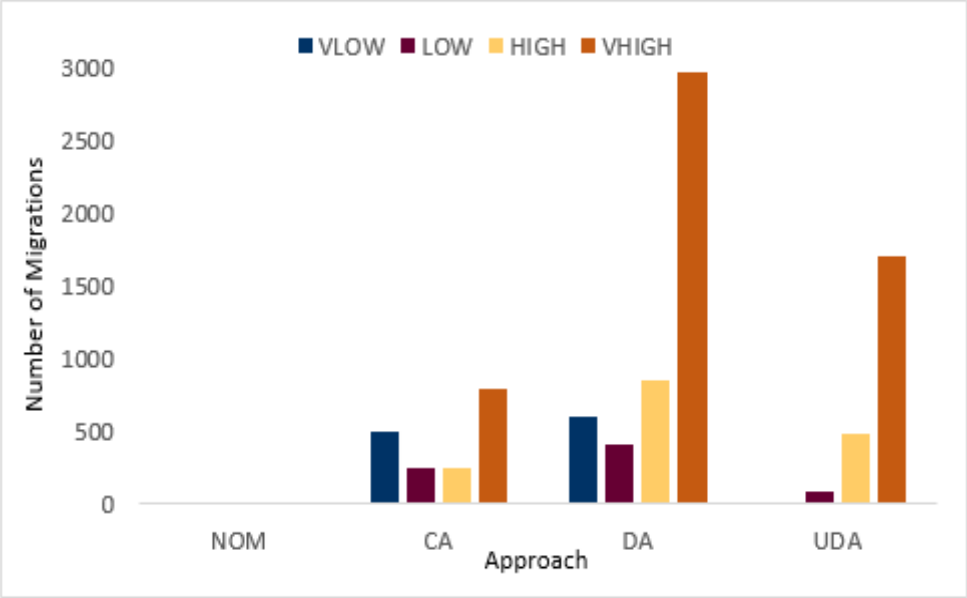


Figure 6.7: Number of live migrations over all load levels

It can be noticed that the number of live migrations of the UDA approach is significantly smaller than CA and DA for low load levels (VLOW and LOW), while for high load levels (HIGH and VHIGH), the number of live migrations of the UDA approach is smaller than DA but greater than the CA approach. This is because for high load levels it needs more VM live migrations to achieve lower values of the SLA violations and ESV metric.

Figure 6.8 shows the ESV metric over four load levels, where it is evident that the ESV metric is smaller for the UDA approach than the other approaches for each load level.

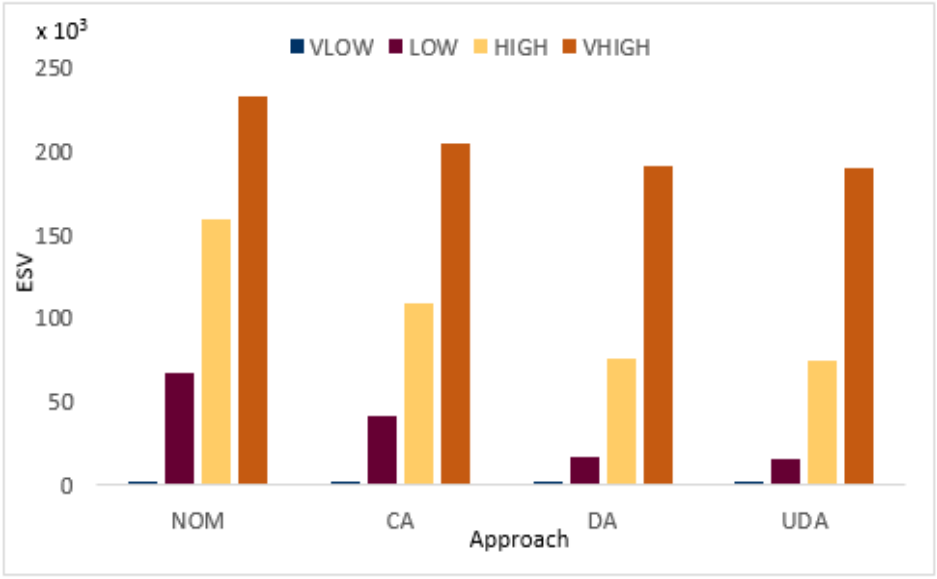
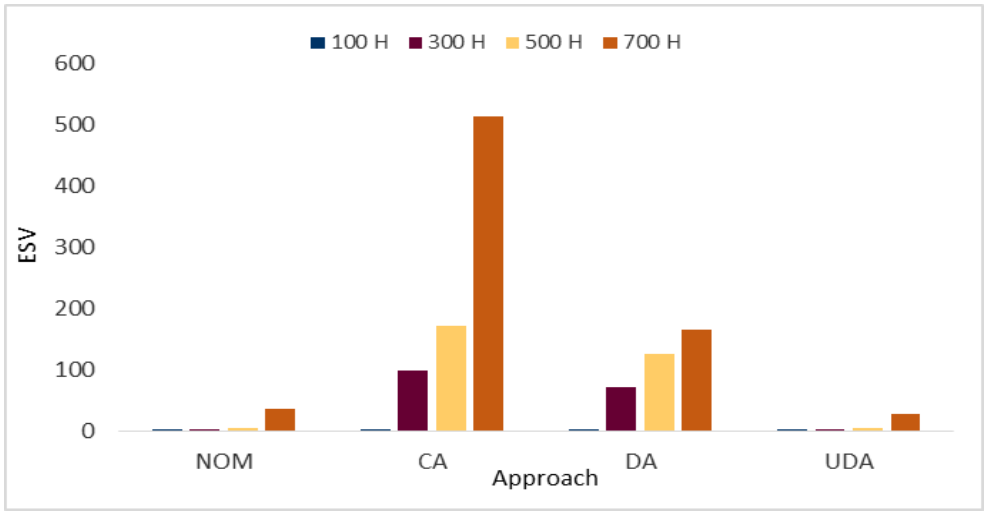


Figure 6.8: ESV over all load levels

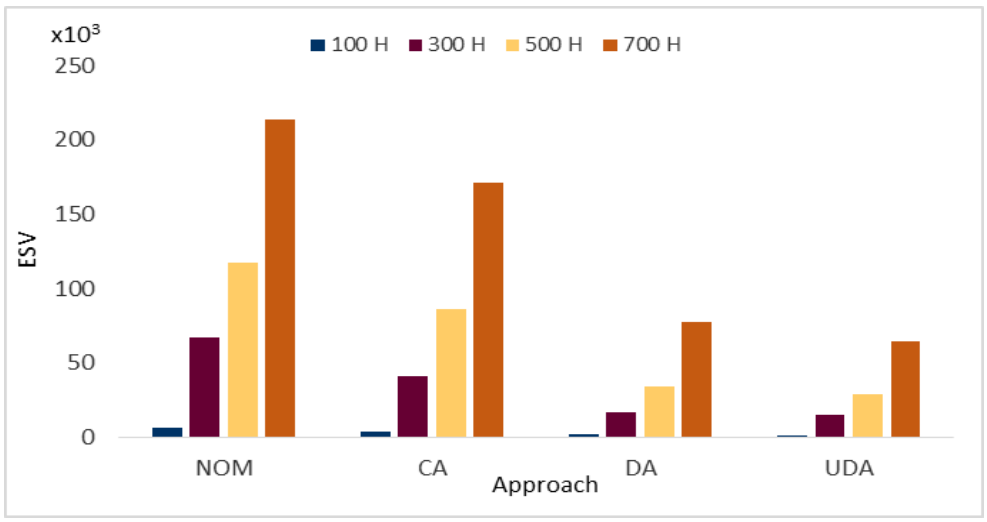
We have also estimated the ESV metric for different number of hosts, for all approaches and four load levels. We have tested for 100, 300, 500 and 700 hosts.

