# VIRTUAL MACHINE CONSOLIDATION IN CLOUD DATA CENTRES USING A PARAMETER-BASED PLACEMENT STRATEGY

A THESIS SUBMITTED TO THE UNIVERSITY OF MANCHESTER
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
IN THE FACULTY OF SCIENCE AND ENGINEERING

2019

By
Abdelkhalik Mosa
Department of Computer Science

# Contents

# List of Tables

# List of Figures

# Abstract


VIRTUAL MACHINE CONSOLIDATION IN CLOUD DATA CENTRES USING A PARAMETER-BASED PLACEMENT STRATEGY

Abdelkhalik Mosa

A thesis submitted to the University of Manchester for the degree of Doctor of Philosophy, 2019

Cloud computing enables cloud providers to offer computing infrastructure as a service in the form of virtual machines (VMs). VM placement is a vital component of any cloud management platform (*e.g.* OpenStack). VM placement is the process of mapping VMs to physical machines (PMs) efficiently according to the cloud provider's objectives and placement constraints. So far, any VM placement solution adopts either a reservation-based or demand-based VM placement strategies. Reservation-based VM placement allocates VMs to PMs according to the reserved VM size regardless of the actual workload. If a VM is making use of only a fraction of its reservation, then this leads to PM underutilization, which wastes energy and results in more costs. In contrast, demand-based VM placement consolidates VMs based on the actual workloads demand which may lead to better utilization. However, it may incur more service level agreement violations (SLAVs) resulting from overloaded PMs and/or VM migrations among PMs due to workload fluctuations. This thesis aims to introduce a novel VM placement strategy to control the tradeoff between PM utilization and SLAVs that will allow cloud providers to explore the whole space of VM placement options that range from demand-based to reservation-based, with the help of a single parameter. The thesis first presents our strategy called parameter-based VM placement using a static parameter. Then it introduces various algorithms that adjust this parameter continuously at run-time in a way that a provider can maintain the number of SLAVs below a certain (predetermined) threshold while using the smallest possible number of PMs. These algorithms fine-tune the parameter both at the cloud data center level and at the VM level using reactive and hybrid (reactive and proactive) approaches. An empirical evaluation using CloudSim confirms that the proposed parameter-based VM placement solution offers more flexibility in choosing between different tradeoffs.

# Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

# Copyright

# Acknowledgements

All thanks are to Allah for all the great favours and blessings that He has bestowed upon me, and for giving me the strength, knowledge, and ability to complete my Ph.D. journey satisfactorily. Next, I would like to express my sincere gratitude to my supervisors, Professor *Rizos Sakellariou* and Professor *John Gurd*, for their invaluable guidance, continuous support, kindness, and insightful advice on my research. I am also grateful to my Ph.D. examiners (Professor *Ashiq Anjum* and Dr *Antoniu Pop*) for their valuable comments and suggestions on improving this thesis.

To my late father, you always believed in me to be successful, you are gone, but your belief in me has made this journey possible. To my adorable mother, the most precious person in life, thanks for your love, care, and endless support. I can not forget all the sacrifices you and my late father have made on my behalf. To my darling wife *Walaa*, thanks for being incredibly supportive, so understanding, and for putting up with me through the toughest moments. To my boys, *Muhammad* and *Yasser*, I love you to the moon and back; you are the pride and joy of my life. To my amazing brothers, sisters, nephews, and nieces, this thesis would not have been possible without your warm love and endless support. I consider myself nothing without you.

I would always remember my fellow mates at the laboratory of High-Performance Parallel and Distributed Software Systems, including Dr. Jorge Buenabad-Chavez, Dr. Admilson Ribeiro, Ilia Pietri, Mingze Ma, Chong Ke, and Chuan Jin. Thanks for the passionate atmosphere, useful discussions, and the fun-time we spent together. Eventually, I acknowledge my lovely country, Egypt, and Suez Canal University, for giving me the opportunity to pursue my MSc and Ph.D. studies in the UK.

Abdelkhalik Mosa
Manchester, UK
2019

# List of Acronyms

| | |
|---|---|
| **ACF** | Autocorrelation Function |
| **ACO** | Ant Colony Optimization |
| **ACS** | Ant Colony System |
| **ANN** | Artificial Neural Network |
| **BFD** | Best Fit Decreasing |
| **CDC** | Cloud Data Centre |
| **DL** | Deep Learning |
| **DT** | Decision Tree |
| **DVFS** | Dynamic Voltage and Frequency Scaling |
| **EC2** | Elastic Compute Cloud |
| **FF** | First Fit |
| **FFD** | First Fit Decreasing |
| **GA** | Genetic Algorithm |
| **IaaS** | Infrastructure as a Service |
| **ILP** | Integer Linear Programming |
| **IQR** | Interquartile Range |
| **KVM** | Kernel-based Virtual Machine |
| **LRR** | Local Regression Robust |
| **LSTM** | Long Short-Term Memory |
| **MAE** | Mean Absolute Error |
| **MC** | Maximum Correlation |
| **ML** | Machine Learning |
| **MMT** | Minimum Migration Time |
| **MSE** | Mean Squared Error |
| **NN** | Neural Network |
| **OS** | Operating System |
| **PACF** | Partial Autocorrelation Function |

| | |
|---|---|
| **PaaS** | Platform as a Service |
| **PM** | Physical Machine |
| **PABFD** | Power-Aware Best Fit Decreasing |
| **PUE** | Power Usage Effectiveness |
| **QoS** | Quality of Service |
| **RF** | Random Forest |
| **ROI** | Return on Investment |
| **RNN** | Recurrent Neural Network |
| **RS** | Random Selection |
| **SaaS** | Software as a Service |
| **SLA** | Service Level Agreement |
| **SLAVs** | Service Level Agreement Violations |
| **SOA** | Service Oriented Architecture |
| **VDC** | Virtual Data Centre |
| **VM** | Virtual Machine |
| **VMM** | Virtual Machine Manager |
| **WFD** | Worst Fit Decreasing |
| **XaaS** | everything as a Service |

# Chapter 1

# Introduction

## 1.1 Introduction to Cloud Computing

Cloud computing is a computing paradigm wherein computing resources are offered as elastic on-demand services over a network [MG10]. According to the NIST [MG10], there are five essential characteristics of the cloud model, namely, *on-demand self-service*, *broad network access*, *resource pooling*, *rapid elasticity* and *measured service*. The on-demand self-service means that the cloud computing services are automatically available upon request without requiring human interaction with the cloud provider. Broad network access means that the cloud data centre (CDC) provides all cloud services through a network and hence clients can access such services using mobile devices or personal computers (PCs). The resource pooling characteristic indicates that the computing resources are pooled in the CDC for serving multiple users based on the multi-tenancy model. Multi-tenancy is a software architecture in which multiple customers (tenants) can use a single instance of the application as if each of them has a separate instance. Rapid elasticity indicates that computing services can be easily provisioned, released and scaled up and down based on the user request. Finally, a measured service means that the cloud services are monitored, measured, and the user pays based on a pay-per-use model. The inherent benefits and the elasticity of the cloud lure cloud customers to use cloud services whenever convenient.