In Figure 6.9, we show the ESV value for all load levels, such as (a) VLOW Load, (b) LOW Load, (c) HIGH Load, and (d) VHIGH Load.

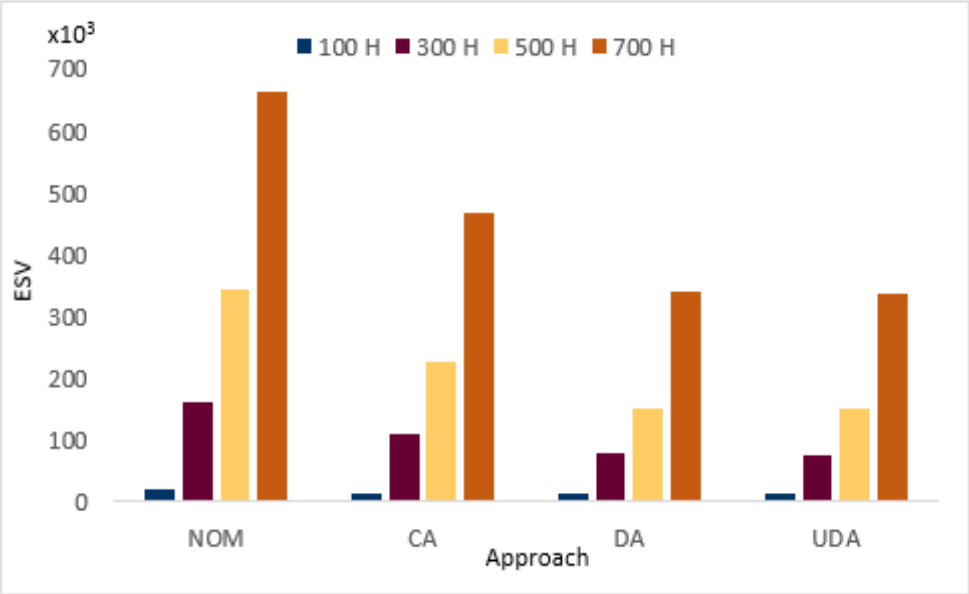
The UDA approach for all load levels achieves the smallest ESV value compared to the other approaches.



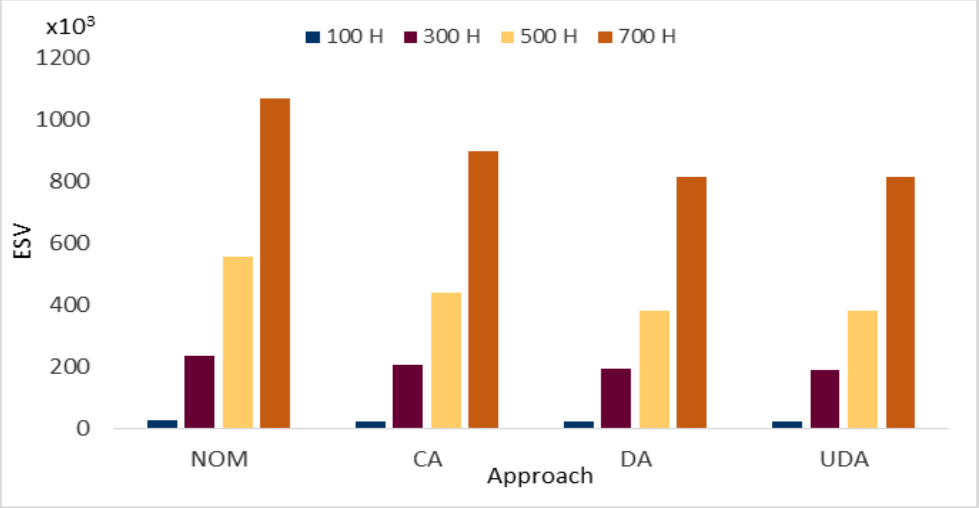
(a)



(b)



(c)



(d)

Figure 6.9: ESV per number of hosts: (a) VLOW load, (b) LOW load, (c) HIGH load, (d) VHIGH load

6.6. Summary

We have presented a multi-agent distributed approach for dynamic resource allocation in cloud infrastructures based on utility function optimization. The advantage of our approach based on multi-agent systems is that in each PM there is an agent who is responsible for making the decision to initiate the live migration process of VMs from one PM to another PM.

Through our proposed approach, a centralized controller is avoided, which in a large cloud infrastructure could be a communication bottleneck and a single point of failure. This would decrease the overall reliability of the system.

Compared to other approaches, the utility-based distributed resource allocation approach shows reduced VM SLA violations and minimized ESV metrics.

Part IV.

Conclusion

7

Conclusion

This chapter summarizes our research contributions for resource allocation in cloud infrastructure as well as presents future work.

7.1. Summary

Cloud computing as a computation model that supports virtualization technology has provided great benefits to the IT infrastructure in general. These benefits are best observed by cloud providers and consumers. Cloud customers can obtain the resources they need on demand, at optimal cost, and with high performance, through a pay-as-you-go model. On the other hand, cloud providers obtain efficient resource utilization and a significant reduction in operational and energy costs. Therefore, to maintain these benefits for cloud consumers and cloud providers, adequate dynamic resource allocation approaches are needed, specifically in virtual machine consolidation.

Although various VM resource allocation approaches have been proposed for cloud environments, there are still open research challenges that require implementation of innovative techniques in this direction. In this dissertation, several VM resource allocation approaches have been presented as a way of addressing some of the drawbacks of existing approaches, with a focus on the dynamic allocation of resources in cloud infrastructures.

In a cloud infrastructure, the dynamic consolidation of VMs through live migration is a fundamental approach to reduce energy consumption and operational costs. More specifically, live migration in a data center has many benefits, such as load balancing, manageability and maintenance, minimum violation of the SLAs, energy management, improved performance and reliability, improving utilization of resources and reducing management costs. Most of the existing

approaches for VM consolidation are based on low-level utilization metrics and thresholds, so they have not guaranteed to deliver high performance of the applications for the cloud consumer and the cost-benefits of the cloud provider. In the context of live migration, the approaches to detect if a PM is overloaded or underloaded are based on short-term predictions. This means that they are based on monitoring if the current or the predicted next value exceeds a specified threshold, to determine whether a PM should be considered as overloaded or underload, respectively. Therefore, making decisions to initiate live migration actions based on short-term predictions can lead to hasty decisions, unnecessary live migration overhead, and stability issues.