The three principal actors in public cloud environments are *cloud providers*, *cloud customers* and *end users* [JS14]. Cloud providers offer computing resources as cloud services through their cloud infrastructures. They offer these services as metered ones based on a pay-per-use model. Cloud customers buy cloud services from cloud providers to host their applications and run the required services. By the end, cloud

customers offer services hosted by cloud providers to the end users. The end users are the consumers of cloud services that are offered either by the cloud customers or directly by the cloud providers. As an illustration, Airbnb[1] (an example of a cloud customer) uses the cloud infrastructure offered by Amazon Elastic Compute Cloud (Amazon EC2)[2] (an example of a cloud provider) to host their applications that are finally used by Airbnb users (an example of end users). Those various actors in cloud environments usually have different objectives. For example, cloud providers normally aim to maximize their profit while satisfying the cloud customers' needs by meeting service level agreement (SLA). The service level agreement (SLA) is an agreement between the service provider and the customer/user that specifies the quality and scope of the service besides the responsibilities of the service provider [AB12]. The SLA defines the quality of service (QoS) requirements (*e.g.* performance, availability, and reliability of applications or infrastructure) in the form of a contractual document between the service provider and customers [LS14]. On the other hand, the cloud customers' objectives include increasing their return on investment (ROI) and satisfying end users by achieving the QoS requirements. This thesis considers the cloud provider's objectives, so it pays more attention to the role of cloud providers and customers and overlooks the end users' role as it is not essential to the context of this research.

**Cloud deployment models:** Figure 1.1 presents the four common cloud deployment models namely, *public*, *private*, *community* and *hybrid* [BKNT11]. In public cloud environments, cloud providers offer the cloud infrastructure as elastic, on-demand services over the Internet to the general public who can use it on a pay-per-use basis. In the public cloud, the cloud customers and the cloud provider usually belong to different organizational entities. Examples of public cloud providers include Amazon, Google, Microsoft, and Rackspace. Public clouds enable cloud customers to offload their data and applications on demand without worrying about the initial investment and maintenance costs in addition to shortening the start-up time for their services. On the contrary, some organizational entities may prefer building their private clouds due to security considerations in public clouds including confidentiality, integrity, and availability. A private cloud, also called *internal cloud* or *intraCloud*, is only accessible to authorized users within the organization and is exclusively owned and managed by a single organization. Private clouds permit organizations to have total control of

[1] https://www.airbnb.com/ [Accessed: 29-January-2018]
[2] https://aws.amazon.com/ec2 [Accessed: 29-January-2018]

their cloud infrastructure. In private clouds, the cloud provider and the users belong to the same organizational entity. If multiple organizations have shared concerns, then they can build a shared cloud infrastructure which is known as a community cloud. The management of the community cloud can be by either some/all members of the community cloud or even a third party. The main purpose of the community cloud is to meet the shared requirements of the participating entities. Finally, the hybrid cloud deployment model is a combination of two or more of the previously mentioned cloud deployment models (public, private, or community). For example, in Figure 1.1, *Hybrid cloud (1)* is composed of a private and a public cloud while *Hybrid cloud (2)* is composed of a community and a public cloud. This thesis is only concerned with the public or private implementations of the cloud.

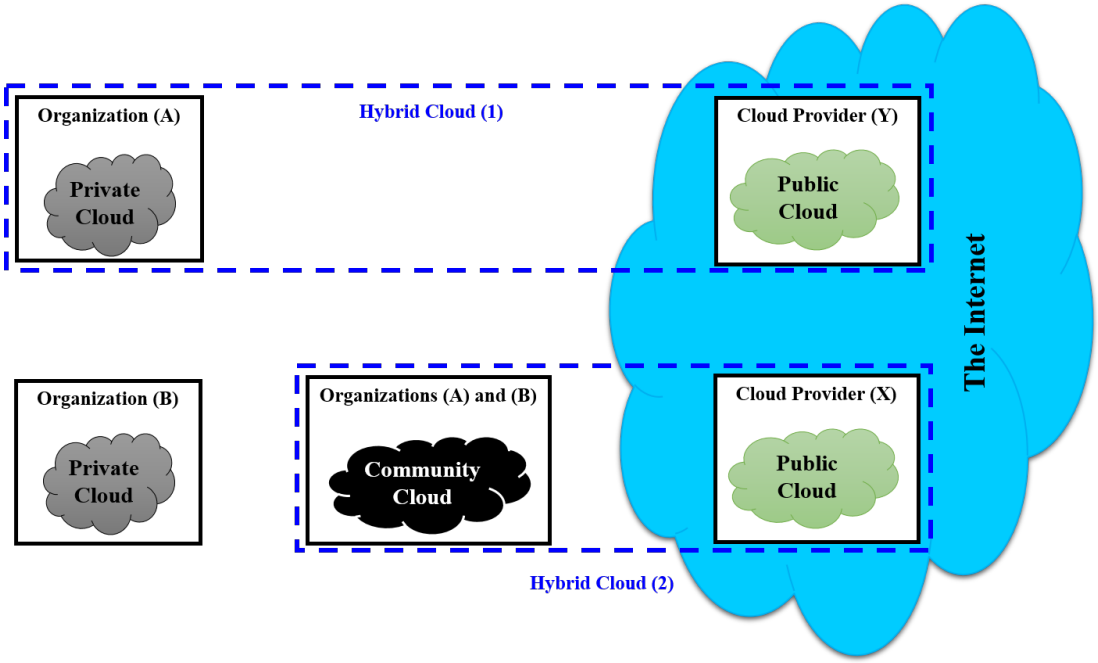



Figure 1.1: Cloud deployment models

**Cloud service models:** Creating a private cloud involves building the whole cloud infrastructure starting from the network design, purchase of the physical machines/servers (PMs) to the installation of the operating system (OS) and the applications and finally monitoring and managing the cloud. Figure 1.2(a) presents the different management components for creating a private (on-premise) cloud. Private cloud owners use cloud

management tools such as OpenStack[3], OpenNebula[4] or Eucalyptus[5] to manage their own implementations of their private clouds. On the contrary, public cloud providers can offer their cloud infrastructure on a pay-per-use basis through different service models. Cloud service models follow an everything-as-a-service (XaaS) model where $X$ denotes the type of the service [BKNT11]. The three fundamental service models are *infrastructure as a service (IaaS)*, *platform as a service (PaaS)* and *software as a service (SaaS)* [BKNT11]. Figure 1.2 shows the responsibilities of the cloud provider compared to the responsibilities of the cloud customer on each service model as opposed to the private cloud scenario.

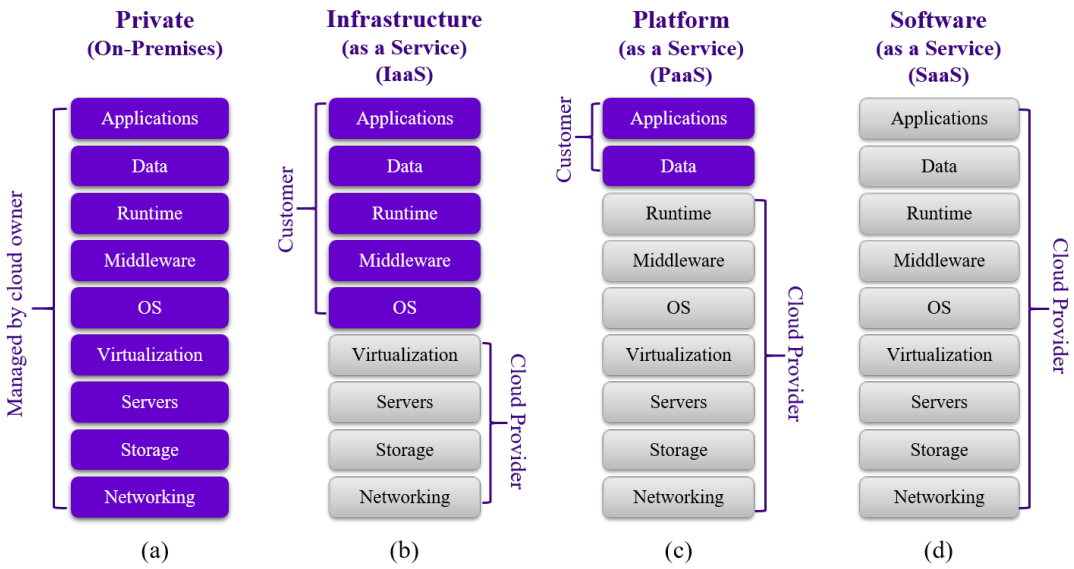



Figure 1.2: Cloud service models

Public IaaS providers provision the cloud infrastructure (including networking, storage, servers, and the virtualization) in the form of virtual machines (VMs) as shown in Figure 1.2(b). An IaaS cloud customer can install the desired OS, programming platform and applications. Common examples of IaaS providers include Amazon Elastic Compute Cloud (Amazon EC2)[6], Microsoft Azure[7] and Rackspace[8]. The PaaS providers offer services for developers such as programming environments, database management systems (DBMSs), execution environments and development tools. The