In this dissertation, an approach to dynamically allocate resources to VMs in cloud infrastructures has been presented. It combines local and global VM resource allocation strategies based on a multi-agent resource architecture. In this approach, to detect whether a PM is overloaded or underloaded is based on long-term resource usage predictions. In the context of this work, long-term predictions mean predicting 7-time intervals ahead into the future. Based on this premise, a PM is declared as overloaded if the current and the predicted total CPU usage of 7-time intervals ahead into the future exceed an overload threshold. The same applies when a PM is declared as underloaded if the current and the predicted total CPU usage of 7-time intervals ahead into the future are less than an underload threshold. On the other side, a PM is declared as not overloaded if the current and the predicted total CPU usage of 7-time intervals ahead into the future is less than the overload threshold. For long-term predictions, a supervised machine learning approach based on Gaussian Processes is used.

Another important issue to note especially in long-term resource predictions is the fact that predicting further into the future increases the prediction error and the uncertainty, thus diminishing the benefits of long-term prediction. Based on this, we have also considered uncertainty as an integral feature of long-term resource predictions. To consider the uncertainty of long-term predictions, a probabilistic model of the prediction error is built online using the non-parametric kernel density estimation method. The results show that the approach presented for dynamic resource allocation which considers long-term predictions of resource demand and uncertainty of predictions increases stability and overall performance in the cloud infrastructure.

A key function in cloud computing resource management is task scheduling, where the application of effective task scheduling techniques can reduce task completion time, increase the efficiency of resource utilization, increase the quality of services, and improve the performance of the system. Since cloud computing delivers services over the Internet, service customers must submit

their request online, where each service has a number of costumers and also a number of tasks that must be processed at a time. Therefore, it is imperative that systems have implemented scheduling techniques and policies that consider certain parameters, such as the nature of the task, the size of the task, the task execution time, the availability of resources, the task queue, and the load on the resources. Thus, proper task scheduling may result in an efficient utilization of resources. The task scheduling problem itself is NP-hard, so heuristic algorithms must be implemented to solve it. Based on this, we have presented a task scheduling algorithm for cloud environments based on the Heterogeneous Earliest Finish Time (HEFT) algorithm, called experiential HEFT (EHEFT). It considers experiences with previous executions of tasks to determine the workload of resources. To realize the EHEFT algorithm, we propose a new way of HEFT rank calculation to specify the minimum average execution time of previous runs of a task on all relevant resources. Experimental results show that our EHEFT algorithm performs better and is more efficient in terms of scheduling length ratio and runtime rather than several well-known existing approaches.

The IaaS model of cloud computing allocates resources in the form of VMs that can be resized and live migrated at runtime. Thus, for dynamic VM consolidation an important mechanism that provides major benefits for data centers is live migration of VMs, from one PM to another PM. Live migration of VMs enables the allocation of resources to running services without interruption during the migration process. This is important, especially for services with particular quality of service (QoS) requirements. In the context of VM resource allocation in most existing approaches, the problem lies in the fact that they are based on centralized resource manager architectures. These architectures are based on a central controller, which does not perform well for large cloud infrastructures, with the possibility of a communication bottleneck, and is a single point of failure in terms of reliability. In contrast to the existing dynamic VM consolidation approaches, we have presented a distributed resource allocation approach based on multi-agent systems.

7.2. Future Work

To further improve the solutions proposed in this dissertation, we have identified several areas for the future work. Based on experimental results, the distributed VM resource allocation approach that uses the utility function based on multi-agent systems has resulted in better overall performance compared to a centralized approach and threshold-based distributed approach. A distributed VM

resource allocation approach is suitable for large-scale cloud infrastructures and avoids the drawbacks arising from the centralized approach in terms of single point of failure and communication bottlenecks. In this distributed architecture based on multi-agent systems, on each physical machine, there is an agent who is responsible for making the decisions for the live migration of VMs from one PM to another PM. A future challenge to be investigated is how local agents with a limited view should coordinate each other to achieve a global optimization objective.

For a distributed VM allocation approach, what can be further investigated is the development of an accurate model that selects the best utility function to improve the reduction in energy consumption in a data center. Another direction that can be investigated in terms of the model for the utility function is the inclusion of other resources, such as memory and network I/O, since they have a significant impact on the overall performance in the process of dynamic VM consolidation.

In the context of VM consolidation with a special emphasis on the live migration process, a challenge is to detect when a PM is in an overloaded or underloaded state, in order to make the decision to initiate the live migration process. Some of the existing approaches have based their prediction on monitoring resource usage where the actual or the predicted next value exceeds specified thresholds, i.e, the upper and lower threshold. Mostly, these approaches rely on live migration decisions based on short-term predictions and this leads to unnecessary live migration overhead and stability issues. Unlike existing approaches, we have based the live migration decisions on resource usage predictions several steps ahead in the future, thus making long-term predictions. This increases the stability of the live migration process. However, an interesting area of future work would be the investigation of long-term predictions of the usage of multiple resources such as CPU, memory, and I/O bandwidth and their interdependencies in allocation decisions.

To further optimize the chain of dynamic resource allocation processes and especially VM placement, other factors such as network traffic and thermal issues should be considered. The process of VM consolidation where the migration of VMs to PMs located in different racks should take place also has network cost and VM traffic load. This network cost in a data center should not be neglected because it can lead to higher SLA violations. Therefore, in order to reduce network traffic, more efficient techniques and approaches with low computational complexity should be investigated. These techniques based on network-aware and traffic-aware patterns should analyze the impact of traffic and network topology inside a data center.

Another direction in which further research can be investigated and which is important for VM consolidation is the thermal problem of computing resources. A significant portion of electrical energy consumed by computing resources is converted into heat. Operation of devices at high temperatures reduces their lifetimes and the availability and reliability of the system. Although nowadays there are advanced cooling systems in modern data centers, however, for cloud providers they are expensive to buy, costly to maintain, and they consume energy as well. Therefore, it is necessary to develop thermal-aware VM consolidation techniques based on machine learning methods that reduce the energy consumption of the devices in the data center, and at the same time to reduce heat.

REFERENCES

[1] Y. O. Yazır, C. Matthews, R. Farahbod, ”Dynamic Resource Allocation in Computing Clouds using Distributed Multiple Criteria Decision Analysis,“ In: IEEE 3rd International Conference on Cloud Computing, IEEE, pp. 91-98, 2010.

[2] N. B. Ruparelia, ”Cloud Computing,” In: The MIT Press, Book, 2016.

[3] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, ”Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility,” In: Future Generation Computer Systems, Volume 25, Issue 6, pp. 599-616, 2009.

[4] P. Mell and T. Grance, ”The NIST Definition of Cloud Computing,” In: Recommendations of the National Institute of Standards and Technology, Information Technology Laboratory, Special Publication 800-145, 2011.

[5] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, and R. Katz, ”Above the Clouds: A Berkeley View of Cloud Computing,” In: UC Berkeley Reliable Adaptive Distributed Systems Laboratory White Paper, 2009.