[3]https://www.openstack.org/ [Accessed: 22-Jan-2018]
[4]https://opennebula.org/ [Accessed: 22-Jan-2018]
[5]https://github.com/eucalyptus/eucalyptus/wiki [Accessed: 22-Jan-2018]
[6]https://aws.amazon.com/ec2/ [Accessed: 22-Jan-2018]
[7]https://azure.microsoft.com/ [Accessed: 22-Jan-2018]
[8]http://www.rackspace.com/ [Accessed: 22-Jan-2018]

PaaS customers (*e.g.*software developers) only care about building their applications without worrying about any hardware or software requirements which can help rising productivity rapidly. Examples of PaaS providers include Google App Engine[9], AWS Elastic Beanstalk[10] and Zoho Creator[11]. Finally, public cloud providers can offer software applications as services through the SaaS service model. The customer of the SaaS needs only to have Internet access to use cloud applications. The SaaS customer neither cares about the cloud infrastructure nor the platform that hosts the applications as shown in Figure 1.2(d). Customers use the SaaS model even before the advent of the cloud hype. As an illustration, one of the earliest SaaS services used by the Internet users was the email service where electronic mail providers offer sending and receiving emails as a service over the Internet. Email service is a form of SaaS as the email user can use the email service on-demand over the Internet without caring about the infrastructure. Some of the well-known SaaS applications include Google Docs, Sheets, and Slides[12] and Microsoft Office 365[13].

The IaaS service model (in public or private clouds) is the primary concern of this thesis; where the cloud provider offers the computing infrastructure in the form of VMs, and cloud customers can request VMs of specific sizes based on the available instance types.

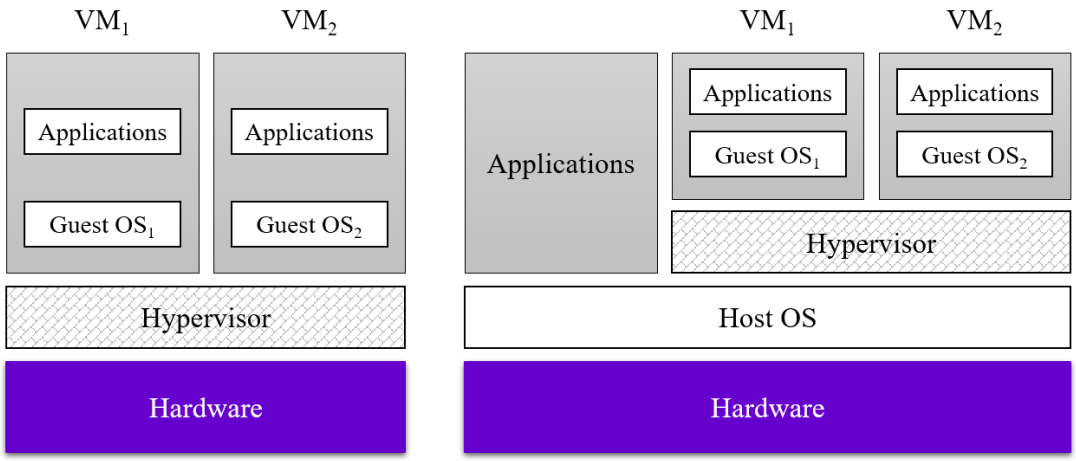
## 1.2 Virtualization Technology

Cloud computing is a normal evolution of the existing technologies. Common examples of technologies that enabled the existence of cloud computing include *service oriented architecture* (SOA) [CHT03], *utility computing* [Rap04], *Web 2.0* [Ale06], *multi-tenancy* [BZP+10], *automation* [KC03] and *virtualization* [BDF+03]. Virtualization technology is one of the basic building blocks that underpin the cloud computing infrastructure by creating multiple virtual instances of hardware, storage, and networking resources.

Virtualization is one of the principal technologies that enabled the existence of the cloud computing. In computing, virtualization means creating a virtual version of a computing resource; this computing resource might be server hardware, a storage or

[9]https://cloud.google.com/appengine/ [Accessed: 22-Jan-2018]
[10]https://aws.amazon.com/elasticbeanstalk/ [Accessed: 22-Jan-2018]
[11]https://www.zoho.com/creator/ [Accessed: 22-Jan-2018]
[12]https://docs.google.com [Accessed: 22-Jan-2018]
[13]http://products.office.com/en-gb/business/Office [Accessed: 22-Jan-2018]

networking device, an OS or an application. Common types of virtualization include *hardware or platform* (using either *full virtualization* or *paravirtualization*), *OS*, *storage*, *network*, *desktop* and *application* virtualization [BKNT11]. This section discusses only the hardware full virtualization due to its relevance to the context.




(a) Type-1 Hypervisor (bare-metal) (a) Type-2 Hypervisor (hosted)

Figure 1.3: Type-1 versus Type-2 Hypervisors, adapted from [BKNT11]

Hardware virtualization (*a.k.a* platform) enables creating multiple virtual machines (VMs) on a single physical machine (PM). A virtual machine refers to the software implementation of a virtual and isolated computer that seems to be a physical machine (end user usually does not know whether it is physical or virtual). The hardware virtualization makes use of a new abstraction layer called *virtual machine monitor (VMM)* or *hypervisor* to make calls from the VM to the PM [Dan09]. The VMM or the hypervisor is low-level software that enables creating, managing and configuring VMs by allocating computing resources to the guest VMs in real-time. Furthermore, the VMM facilitates both online and offline migration of the VMs. Full hardware virtualization emulates the entire VM by using one of the two types of hypervisors namely, *type-1* (*a.k.a bare-metal or native*) or *type-2* (*a.k.a hosted*) hypervisors. Type-1 hypervisor runs directly on top of the bare metal hardware without the need for any hosting OSs; therefore it is an OS independent, as shown in Figure 1.3(a). Some examples of type-1 hypervisors include VMware ESXi[14], Citrix XenServer[15] and Microsoft Hyper-V[16].

[14]http://www.vmware.com/products/esxi-and-esx/overview [Accessed: 22-Jan-2018]
[15]https://www.citrix.com/products/xenserver/overview.html [Accessed: 22-Jan-2018]
[16]https://www.microsoft.com/en-gb/cloud-platform/server-virtualization [Accessed: 22-Jan-2018]

On the contrary, type-2 hypervisors run on top of a traditional host OS like any other software as shown in Figure 1.3(b); therefore, type-2 hypervisors are OS dependent. Examples of type-2 hypervisors include kernel-based virtual machine[17] (KVM), VirtualBox[18] and VMware Workstation[19]. By and large, type-1 hypervisors are more relevant to large-scale data centres as they provide higher performance and availability compared to type-2 hypervisors.

Hardware virtualization simplifies the process of creating, terminating, and managing VMs. The virtualization technology played a vital role in addressing the issue of overprovisioning in traditional (non-virtualized) data centres. The following subsection illustrates how virtualization technology managed to solve key issues in traditional data centres and facilitated the emergence of cloud data centres.