[6] W. Voorsluys, J. Broberg, and R. Buyya, “Introduction to Cloud Computing,” In: Chapter 1, Book: Cloud Computing Principles and Paradigms. Edited by: Rajkumar Buyya, James Broberg, Andrzej Goscinski, Wiley, 2011.

[7] S. Murugesan and I. Bojanova, ”Cloud Computing: An Overview,” In: Chapter 1, Book: Encyclopedia of Cloud Computing, IEEE Press/WILEY, 2016.

[8] W. Stallings, ”Overview of Cloud Computing,” In: Chapter 2, Book: Cloud Computing Security, Foundations and Challenges, edited by J. R. Vacca, CRC Press, Taylor & Francis Group, 2017.

[9] National Institute of Standards and Technology, ”The NIST Cloud Computing Reference Architecture,” In: Recommendations of the National Institute of Standards and Technology, Information Technology Laboratory, Special Publication SP-500-292, 2011.

[10] Trusting the Cloud, ” Provides Cloud Audit Services,” In: http://www.trustingthecloud.eu/joomla/, Accessed November 2017.

[11] Gartner, “Gartner Says Cloud Consumers Need Brokerages to Unlock the Potential of Cloud Services,” In: http://www.gartner.com/it/page.jsp?id=1064712, Accessed November 2017.

[12] S. Nanda and T. Chiueh, ”A Survey on Virtualization Technologies,” In: Report: Department of Computer Science, SUNY at Stony Brook, 2005.

[13] Redswitches, ”The Different Types of Virtualization in Cloud Computing – Explained,” In: https://redswitches.com/blog/different-types-virtualization-cloud-computing-explained/, Accessed November 2017.

[14] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer , I. Pratt and A. Warfield, ”Xen and the Art of Virtualization,” In: SOSP '03 Proceedings of the Nineteenth

ACM Symposium on Operating Systems Principles, Pages 164-177, ACM, October 2003.

[15] IBM Knowledge Center, ”KVM Overview,” In: https://www.ibm.com/support/knowledgecenter/en/linuxonibm/liaat/liaatkvmover.htm, Accessed November 2017.

[16] B. Jennings and R. Stadler, ”Resource Management in Clouds: Survey and Research Challenges,” In: Journal of Network and Systems Management, Volume 23, Issue 3, pp.567-619, Springer, 2014.

[17] M. Ullrich, J. Lassig and M. Gaedke, ”Towards Efficient Resource Management in Cloud Computing: A Survey,” In: IEEE 4th International Conference on Future Internet of Things and Cloud, IEEE, Austria, 2016.

[18] Q. Zhang, E. Gurses, R. Boutaba, and J. Xiao, ”Dynamic Resource Allocation for Spot Markets in Clouds,” In: Proceeding in Hot-ICE'11 Proceedings of the 11th USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services , ACM, 2011.

[19] A. Beloglazov, J. Abawajy and R. Buyya, “Energy-aware Resource Allocation Heuristics for Efficient Management of Data Centers for Cloud Computing,” In: Future Generation Computer Systems 28, pp. 755–768, 2012.

[20] D. Deng, K. He, Y. Chen, ”Dynamic Virtual Machine Consolidation for Improving Energy Efficiency in Cloud Data Centers,” In: 4th International Conference on Cloud Computing and Intelligence Systems (CCIS), pp. 366-370, 2016.

[21] A. Beloglazov and R. Buyya, ”Optimal Online Deterministic Algorithms and Adaptive Heuristics for Energy and Performance Efficient Dynamic Consolidation of Virtual Machines in Cloud Data Centers,” In: Concurrency And Computation: Practice And Experience, pp. 1397-1420, Volume 24, Issue 13, 2012.

[22] M. A. H. Monil and R. M. Rahman, ”VM Consolidation Approach based on Heuristics, Fuzzy Logic, and Migration Control,” In: Journal of Cloud Computing: Advances, Systems and Applications, Springer, 2016.

[23] M. R. Chowdhury, M. R. Mahmud, R. M. Rahman, “Implementation and Performance Analysis of Various VM Placement Strategies in CloudSim,” In: Journal of Cloud Computing: Advances, Systems and Applications, Springer, 2015.

[24] W.S. Cleveland, C. Loader, “Smoothing by Local Regression: Principles and Methods,” In: Statistical Theory and Computational Aspects of Smoothing, pp. 10-49, Physica-Verlag Heidelberg, 1996.

[25] X. Fu, Ch. Zhou, ”Virtual Machine Selection and Placement for Dynamic Consolidation in Cloud Computing Environment,” In: Frontiers of Computer Science, Volume 9, Issue 2, pp. 322–330, April 2015.

[26] A. Verma, G. Dasgupta, T.K. Nayak, P. De, R. Kothari, ”Server Workload Analysis for Power Minimization using Consolidation,” In: USENIX'09 Proceedings of the 2009 Conference on USENIX Annual Technical Conference, June 2009.

[27] Z. Usmania and S. Singh,”A Survey of Virtual Machine Placement Techniques in a Cloud Data Center,” In: Procedia Computer Science, Volume 78, pp. 491 – 498, , Elsevier, 2016.

[28] M. Masdari, S. S. Nabavi, V. Ahmadi, ”An Overview of Virtual Machine Placement Schemes in Cloud Computing,” In: Journal of Network and Computer Applications, Volume 66, pp. 106–127, May 2016.

[29] M. A. Haque Monil, R. Qasim, R. M. Rahman, ”Energy-Aware VM Consolidation Approach using Combination of Heuristics and Migration Control,” In: Ninth International Conference on Digital Information Management, pp. 74–79, IEEE, 2014.

[30] A. Mosa and N. W. Paton, ”Optimizing Virtual Machine Placement for Energy and SLA in Clouds using Utility Functions,” In: Journal of Cloud Computing: Advances, Systems and Applications, 2016.

[31] P. X. Gao, A. R. Curtis, B. Wong, and S. Keshav, ”It's not Easy being Green,” In: Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, pp. 211-222, ACM, 2012.

[32] G.B. Fioccola, P. Donadio, R. Canonico, and G. Ventre, ”Dynamic Routing and Virtual Machine Consolidation in Green Clouds,” In: 8th International Conference on Cloud Computing Technology and Science (CloudCom), IEEE, pp. 590-595, 2016.

[33] A. Murtazaev and S. Oh, “Sercon: Server Consolidation Algorithm using Live Migration of Virtual Machines for Green Computing,” In: IETE Technical Review, Volume 28, Issue 3, pp. 212-231, Taylor & Francis, 2011.

[34] E. Feller, C. Morin, and A. Esnault, ”A Case for fully Decentralized Dynamic VM Consolidation in Clouds,” In: Cloud Computing Technology and Science (CloudCom), 4th IEEE International Conference, pp. 26-33, IEEE, December 2012.

[35] M. Marzolla, O. Babaoglu and F. Panzieri, ”Server Consolidation in Clouds through Gossiping,” In: World of Wireless, Mobile and Multimedia Networks (WoWMoM), IEEE International Symposium, IEEE, pp. 20-24, June 2011.