### 1.2.1 Data Centres and Virtualization

A data centre is a large group of networked computing and storage resources that are used by organizations to deliver, process, and store shared applications and data. Traditional data centres used to suffer from an *overprovisioning* problem before the birth of the virtualization technology. Overprovisioning means allocating more resources than the required to meet peak workload demands. Common reasons for overprovisioning in traditional data centres include: (1) guaranteeing reliable performance during peak times, and (2) using dedicated servers for each type of application even though those servers are rarely used. Resource overprovisioning made traditional data centres experience *sprawl* situations. *Server sprawl*[20] occurs when the data centre has many underutilized dedicated servers that take up too much space, resources and energy compared to the actual workload. Even though overprovisioning can help to meet SLA requirements, it wastes resources and incurs additional costs.

Using server virtualization, the operators of data centres can consolidate multiple VMs into a single PM and thus cuts down the unjustified cost of server sprawl. Virtualization achieves efficient resource utilization as well as energy savings through *server consolidation* [BKNT11] by gaining from the underutilized servers [LKN+09]. Server consolidation is an approach to efficiently utilize the data centre resources by either replacing legacy dedicated servers (to save space) or combining multiple virtualized

[17] https://www.linux-kvm.org/page/Main_Page [Accessed: 22-Jan-2018]
[18] https://www.virtualbox.org/ [Accessed: 22-Jan-2018]
[19] http://www.vmware.com/uk/products/workstation [Accessed: 22-Jan-2018]
[20] http://searchdatacenter.techtarget.com/definition/server-sprawl

servers, which minimizes the total number of running PMs. Server consolidation is one example of the *capacity planning or capacity management* problems and is also known as *workload concentration* [WBS16]. Cloud data centres (CDCs) use server consolidation to mitigate the server sprawl problem. A CDC is a virtualized data centre that is self-managed by a cloud management software. Consequently, virtualization technology helped in mitigating the resource overprovisioning problem and server underutilization in traditional data centres by consolidating multiple VMs into a single PM. However, overprovisioning still exists in cloud data centres due to the underutilization of VMs.

Server consolidation using virtualization can be either *manual* or *automated*. Under manual consolidation, the CDC operator has to manually decide which VMs should be consolidated on a specific PM. On the contrary, automated VM consolidation is carried out by a *cloud management software* . A cloud management software controls PMs, storage, and networking resources throughout the CDC. Existing cloud management tools (*a.k.a.* cloud operating systems (COSs)) such as *OpenStack* with the embedded VMMs such as *KVM or VMware ESXi* manage the infrastructure of the CDC. Automated VM consolidation includes the assignment of VMs to the PMs, a problem generally known as VM placement. The following section illustrates the effect of servers' underutilization on energy waste in data centres.

## 1.3 Server Utilization and Energy Efficiency

Cloud data centres are growing to be the backbone of the Internet and the back-end infrastructure of the new generation of end-users' thin clients [AE15]. The global energy consumption of data centres is growing and expected to grow. By 2030, the global data centres are expected to consume a total of 2967 TWh [AE15]. However, data centres' operators can save over 50% of the energy consumption by adopting the best practices to efficiently use the data centres' resources [AE15]. According to the Uptime Institute survey [Sta15], power usage efficiency (PUE), operating expense, kW per rack, and server utilization are the top four most important metrics by data centre operators and IT practitioners. Additionally, power efficiency and hardware consolidation (for reducing costs and utilizing data centres efficiently) are two of the top ten data centre management priorities [HU13]. Low server utilization is one of the leading causes of energy waste [Nrd15], and it is one of the most common topics for data centre efficiency after PUE [WD14].

Server utilization represents the amount of used resources on the server relative to the maximum server capacity. For example, a server's CPU utilization is the CPU load divided by the maximum CPU capacity. Low server utilization (a result of overcapacity or overprovisioning) is one of the critical concerns for the data centre management as server resources unused most of the time. The server underutilization wastes energy as an entirely idle PM can consume around 70% of its peak power [FWB07] due to the narrow dynamic power range of the servers. Additionally, more than a fifth of IT devices do nothing rather than warming the planet [Nrd15]. Furthermore, according to the findings by the Uptime Institute [KT15, KFK08], 30% of the US data centres are comatose. A comatose server is the one that is running and using electricity but does not deliver useful services. The main reason for the low server utilization is the overprovisioning of the computing resources to meet the demands at peak periods. Data centres are mostly overprovisioned, and the PMs are only 10-50% utilized most of the time [BH07]. Other examples for low server utilization include corporate servers that are idle out of the work hours and during the night. An additional example, developer workstations are idle most of the time; only used during code compilation, testing, etc. This low server utilization means that extra PMs are wasting energy without a real added value.

The inefficiency of server utilization and the related energy consumption requires more efficient solutions that reduce the underutilization per PMs. Accomplishing efficiency with big cloud providers (such as Amazon or Google) does not solve the problem as the primary cloud providers only account for about 5% while the majority are small and medium-size governmental and corporate data centres [WD14]. Energy efficient VM placement solutions that achieve better utilization ratios can cut energy waste from server underutilization which results in a better environmental (by increasing the resulting $CO_2$ emissions) and financial impact.

## 1.4 Virtual Machine Placement

For an efficient consolidation of the VMs in CDCs, a cloud provider needs to optimize the placement of VMs on the data centre's PMs in a way that increases the profit from hosting the VMs. VM placement is the process of assigning VMs to PMs to achieve organizational objectives. IBM launched the concept of *autonomic computing* to refer to self-managing systems that automatically adapt to the changing requirements of the system and manage upcoming issues with minimum human intervention [SB03]. A

VM placement solution is an example of an *autonomic* (self-managing) system as it can manage itself by adapting to changes without requiring human intervention. Developing such an autonomic system requires adopting one of the common policies: *action (rule-based)*, *goal* or *utility function* [KD07]. A VM placement solution considers either the initial mapping of the VMs to PMs (*a.k.a.* initial or static VM placement) or the dynamic reallocation of the VMs using live migration (*a.k.a.* dynamic VM placement) [LL+12].

### 1.4.1 Initial (Static) VM Placement

Cloud customers send VM placement requests to cloud providers; this VM placement request specifies the size of the required resources per the VM (*e.g.* CPU, memory, and bandwidth). The cloud management software uses the initial VM placement controller to either accept or reject the VM placement requests and then assigns each accepted VM to an appropriate PM. The fundamental question is how to place the VMs on the least possible number of PMs while satisfying the SLA requirements. This question (of initial VM placement problem) is an example of a high-dimensional bin packing problem, which is one of the well-known NP-hard problems. Cloud management software that only uses initial or static VM placement does not make use of the live migration of the VMs. Consequently, the initial VM placement determines the PM on which a VM will be launched, and this VM-to-PM assignment is not changed until the deallocation of the VM regardless of any changes in the cloud environment. Commonly, the initial VM placement uses a *reservation-based* VM placement strategy, which assigns VMs to PMs according to a (statically) reserved VM size regardless of the actual workload utilization.

Employing only an initial VM placement solution might not utilize the PMs efficiently due to the static nature of the initial VM placement as well as the adoption of a reservation-based VM placement strategy. As an illustration, the initial VM placement does not reallocate VMs due to the deallocation of some other VMs which might lead to underutilization per PMs. Moreover, the initial VM placement does not react to workload changes per VMs. For example, if (at some point in time) a VM is making use of only a fraction of its reservation this also leads to PM underutilization. The underutilization per PMs leads to inefficient utilization of the PMs which wastes energy and, on a grand scale, it may result in substantial financial and environmental costs. Therefore, using dynamic VM placement solutions that can adapt to changes in CDC by reallocating the VMs could result in a better utilization.

### 1.4.2 Dynamic VM Placement

The dynamic VM placement involves creating a self-managing system that adapts to the changes in the cloud environment by migrating VMs in a way that helps in achieving the cloud provider's goals. As previously described (in Section 1.4.1), the changes in the cloud environment might include allocations of new VMs, deallocation of existing VMs, the numerous variations of the applications' workload on the hosted VMs and hardware failures. Therefore, using only the initial VM placement is not sufficient. The dynamic consolidation of the VMs is crucial for utilizing the data centre's resources efficiently by either reacting or proacting to changes in the cloud data centre. The dynamic allocation runs over short time periods either periodically or based on specific triggers. The dynamic VM placement solution automatically reallocates VMs among the suitable PMs whenever needed using the *live migration* utility. Occasionally, the dynamic placement controller may migrate VMs from underutilized and overutilized PMs. As an illustration, the dynamic VM placement controller might migrate all VMs from underutilized PMs and turn these PMs off or switch them into one of the power saving modes to minimize the number of running PMs for more efficient utilization of the PMs and hence more energy savings.

The question that arises is: on which basis does the dynamic VM placement controller reallocate the VMs? In general, the dynamic VM placement controller estimates the VM reallocation decision based on the adopted *VM placement strategy*. So far, the two common VM placement strategies are *reservation-based* and *demand-based* [WTAPB15].

In reservation-based VM placement, VMs are assigned to PMs based on the requested (by the user) VM size (CPU, memory and bandwidth capacities) regardless of the actual utilization. Dynamic reservation-based placement may react to VM deallocation or new VM allocations by migrating VMs between PMs. Nevertheless, it does not respond to fluctuations in the workload, which means that VM resources may be reserved regardless of actual use.

In contrast, demand-based placement assigns VMs to PMs based on the *current workload demand* rather than the *reserved resources*. The demand-based placement requires a knowledge of the resource demand that can be identified using *reactive*, *predictive* or *hybrid* methods [HKR+14]. The VM reallocation based on the actual workload demand, rather than the reserved VM size, may potentially improve a PM's utilization. The improvement in utilization can be achieved by migrating VMs from underutilized (according to actual workload utilization) PMs. This migration process

may help switch some PMs into one of the power saving modes whenever possible. However, such a migration (and demand-based placement in general) may result in more service level agreement violations (SLAVs). A service level agreement (SLA) defines the quality of service (QoS) requirements (*e.g.* availability of VMs) in the form of a contractual document between the service provider and the customer [LS14]. An SLA violation (SLAV) occurs when previously agreed upon SLA requirements are not met. The two common sources of SLAVs in the IaaS are: (1) the unavailability of VMs due to the migration process, and (2) the overutilization of PMs due to the aggressive consolidation of VMs. Both reservation-based and demand-based VM placement strategies confirm that there is a trade-off between the utilization of PMs and the resulting SLAVs. Figure 1.4 demonstrates this trade-off where a better PM utilization (*i.e.* fewer number of PMs being used) may lead to a high number of SLAVs, whereas a low number of SLAVs may imply worse PM utilization.


Figure 1.4: Trade-off between PM's utilization and number of SLAVs

## 1.5 Motivation

According to Aristotle's philosophy, "[t]he golden mean or golden middle way is the desirable middle between two extremes, one of excess and the other of deficiency" [Gol]. However, so far, the existing VM placement literature has only considered the two extremes of the VM placement strategies. The first extreme strategy is the reservation-based placement, which results in utilization deficiency (underutilization) of the data centre's PMs. The second extreme strategy is the excess demand-based placement, which may tend to contribute to more SLAVs. These two extremes of the VM placement strategies suffer from resource/energy wastage due to resource overprovisioning in case of reservation-based placement and incurs financial penalties besides affecting customer satisfaction due to resource underprovisioning in case of demand-based placement.

Consequently, this thesis aims to find a more desirable and flexible strategy (by exploring the middle way) that could help to overcome the problems associated with the two extremes of the placement strategies. The proposed strategy seeks to exploit the area between demand-based and reservation-based placement striving for a workable and flexible balance between the number of SLAVs and the utilization of PMs according to some cloud providers' goals. For example, allowing a limited number of SLAVs may yield an economic benefit to cloud providers and better resource utilization while reducing cost for cloud customers.

## 1.6 Aim and Research Questions

This thesis aims to investigate the benefits of a novel VM placement strategy that can generate a multitude of VMs-to-PMs allocations beyond reservation-based and demand-based allocation. The proposed VM placement strategy introduces a tunable *parameter*, which becomes an input to the VM placement problem. Therefore, we have named this new placement strategy as *parameter-based VM placement* strategy. The parameter-based VM placement strategy seeks to find VMs-to-PMs mappings that strike a workable and flexible balance according to cloud providers' goals, such as minimizing energy consumption, improving PM utilization, and reducing the resulting SLA violations.

In developing this research, the following research questions were considered to help define the objectives to accomplish our research aim.

1. **What are the benefits of a strategy that combines both reservation-based and demand-based allocation of VMs to PMs?**
   To answer this question, the approach followed was to conduct a critical analysis of current VM placement solutions. The literature reports various measurable benefits for both reservation-based and demand-based allocation of VMs to PMs. These benefits of reservation-based placement include adherence to the QoS requirements, while demand-based ensures better utilization and energy savings, among others. The details of the advantages and the shortcomings of these two VM placement strategies are discussed at length in Chapter 2, within a novel classification of the state-of-the-art of VMs placement and consolidation solutions. Through this analysis, it was found that there are two principal metrics (namely, number of SLAVs and PMs utilization) that can measure the benefits of the existing VM placement strategies as shown in Figure 1.4. As a result, these two metrics, or variations thereof (such as the number of PMs, energy consumption and percentage of SLAVs), were chosen to evaluate our proposed strategy.

2. **What are the main components of the VM placement solution and how to model the proposed strategy to combine both reservation-based and demand-based allocation of VMs to PMs?**
   The critical analysis that serves to answer the previous question also helped to answer this question. Particularly, we also analysed the literature as to how the VMs-to-PMs placement problem has been modelled. Based on this analysis, we decided to design a system architecture and a mathematical model of the VM placement solution. The system architecture provides a framework to organize the software components of our strategy, and is described in Chapter 3. The mathematical model (described in Chapter 4) allows a quick and precise way to quantify the relationship between the number of SLAs and PMs utilization, along with other related metrics, and is thus the basis for the design of the various algorithms that comprise our strategy.

3. **How to find flexible VMs-to-PMs mappings by exploring the space between demand-based and reservation-based placement?**
   To answer this question, the research objective was to determine an efficient way for choosing VMs-to-PMs mappings that consider both demand-based and reservation-based placement. Our design decision was the definition of a single parameter $\alpha$ whose extreme values (zero and one) represent the two extremes of

VM placement strategies, namely, reservation-based and demand-based placement. The following equation formulates the operation of our parameter-based VM placement strategy.

$$\mathrm{a}_j^r(t) = \alpha \cdot (c_j^r - d_j^r(t)) + d_j^r(t),$$

where $\mathrm{a}_j^r(t)$ is the amount of resources allocated to $\mathrm{VM}_j$ at time $t$, $c_j^r$ is the capacity of resource $r$ per $\mathrm{VM}_j$ (reservation) and $d_j^r(t)$ represents the actual demand of resource $r$ at time $t$. That is:

$$\alpha = \begin{cases} 0, & \text{Demand-based VM Placement (only)} \\ 1, & \text{Reservation-based VM Placement (only)} \end{cases}$$

By simply changing the value of $\alpha$ (to a value between 0 and 1), our strategy makes it possible to define VMs-to-PMs mappings that combine both reservation-based and demand-based placement.