[36] T. Wood, P. Shenoy, A. Venkataramani and M. Yousif, ”Sandpiper: Black-Box and Gray-Box Resource Management for Virtual Machines,” In: Computer Networks, Volume 53, Issue 17, pp. 2923–2938, Elsevier, December 2009.

[37] K. Park and V. S. Pai, ” CoMon: A Mostly-Scalable Monitoring System for PlanetLab,” In: ACM SIGOPS Operating Systems Review, Volume 40, Issue 1, pp. 65-74, ACM, 2006.

[38] D. Minarolli, A. Mazrekaj and B. Freisleben, ”Tackling Uncertainty in Long-Term Predictions for Host Overload and Underload Detection in Cloud Computing,” In: Journal of Cloud Computing: Advances, Systems and Applications, Springer, 2017.

[39] F. Farahnakian, T. Pahikkala, P. Liljeberg, J. Plosila, N. T. Hieu, and H. Tenhunen, ”Energy-aware VM Consolidation in Cloud Data Centers Using Utilization Prediction Model,” In: IEEE Transaction on Cloud Computing, Volume XX, No. X, IEEE, 2016.

[40] A. Mazrekaj, D. Minarolli and B. Freisleben, ”Distributed Resource Allocation in Cloud

Computing using Multi-Agent Systems," In: Telfor Journal, pp.110-115, 2017.

[41] National Institute of Standards and Technology, "The NIST Cloud Computing Standards Roadmap," In: Recommendations of the National Institute of Standards and Technology, Information Technology Laboratory, Special Publication SP-500-292, 2011.

[42] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. D. Rose and R. Buyya, "CloudSim: A Toolkit for Modelling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms," In: Journal Software-Practice & Experience, Volume 41, Issue 1, pp. 23-50, ACM, 2011.

[43] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. D. Rose and R. Buyya, "CloudSim: A Novel Framework for Modeling and Simulation of Cloud Computing Infrastructures and Services," In: http://www.cloudbus.org/cloudsim/, Accessed November 2017.

[44] T. Wood, P. Shenoy, A. Venkataramani and M. Yousif, "Sandpiper: Black-box and Gray-Box Resource Management for Virtual Machines," In: Computer Networks, Volume 53, Issue 17, pp. 2923-2938, Elsevier, 2009.

[45] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "The Eucalyptus Open-Source Cloud-Computing System," In: Cluster Computing and the Grid, 9th IEEE/ACM International Symposium, pp. 124–131, 2009.

[46] G. Khanna, K. Beaty, G. Kar, and A. Kochut, "Application Performance Management in Virtualized Server Environments," In: Proceedings 10th IEEE/IFIP Network Operations and Management Symposium (NOMS 2006), pp. 373–381, IEEE, 2006.

[47] Z. Gong and X. Gu, "PAC: Pattern-driven Application Consolidation for Efficient Cloud Computing," In: Proceedings IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, pp. 24-33, IEEE, 2010.

[48] M. Andreolini, S. Casolari, M. Colajanni, and M. Messori, "Dynamic Load Management of Virtual Machines in Cloud Architectures," In: CloudComp, Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, Volume 34, pp. 201-214, Springer, 2009.

[49] F. Farahnakian, A. Ashraf, P. Liljeberg, T. Pahikkala, J. Plosila, I. Porres, and H. Tenhunen, "Energy-aware Dynamic VM Consolidation in Cloud Data Centers using Ant Colony System," In: 7th International Conference on Cloud Computing (CLOUD),pp. 104–111, IEEE, 2014.

[50] N. Bobroff, A. Kochut, and K. Beaty, "Dynamic Placement of Virtual Machines for Managing SLA Violations," In: Integrated Network Management, 10th IFIP/IEEE International Symposium, pp. 119–128, IEEE, 2007.

[51] Zh. Gong, X. Gu, J. Wilkes, "Press: Predictive Elastic Resource Scaling for Cloud Systems," In: Proceedings of International Conference on Network and Service Management (CNSM'10), pp. 9–16, IEEE, 2010.

[52] Zh. Shen, S. Subbiah, X. Gu, and J. Wilkes, "Cloudscale: Elastic Resource Scaling for Multi-Tenant Cloud Systems, " In: Proceedings of the 2nd ACM Symposium on Cloud Computing,

pp. 1–14, ACM, 2011.

[53] S. Islam, J. Keung, K. Lee, and A. Liua, ”Empirical Prediction Models for Adaptive Resource Provisioning in The Cloud,” In: Future Generation Computer Systems, Volume 28, Issue 1, pp. 155–162, 2012.

[54] S. Khatua, M. M. Manna, and N. Mukherjee, ”Prediction-Based Instant Resource Provisioning for Cloud Applications,” In: Proceedings of the IEEE/ACM 7th International Conference on Utility and Cloud Computing, IEEE Computer Society, pp. 597–602, 2014.

[55] F. Qiu, B. Zhang, and J. Guo, ”A Deep Learning Approach for VM Workload Prediction in the Cloud,” In: 17th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, pp. 319–324, IEEE, 2016.

[56] F. Farahnakian, T. Pahikkala, P. Liljeberg, J. Plosila, H. Tenhunen, ”Multi-Agent based Architecture for Dynamic VM Consolidation in Cloud Data Centers,” In: 40th Euromicro Conference on Software Engineering and Advanced Applications, pp. 111–118, IEEE, August 2014.

[57] F. Farahnakian, P. Liljeberg, T. Pahikkala, J. Plosila and H. Tenhunen, ”Hierarchical VM Management Architecture for Cloud Data Centers,” In: 6th International Conference on Cloud Computing Technology and Science, pp. 306-311, IEEE, 2014.

[58] G. Jung, M. Hiltunen, K. Joshi, R. Schlichting, and C. Pu, “Mistral: Dynamically Managing Power, Performance, and Adaptation Cost in Cloud Infrastructures,” In: Distributed Computing Systems (ICDCS), 30th International Conference on, pp. 62–73, IEEE, 2010.

[59] S.A. Baset, ”Cloud Service Level Agreement,” In: Chapter 36, Book: Encyclopedia of Cloud Computing, IEEE Press/WILEY, 2016.

[60] A. Sahai, S. Graupner, V. Machiraju and V. Moorsel, “Specifying and Monitoring Guarantees in Commercial Grids through SLA,” In: Proceedings Cluster Computing and the Grid (CCGrid), 3rd IEEE/ACM International Symposium, IEEE, 2003.

[61] E. Feller, L. Rilling, and Ch. Morin, ”Snooze: A Scalable and Autonomic Virtual Machine Management Framework for Private Clouds,” In: 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), pp. 482-489, IEEE/ACM, 2012.

[62] M. Dorigo, G. D. Caro, and L. M. Gambardella, “Ant Algorithms for Discrete Optimization,” In: Artificial Life Journal, Volume 5, Issue 2, pp. 137-172, ACM, April 1999.

[63] F. Farahnakian, ”Energy and Performance Management of Virtual Machines Provisioning, Placement and Consolidation,” In: PhD thesis, Turku, Finland, 2016.

[64] N. M. Seel, ”Encyclopedia of the Sciences of Learning,” In: Mathematical Models, Springer-Verlag, 2012.