The proposed parameter-based VM placement strategy takes into account the current workload demand and the maximum resource size per each VM. Then, based on the VM placement objective, it seeks to utilize the PMs efficiently (maximizing the utilization ratio) by minimizing the estimated underutilization per resource type, while reducing SLAVs by reducing overutilization situations.

4. **How to estimate the parameter $\alpha$ on the fly to achieve the cloud provider's objectives?**
   Our research objective was to investigate and design various tuning algorithms that estimate a dynamic parameter which optimizes utilization without exceeding a certain threshold of number of SLAVs. Instead of using a predefined static parameter, such tuning algorithms should observe the predefined SLAVs threshold while actively seeking to optimize resource utilization. It should scale up a VM's allocated resources (bounded by the VM's maximum size) when the number of SLAVs exceeds the SLAVs threshold. On the other hand, it should scale down the allocated resources when the number of current SLAVs is lower than the SLAVs threshold to ensure better utilization of the PMs. The tuning algorithms can estimate the dynamic parameter either instantaneously (based on the current SLAVs and its distance from the predefined SLAVs threshold) or by considering the historical values of the SLAVs.

5. **How to incorporate the workload prediction of the individual VMs into our strategy?**
   To answer this question, the research objective to develop a hybrid approach that adjusts the value of the parameter $\alpha$ both reactively and proactively. Such a hybrid approach reacts to the current SLAVs and its relation to the SLAVs threshold, aiming to fine-tune the parameter $\alpha$ by making use of the predicted workload of each individual VM. For predicting the VMs' workload, we adopted two supervised learning algorithms, namely random forests and long short-term memory, which are illustrated in Chapter 6.

6. **How to build the cloud environment for testing and evaluating the performance of the algorithms in the proposed VM placement strategy?**
   An important question was to decide whether to use a real or simulated cloud environment to evaluate the proposed strategy. The decision was to find a proper cloud simulation framework to simulate the required cloud data centre and the VM placement strategy. Using simulation can help in overcoming the reproducibility and scalability problems associated with the real-world cloud experiments. Cloud simulation can make the testing and evaluation process more manageable, cost-effective and reproducible. This thesis made use of the well-known and widely-used open source CloudSim which stands out among counterparts in the VM placement research. The proposed algorithms have has been evaluated using both real (from PlanetLab) and synthetic workload traces.

## 1.7 Contributions

The key contributions of this thesis are summarized as follows:

1. Introducing a novel classification and a critical analysis of the state-of-the-art of the VM placement literature. The proposed classification presents a roadmap for introducing VM placement solutions. This roadmap can also be followed when creating autonomic systems in other domains of computer science. Additionally, it demonstrates common pitfalls in VM placement research and points out potential research directions. The details of the novel classification can be found in Chapter 2. During the review, an empirical evaluation of a genetic algorithm and simulated annealing has been conducted to reallocate VMs in CDCs dynamically; more details in Appendix A. Finally, a genetic algorithm

for dynamic VM placement that considers both CPU and memory resources was evaluated [MS19]; more details in Appendix B:

- *Abdelkhalik Mosa* and Rizos Sakellariou. Dynamic Virtual Machine Placement Considering CPU and Memory Resource Requirements. In 2019 IEEE 12th International Conference on Cloud Computing (CLOUD) (pp. 196-198). IEEE.

2. Designing a system architecture for the VM placement solution that utilizes the novel parameter-based VM placement strategy. The architecture includes components for initial VM placement, dynamic VM placement, and the parameter estimation. Additionally, establishing and calculating performance metrics that can be used for evaluating various VM placement solutions. More details can be found in Chapter 3.

3. A novel VM placement strategy (parameter-based) that works in the area between the actual resource demand and the full reservation of the VMs. This novel VM placement strategy offers the cloud provider more flexibility to choose between utilization and SLAVs tradeoffs. By and large, the proposed parameter-based strategy can motivate researchers to pay more attention to the middle way between extreme solutions as it can add significant value. The details of the parameter-based VM placement strategy can be found in Chapter 4. Parts of the material in Chapter 4 have been published in [MS17b]:

   - *Abdelkhalik Mosa* and Rizos Sakellariou. Virtual machine consolidation for cloud data centres using parameter-based adaptive allocation. In Proceedings of the Fifth European Conference on the Engineering of Computer-Based Systems, pages 1–16. ACM, 2017.

4. Introducing adaptive heuristics that can estimate the proper value of the parameter (of the parameter-based solution) on the fly to optimize resource utilization without exceeding the cloud provider's SLAVs thresholds. These heuristics can enable cloud providers to set SLAVs thresholds that should not be surpassed. The proposed dynamic tuning algorithms can be found in Chapter 5. Parts of this material have been published in [MS18]:

   - *Abdelkhalik Mosa* and Rizos Sakellariou. Dynamic tuning for parameter-based virtual machine placement. In Proceedings of the 17th International

Symposium on Parallel and Distributed Computing (ISPDC), pages 38 – 45, IEEE, 2018.

5. A proposal of a hybrid approach (reactive and proactive) to estimate the value of the parameter per VM. To do so, two supervised learning techniques (random forests and long short-term memory) are adopted to predict the future CPU utilization of the individual VMs. The details of the hybrid approach for the parameter estimation and the prediction of the VMs' resource utilization can be found in Chapter 6.

6. Conducting an empirical evaluation of the proposed VM placement strategy and the associated algorithms using CloudSim. The empirical evaluation has been conducted by using both synthetic and real workload traces. The details of the empirical evaluation can be found in Chapters 4, 5, and 6.

## 1.8 Summary and Thesis Structure

This chapter has introduced the scope of the research described in this thesis, which involves designing a novel flexible VM placement strategy that spans the area between the reservation-based and demand-based VM placement strategies. The novel aspects of the proposed VM placement strategy were outlined. A critical task involves developing a new autonomic VM placement solution to facilitate exploring the tradeoff between the costs of resource overutilization and SLA violations. Another task involves the autonomic management of VM consolidation without exceeding a given threshold of SLA violations while optimizing resource utilization.

The remainder of the thesis is structured as follows:

- Chapter 2 reviews the state of the art of VM placement as well as classifies the existing VM placement solutions based on the type of the problem being solved, the VM placement strategy being followed, the adopted policies for making the VMP solution autonomic and the placement objectives.

- Chapter 3 describes the system design of the proposed parameter-based VM placement solution. Additionally, it illustrates the design assumption, the simulation environment, and the performance metrics that are used to measure the effectiveness of the proposed solution.

- Chapter 4 introduces the proposed parameter-based VM placement solution based on our novel parameter-based VM placement strategy. It demonstrates a heuristics-based VM placement solution for both the initial and the dynamic placement of the VMs; Chapter 4 is based on [MS17b].

- Chapter 5 presents the dynamic algorithms that automatically estimates and fine-tune the parameter of the proposed parameter-based VM placement solution using reactive approaches; Chapter 5 is derived from [MS18].

- Chapter 6 adopts a hybrid approach (that is both reactive and proactive) to estimate the parameter per each VM based on the SLAVs threshold and the predicted CPU usage of the VMs. This chapter makes use of two parameters: one for the CDC level and the other for the VM level. Chapter 6 makes use of both random forests and long short-term memory (LSTM) to predict CPU utilization of the VMs.

- Chapter 7 concludes the work done and proposes a list of possible future directions that may require further investigation.

- Appendix A conducts an empirical evaluation of a genetic algorithm and a simulated annealing algorithm for the dynamic demand-based reallocation of VMs in cloud data centres for reducing energy and SLA violation costs.

- Appendix B examines the initial and dynamic reallocation of VMs using both CPU and memory requirements. It uses a genetic algorithm for the dynamic reallocation and compares it to the well-known best-fit decreasing algorithm. It confirms the importance of considering memory resources during VM placement, particularly in the case of memory-intensive VMs.

# Chapter 2

# VM Placement Problems, Strategies, Policies and Objectives

## 2.1 Introduction

Efficient virtual VM placement solutions enable operators of public or private cloud data centres (CDCs) to consolidate VMs on the least possible number of physical machines (PM)s which is an attractive option for energy and savings [Man15, BB12, MP16]. However, the VM placement problem is difficult for many reasons, including the NP-hard nature of the problem itself [HMGW07], the many variants of the VM placement problem [Man15], and the non-obvious framework/pathway to follow when proposing a new VM placement solution. The various variants of the VM placement usually result from the different cloud deployment models (*e.g.* public or private) and the type of service model (IaaS, PaaS, or SaaS), which leads to entirely different VM placement objectives and constraints. Cloud providers offer VMs in two formats: (1) VMs are offered directly to customers in the form of IaaS, and the user can install and run the required OS and applications, (2) VMs are offered as a wrapper of the applications or platform in case of SaaS or PaaS [ZCB10].

In addition to the previously stated difficulties, what makes it even more difficult is that many authors do not explicitly define the variant of VM placement that they are solving. The reader has to infer the type of the VM placement problem indirectly either from the proposed algorithms or from the conducted experiments [Man15]. As an illustration, one common variant is the process of assigning short-term jobs or tasks (in the form of VMs) to PMs. This variant is more related to workload or task scheduling than VM placement (as the VMs are used to wrap applications and not provided as an

IaaS). For that task scheduling problem, it may be better for the cloud provider to run these tasks on Linux containers instead of VMs as the performance of containers is equal to or outperforms VMs [FFRR15]. Furthermore, in the task scheduling problem, the cloud customer may care about the response time rather than the availability of the VMs. Therefore, a precise definition of the variant of the VM placement problem being addressed is critical. This thesis tackles the VM placement problem where the VMs are offered in the form of IaaS (previously introduced in Section 1.4). The cloud customers request VMs of a specific size (with particular resource characteristics such as the CPU and memory) for a particular time. The cloud operators offer their infrastructure directly to the customers in the form of various VM instances. As an example, Amazon EC2 offers multiple types of VM instances[1] to meet the needs of the diverse cloud customers. Then, cloud customers can run any number and type of applications on their VMs bounded by the capacity of the requested VMs. Accordingly, the VM placement solution is considered as application-agnostic as it does not have any prior knowledge of the nature of applications running by every single customer.

Following a critical literature review, this chapter introduces a novel classification of VM placement solutions. This novel classification points out the relevant issues in current VM placement solutions and demonstrates a clear framework/pathway for researchers to follow when they try to tackle any variant of the VM placement problems. In comparison to existing reviews of VM placement and resource management in CDCs (such as [Man15, PS16]), our review is different in what follows: (1) It concentrates only on the VM placement from the IaaS perspective. (2) It introduces a novel classification of the existing VM placement research. The following section provides an overview of our classification.

## 2.2 Overview of the Proposed Classification

The proposed novel classification has four aspects represented in four main classes and a few subclasses, as shown in Figure 2.1. The main classes are the VM placement *problems*, *strategies*, *policies* and *objectives*. To the best of our knowledge, there are no previous review articles (of VM placement) that have explicitly classified the VM placement based on the adopted VM placement strategies and policies. Our classification offers a high-level yet comprehensive view that can guide researchers to identify and reason about research avenues in VM placement. As an illustration, a researcher

[1] https://aws.amazon.com/ec2/instance-types/

can delineate his/her pathway when proposing a new VM placement solution by answering the following questions:

- Firstly, which version of the *VM placement problem* is he/she trying to solve?
  By and large, any VM placement solution tries to solve either an *initial (static)* or a *dynamic* version of the VM placement problem. Additionally, the literature solves both *online* and *offline* versions of the initial VM placement.

- Secondly, on reallocating a VM, should it be reallocated considering its reserved size or the actual resources it has been using (or it may require in the future)? This defines what we call the *VM placement strategy*.
  So far, all existing VM placement solutions follow either a *reservation-based* or a *demand-based* strategy to allocate resources to the VMs. Additionally, how to adapt to changes in the demand in case of a demand-based VM placement strategy? This can be achieved by following a *reactive*, *proactive* or *hybrid* approaches.

- Thirdly, how the VM placement solution is going to be self-managing (autonomic) by adapting to various changes in the CDC without requiring a human intervention?
  This can be achieved by following one the *VM placement policies* such as *action (rule-based)* or *utility functions* for creating an autonomic VM placement system. Adopting such policies allows the system to automatically change its state to a better one when necessary.

- Finally, what are the *VM placement objectives*?
  Any VM placement solution seeks to achieve multiple and usually contradictory objectives. Common examples of the VM placement objectives include *energy savings*, *efficient utilization of the PMs*, *minimizing SLA violations (SLAVs)*, *maximizing profit*, *balancing the load among the PMs*, *minimizing network overheads*, *reducing temperature*, or a *combination* of them.

The remainder of this chapter provides an in-depth investigation of the proposed classification of the VM placement research while discussing relevant literature under each category.