[65] I. Hwang and M. Pedram, “Hierarchical Virtual Machine Consolidation in a Cloud Computing System,” In: Proceedings of IEEE Sixth International Conference on Cloud Computing, pp. 196–203, 2013.

[66] G. E. I. Selim, M. A. El-Rashidy N. A. El-Fishawy, “An Efficient Resource Utilization

Technique for Consolidation of Virtual Machines in Cloud Computing Environments," In: 33rd National Radio Science Conference, pp. 316-324, IEEE, 2016.

[67] Sh. Di, D. Kondo, and W. Cirne, "Host Load Prediction in a Google Compute Cloud with a Bayesian Model," In: SC '12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, IEEE, USA, 2012.

[68] J. J. Prevost, K. Nagothu, B. Kelley and M. Jamshidi, "Prediction of Cloud Data Center Networks Loads Using Stochastic and Neural Models," In: 6th International Conference on System of Systems Engineering (SoSE), IEEE, USA, 2011.

[69] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski, "Cost- and Deadline-Constrained Provisioning for Scientific Workflow Ensembles in IaaS Clouds," In: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, pp. 1–11, 2012.

[70] P. Bailin, W. Yanping, L. Hanxi, Q. Jie, "Task Scheduling and Resource Allocation of Cloud Computing Based on QoS," In: Advanced Materials Research, Volume 915-916, pp. 1382-1385, 2014.

[71] K. Dubey, M. Kumar, S.C. Sharma, "Modified HEFT Algorithm for Task Scheduling in Cloud Environment," In: Procedia Computer Science, Volume 125, pp. 725-732, 2018.

[72] H. Arabnejad, J. G. Barbosa, "List Scheduling Algorithm for Heterogeneous Systems by an Optimistic Cost Table," In: IEEE Transactions on Parallel and Distributed Systems, Volume 25, Issue 3, 2014.

[73] J. Yu, R. Buyya, K. Ramamohanarao, "Workflow Scheduling Algorithms for Grid Computing," In: Metaheuristics for Scheduling in Distributed Computing Environments, pp. 173-214, Springer, 2008.

[74] L. Singh, Sarbjeet Singh, "A Survey of Workflow Scheduling Algorithms and Research Issues," In: International Journal of Computer Applications, Volume 74, Issue 15, July 2013.

[75] M. Choudhary, S. K. Peddoju, "A Dynamic Optimization Algorithm for Task Scheduling in Cloud Environment," In: Journal of Engineering Research and Applications (IJERA), Volume 2, Issue 3, pp. 2564-2568, 2012.

[76] M. R. Garey, D. S. Johnson, "Computers and Intractability; A Guide to the Theory of NP-Completeness," In: Book, 1979.

[77] S. Parsa, R. Entezari-Maleki, "RASA: A New Task Scheduling Algorithm in Grid Environment," In: World Applied Sciences Journal, 7 (Special Issue of Computer & IT), pp. 152-160, 2009.

[78] O. M. Elzeki, M. Z. Reshad, M. A. Elsoud, "Improved Max-Min Algorithm in Cloud Computing," In: International Journal of Computer Applications, Volume 50, Issue 12, 2012.

[79] Y. Hu, J. Wong, G. Iszlai, M. Litoiu, "Resource Provisioning for Cloud Computing," In: Conference of the Centre for Advanced Studies on Collaborative Research, CASCON, pp. 101-111, ACM, 2009.

[80] S. Pandey, L. Wu, S. M. Guru, R. Buyya, ”A Particle Swarm Optimization-based Heuristic for Scheduling Workflow Applications in Cloud Computing Environments,” In: 24th IEEE International Conference on Advanced Information Networking and Applications (AINA), IEEE, 2010.

[81] W. N. Chen and J. Zhang, “An Ant Colony Optimization Approach to a Grid Workflow Scheduling Problem with Various QoS Requirements,” In: IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews, Volume 39, Issue 1, pp. 29-43, 2009.

[82] E. K. Byun, Y. S. Kee, J. S. Kim, S. Maeng, “Cost Optimized Provisioning of Elastic Resources for Application Workflows,” In: Future Generation Computer Systems, Volume 27, Issue 8, pp. 1011–1026, 2011.

[83] M. A. Rodriguez, R. Buyya, ”Deadline Based Resource Provisioning and Scheduling Algorithm for Scientific Workflows on Clouds,” In: IEEE Transactions on Cloud Computing, Volume 2, Issue 2, 2014.

[84] L.C. Canon, E. Jeannot, R. Sakellariou and W. Zheng, ”Comparative Evaluation of the Robustness of DAG Scheduling Heuristics,” In: Grid Computing - Achievements and Prospects, edited by Sergei Gorlatch, Paraskevi Fragopoulou and Thierry Priol, pp. 73-84, Springer, 2008.

[85] E. Ilavarasan, P. Thambidurai, ”Low Complexity Performance Effective Task Scheduling Algorithm for Heterogeneous Computing Environments,” In: Journal of Computer Sciences, Volume 3, Issue 2, pp. 94-103, 2007.

[86] H. Topcuoglu, S. Hariri, Wu, W.Min-You, “Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing,” In: IEEE Transactions on Parallel and Distributed Systems, Volume 13, Issue 3, pp. 260–274, 2002.

[87] H. El-Rewini and T.G. Lewis, ”Scheduling Parallel Program Tasks onto Arbitrary Target Machines,” In: Journal of Parallel and Distributed Computing, Volume 9, Issue 2, pp. 138-153, 1990.

[88] G. C. Sih and E.A. Lee, ”A Compile-Time Scheduling Heuristic for Interconnection-Constrained Heterogeneous Processor Architecture,” In: IEEE Transactions on Parallel and Distributed Systems, Volume 4, Issue 2, pp. 175-187, 1993.

[89] M. A. Iverson, F. Ozguner and G. J. Follen, ”Parallelizing Existing Applications in a Distributed Heterogeneous Environment”, In: 4th Heterogeneous Computing Workshop (HCW 95), pp. 93-100, 1995.

[90] H. Oh and S. Ha, ”A Static Scheduling Heuristic for Heterogeneous Processors, ” In: Euro-Par. 96 Parallel Processing, Volume 1124 of Lecture Notes in Computer Science, pp. 573-577, 1996.

[91] A. Radulescu and A. J. C. van Gemund, ”Fast and Effective Task Scheduling in Heterogeneous Systems,” In: 9th Proceedings Heterogeneous Computing Workshop (HCW), pp. 229-238, 2000.

[92] H. Topcuoglu, S. Hariri and M.-Y. Wu, ”Task scheduling algorithms for heterogeneous processors,” In: 8th Proceedings Heterogeneous Computing Workshop (HCW), pp. 3-14, 1999.

[93] T. Hagras and J. Janecek, ”A Simple Scheduling Heuristic for Heterogeneous Computing Environments,” In: Proceedings Second International Symposium on Parallel and Distributed Computing, pp. 104-110, 2003.

[94] E. Ilavarasan, P. Thambidurai and R. Mahilmannan, ”High Performance Task Scheduling Algorithm for Heterogeneous Computing System,” In: Distributed and Parallel Computing, Volume 3719 of Lecture Notes in Computer Science, pp. 193-203, 2005.

[95] M.I. Daoud and N. Kharma, ”A High Performance Algorithm for Static Task Scheduling in Heterogeneous Distributed Computing Systems,” In: Journal of Parallel and Distributed Computing, Volume 68, Issue 4, pp. 399-409, 2008.

[96] L.F. Bittencourt, R. Sakellariou and E.R.M. Madeira, ”DAG Scheduling Using a Lookahead Variant of the Heterogeneous Earliest Finish Time Algorithm,” In: 18th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP’10), pp. 27-34, 2010.

[97] Y. Cui and Zh. Xiaoqing, ”Workflow Tasks Scheduling Optimization Based on Genetic Algorithm in Clouds,” In: 3rd IEEE International Conference on Cloud Computing and Big Data Analysis (ICCCBDA), IEEE, pp.6-10, 2018.

[98] Sh. Mittal and A. Katal, ”An Optimized Task Scheduling Algorithm in Cloud Computing,” In: 6th International Conference on Advanced Computing, IEEE, pp.197-202, 2016.

[99] Ch.-Y. Liu, Ch.-M. Zou, P. Wu, ”A Task Scheduling Algorithm based on Genetic Algorithm and Ant Colony Optimization in Cloud Computing,” In: 13th International Symposium on Distributed Computing and Applications to Business, Engineering and Science, IEEE, pp. 68-72, 2014.

[100] M. B. Gawali and S. K. Shinde, ”Task Scheduling And Resource Allocation in Cloud Computing using a Heuristic Approach,” In: Journal of Cloud Computing: Advances, Systems and Applications, Volume 7, Issue 4, Springer, 2018.

[101] A. Choudhary, M. C. Govil, G. Singh, L. K. Awasthi, E. S. Pilli and D. Kapil, ”A Critical Survey of Live Virtual Machine Migration Techniques,” In: Journal of Cloud Computing: Advances, Systems and Applications, 6(23), Springer, 2017.

[102] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, A. Warfield, ”Live Migration of Virtual Machines," In: Proceedings of the 2nd Conference on Symposium on Networked Systems Design & Implementation (NSDI’05), Volume 2, pp. 273–286, 2005.

[103] Anja Strunk, ”Costs of Virtual Machine Live Migration: A Survey,” In: IEEE Eighth World Congress on Services, pp. 323-329, 2012.

[104] M. Hines and K. Gopalan, “Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning,” In: Proceedings of ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE’09), Washington, 2009.

[105] S. Sharma and M. Chawla, ”A technical Review for efficient virtual machine migration,” In: International Conference on Cloud & Ubiquitous Computing & Emerging Technologies, IEEE, pp.20-25, 2013.

[106] S. Venkatesha, S. Sadhu, S. Kintali, ”Survey of Virtual Machine Migration Techniques,” In: University of California, Santa Barbara, 2014.

[107] H. Liu, C.-Z. Xu, H. Jin, J. Gong, X. Liao, ”Performance and Energy Modeling for Live Migration of Virtual Machines,” In: Proceedings of the 20th International Symposium on High Performance Distributed Computing, ACM, California, pp. 171–182, 2011.

[108] S. Akoush, R. Sohan, A. Rice, A.W. Moore, A. Hopper, ”Predicting the Performance of Virtual Machine Migration,” In: 18th Annual IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, IEEE, Miami Beach, pp. 37–46, 2010.

[109] R. Yadav, W. Zhang, K. Li, C. Liu, M. Shafiq, and N. K. Karn, ”An Adaptive Heuristic for Managing Energy Consumption and Overloaded Hosts in a Cloud Data Center,” In: Wireless Networks: The Journal of Mobile Communication, Computation and Information, pp. 1-15, Springer, 2018.

[110] S. Esfandiarpoor, A. Pahlavan, and M. Goudarzi, ”Structure-aware Online Virtual Machine Consolidation for Data Center Energy Improvement in Cloud Computing,” In: Computers & Electrical Engineering, Volume 42, pp. 74-89, 2015.

[111] P. Bryk, M. Malawski, G. Juve, E. Deelman, ”Storage-Aware Algorithms for Scheduling of Workflow Ensembles in Clouds,” Journal of Grid Computing, Volume 14, Issue 2, pp. 359-378, 2016.

[112] R. Yadav, W. Zhang, H. Chen, and T. Guo, ”Mums: Energy-Aware VM Selection Scheme for Cloud Data Center,” In: 28th International Workshop on Database and Expert Systems Applications (DEXA), IEEE, pp. 132–136, 2017.

[113] S. M. Moghaddam, S. F. Piraghaj, and M. O’Sullivan, ”Energy-efficient and SLA-aware Virtual Machine Selection Algorithm for Dynamic Resource Allocation in Cloud Data Centers,” In: 11th International Conference on Utility and Cloud Computing (UCC), IEEE, pp. 103-113, 2018.

[114] S. Y. Z. Fard, M. R. Ahmadi, and S. Adabi, ”A Dynamic VM Consolidation Technique for QoS and Energy Consumption in Cloud Environment,” In: The Journal of Supercomputing, Volume 73, Issue 10, pp. 4347–4368, 2017.

[115] H. Wang, H. Tianfield, ”Energy-aware Dynamic Virtual Machine Consolidation for Cloud Data Centers,” In: IEEE Access, Volume 6, IEEE, pp. 15259–15273, 2018.

[116] X.-F. Liu, Zh.-H. Zhan, J. D. Deng, Y. Li, T. Gu, and J. Zhang, ”An Energy Efficient Ant Colony System for Virtual Machine Placement in Cloud Computing,” In: IEEE Transactions on Evolutionary Computation, Volume 22, Issue 1, IEEE, pp. 1-15, 2018.

[117] M. A. Khan, A. Paplinski, A. M. Khan M. Murshed, and R. Buyya, ”Dynamic Virtual Machine Consolidation Algorithms for Energy-Efficient Cloud Resource Management: A Review,” In:

Sustainable Cloud and Energy Services, Springer, pp. 135-165, 2018.

[118] A. Varasteh and M. Goudarzi, ”Server Consolidation Techniques in Virtualized Data Centers: A Survey,” In: Systems Journal, Vol. 11, Issue 2, IEEE, pp. 772-783, 2017.

[119] M. Dabbagh, B. Hamdaoui, M. Guizani, A. Rayes, ”Toward Energy-Efficient Cloud Computing: Prediction, Consolidation, and Overcommitment,” In: IEEE Network, Volume 29, Issue 2, pp. 56–61, 2015.

[120] A. Mazrekaj, D. Minarolli, and B. Freisleben, ”Dynamic Resource Allocation in Cloud Environments,” In: Information & Communication Technologies at Doctoral Student Conference (ICT@DSC), Thessaloniki, Greece, 9-11 May 2018.

[121] A. Mazrekaj, A. Sheholli, D. Minarolli, and Bernd Freisleben, “The Experiential Heterogeneous Earliest Finish Time Algorithm for Task Scheduling in Clouds,” In: 9th International Conference on Cloud Computing and Services Science (CLOSER 2019), Heraklion, Crete, Greece, 2-4 May 2019.

[122] W. Hu, A. Hicks, L. Zhang, E.M. Dow, V. Soni, H. Jiang, R. Bull, J.N. Matthews, ”A Quantitative Study of Virtual Machine Live Migration,” In: Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference on - CAC ’13, Florida, USA.

[123] D.W. Scott, ”Multivariate Density Estimation: Theory, Practice, and Visualization,” In: Wiley Series in Probability and Mathematical Statistics: Applied Probability and Statistics Section, Wiley, Brisbane, New York, 1992.

[124] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I.H. Witten, ”The Weka Data Mining Software: An update, ” In: ACM SIGKDD Explorations Newsletter, Volume 11, No.1, 2009.

[125] Commons Math: The Apache Commons Mathematics Library, http://commons.apache.org/ [Accessed 2017].

[126] A. Mazrekaj, Sh. Nuza, M. Zatriqi, V. Alimehaj. “An Overview of Virtual Machine Live Migration Techniques,” In: International Journal of Electrical and Computer Engineering (IJECE), Vol. 9, No. 5, 2019.

[127] Y. Samadi, M. Zbakh, C. Tadonki, “E-HEFT: Enhancement Heterogeneous Earliest Finish Time algorithm for Task Scheduling based on Load Balancing in Cloud Computing,” In: International Conference on High Performance Computing & Simulation (HPCS), pp. 601-609, Orleans, France, 2018.

[128] F. Farahnakian, T. Pahikkala, P. Liljeberg, J. Plosila, H. Tenhunen, “Utilization Prediction Aware VM Consolidation Approach for Green Cloud Computing,” In: 8th International Conference on Cloud Computing, IEEE, pp. 381-388, NY, USA, 2015.

[129] S. Ismaeel, R. Karim, A. Miri, “Proactive Dynamic Virtual-Machine Consolidation for Energy Conservation in Cloud Data Centres,” In: Journal of Cloud Computing: Advances, Systems and Applications 7(10), Springer, 2018.

[130] S. Ismaeel, A. Miri, “Using ELM Techniques to Predict Data Centre VM Requests,” In:

Proceedings of the 2nd IEEE International Conference on Cyber Security and Cloud Computing, IEEE, pp. 80–86, New York, 2015.

[131] R. Karim, S. Ismaeel, A. Miri, “Energy-efficient Resource Allocation for Cloud Data Centres using a Multiway Data Analysis Technique,” In: International Conference on Human-Computer Interaction. Theory, Design, Development and Practice, Volume 9731, Springer-Verlag, pp. 577–585, New York, 2016.

[132] R. Karim, C. Ding, A. Miri, “End-to-end Performance Prediction for Selecting Cloud Services Solutions,” In: IEEE Symposium on Service-Oriented System Engineering, IEEE, pp. 69–77, San Francisco, USA, 2015.

[133] G. F. Shidik, Azhari, K. Mustofa, “Evaluation of Selection Policy with Various Virtual Machine Instances in Dynamic VM Consolidation for Energy Efficient at Cloud Data Center,” In: Journal of Networks, Volume 10, No. 7, pp. 397–406, 2015.

[134] A. Song, W. Fan, W. Wang, J. Luo, Y. Mo, “Multi-objective Virtual Machine Selection for Migrating in Virtualized Data Centers,“ In: Joint International Conference on Pervasive Computing and the Networked World. Pervasive Computing and the Networked World. Springer, pp. 426–438, 2013.

[135] S. S. Masoumzadeh, H. Hlavacs, “Integrating VM Selection Criteria in Distributed Dynamic VM Consolidation using Fuzzy Q-Learning,“ In: Proceedings of the 9th International Conference on Network and Service Management (CNSM), IEEE, pp. 332–338, Zurich, Switzerland, 2013.

[136] M. Monil, R. Rahman, “Fuzzy Logic based Energy Aware VM Consolidation,” In: International Conference on Internet and Distributed Computing Systems (IDCS). Lecture Notes in Computer Science, Volume 9258. Springer, pp. 31–38, 2015.

[137] J. T. Tsai, J. C. Fang, J.H. Chou, “Optimized Task Scheduling and Resource Allocation on Cloud Computing Environment using Improved Differential Evolution Algorithm, “In: Computer and Operation Research, Volume 40, No. 12, pp. 3045–3055, 2013.

[138] S. T. Maguluri, R. Srikant, “Scheduling Jobs with Unknown Duration in Clouds,” In: IEEE/ACM Transactions on Networking, Volume 22, No. 6, pp. 1938–1951, 2014.

[139] Ch. Cheng, J. Li, Y. Wang, “An Energy-saving Task Scheduling Strategy based on Vacation Queuing Theory in Cloud Computing,” In: Tsinghua Science and Technology, Volume 20, No. 1, pp. 28–39, 2015.

[140] W. Lin, Ch. Liang, J. Z. Wang, R. Buyya, “Bandwidth-aware Divisible Task Scheduling for Cloud Computing,” In: Software: Practice and Experience, Volume 44, No.2, pp. 163–174, 2014.

[141] X. Liu, Y. Zha, Q. Yin, Y. Peng, L. Qin, “Scheduling Parallel Jobs with Tentative Runs and Consolidation in the Cloud,” In: Journal of Systems and Software, Volume 104, No. C, pp.141–151, 2015.

[142] A. E. Keshk, A. B. El-Sisi, M.A. Tawfeek, “Cloud Task Scheduling for Load Balancing Based on Intelligent Strategy,” In: I. J. Intelligent Systems and Applications, Volume 6, No. 6, 2014.

[143] G. Shamsollah, M. Othman, W. J. Leong, M. R. A. Bakar, “Multi-Criteria Based Algorithm for Scheduling Divisible Load,” In: Proceedings of the first international conference on advanced data and information engineering (DaEng-2013), Springer, pp. 547–554, 2014.

[144] H. Goudarzi, M. Ghasemazar, M. Pedram M, “SLA-Based Optimization of Power and Migration Cost in Cloud Computing,” In: Proceedings of 12th IEEE/ ACM International Symposium on Cluster, Cloud and Grid Computing, IEEE, pp. 172-179, Ottawa, Canada, 2012.

[145] Sh. Ghanbari, M. Othman, M. R. A. Bakar, W. J. L Leong, “Multi-Objective Method for Divisible Load Scheduling in Multi-Level Tree Network,” In: Future Generation Computer Systems, Volume 54, pp. 132–143, 2016.

[146] B. Radojevic, M. Zagar, “Analysis of Issues with Load Balancing Algorithms in Hosted (Cloud) Environments,” In: Proceedings of the 34th International Convention MIPRO, pp. 416–420, 2011.

[147] Zh. Xiaomin, L. T. Yang, H. Chen, J. Wang, S. Yin, X. Liu, “Real-Time Tasks-Oriented Energy-Aware Scheduling in Virtualized Clouds,“ In: IEEE Transactions on Cloud Computing, Volume 2, Issue 2, IEEE, pp. 168–180, 2014.