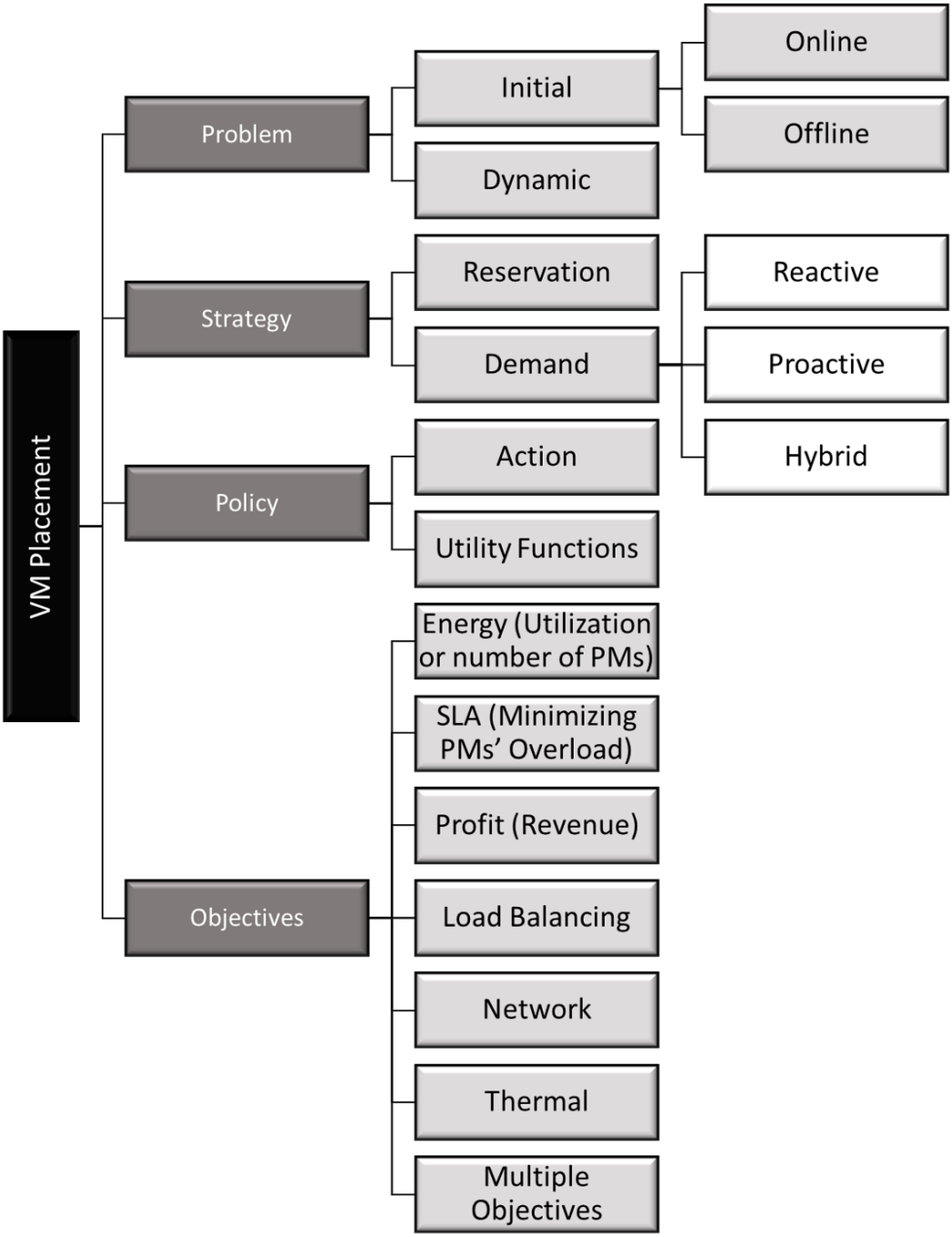



Figure 2.1: Classification of current VM placement research in IaaS

## 2.3 VM Placement Problems

The existing literature on VM placement attempts to solve either an initial (*a.k.a* static) or a dynamic version of the VM placement problem. A solution to an initial VM placement problem tries to find the most suitable PM to host the VM specified in a new VM placement request. An initial VM placement controller does not reallocate a VM after it has been initially allocated, regardless of any changes in the resource demand. Only when the lease duration of the VM is expired, it is completely deallocated. However, a dynamic VM placement controller uses the live migration utility to redistribute the VMs if this redistribution is expected to work towards achieving the cloud provider's objectives. By and large, the operators of CDCs use dynamic VM placement to ensure better utilization of data centres' resources.

Most of the literature on dynamic VM placement confirms that using the dynamic VM placement to reallocate VMs can reduce energy consumption, which results in financial and environmental gains. However, static VM placement might be more efficient than dynamic VM placement in some specific scenarios of private clouds implementations [WBS16]. For example, Wolke *et al.* [WBS16] have conducted a set of empirical experiments to evaluate the effectiveness of different types of static and dynamic resource allocation mechanisms in private (corporate) cloud environments. They conclude that using static resource allocation in corporate data centres is more efficient than using dynamic resource allocation for a typical workload of business applications. They suggested using live migration only exceptionally as it may cause more overheads, which can affect the response time. Again, this notion cannot be generalized as it is only valid for a specific scenario in private clouds and a particular nature of the workload (for example, as a fixed set of business applications with repeating workload patterns). Moreover, private clouds usually host a fixed number of VMs which host a set of corporate applications; however, in a public cloud, the number of VMs is changing, and VMs are being provisioned and deallocated continuously [WBS16]. This section presents an in-depth investigation and analysis of the current solutions for both the initial and dynamic versions of the VM placement problem.

### 2.3.1 Initial (static) VM Placement

A great deal of previous research into VM placement has focused on the initial (static) mapping of VMs-to-PMs. The mapping of VMs-to-PMs in CDCs is a variant of the

vector bin packing problem where PMs represent the bins, and the VMs represent the items that need packing [BB12]. Up to now, there are no polynomial-time algorithms that can solve bin packing problems and their variants, and therefore they have been classified as NP-hard problems [Hoc97]. Initial VM placement solutions have been further divided into *online* or *offline* versions [LPB15]. Offline versions receive the entire input of VMs, PMs, and workload in advance, and hence can produce an optimal solution of VM placement problem. In online versions of initial VM placement, all the input data is not known in advance, and the algorithm usually receives new VM placement requests over time. As an illustration, if there are $n$ VMs that need to be assigned to the existing PMs, then an offline solution must have all the $n$ VM placement requests in advance. On the contrary, an online solution is only given the $i^{th}$ VM placement request after mapping the previous $i-1$ requests. One of the common pitfalls on existing initial VM placement research is that it solves an offline version which does not reflect the online nature of the problem in public clouds. Therefore online algorithms are adopted in the real world practical implementations of VM placement controllers. Figure 2.2 demonstrates the function of the initial VM placement, where it receives requests for allocating VMs and tries to map the VMs to the most fitting hosts (PMs) based on the adopted filters and weights. The NOVA filter scheduler, described next, chooses the most weighted hosts after filtering out non-suitable hosts.

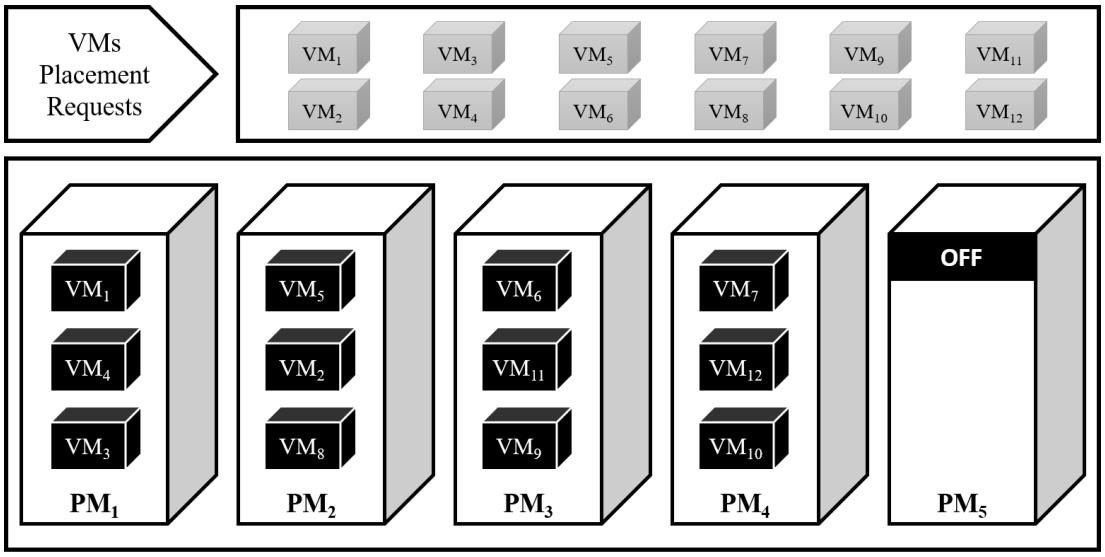



Figure 2.2: Initial (static) VM Placement

As an example of an exiting initial VM placement controller, this paragraph describes how does *OpenStack NOVA* allocates VMs to PMs in cloud data centres. OpenStack is a cloud management tool that controls compute, storage, and networking resources throughout a cloud data centre. OpenStack is a component-based cloud platform like Amazon Web Services (AWS) but developed by the community. OpenStack automates the creation and management of compute instances such as VMs using the NOVA[2] component while it uses other components to automate other services such as the network, security, and storage. The NOVA component uses *NOVA Filter Scheduler* [Fil] to allocate VMs to PMs (which is an example of an initial VM placement controller). The Nova-schedule process receives a VM request and determines where it should run (on which PM a VM should be launched). The Nova scheduler uses *filters* and *weighting* to choose a PM to which the VM should be allocated as shown in Figure 2.3. The NOVA filter scheduler chooses the most weighted hosts after filtering out non-suitable hosts. The following paragraphs review and analyse some of the research conducted for solving the initial VM placement problem.

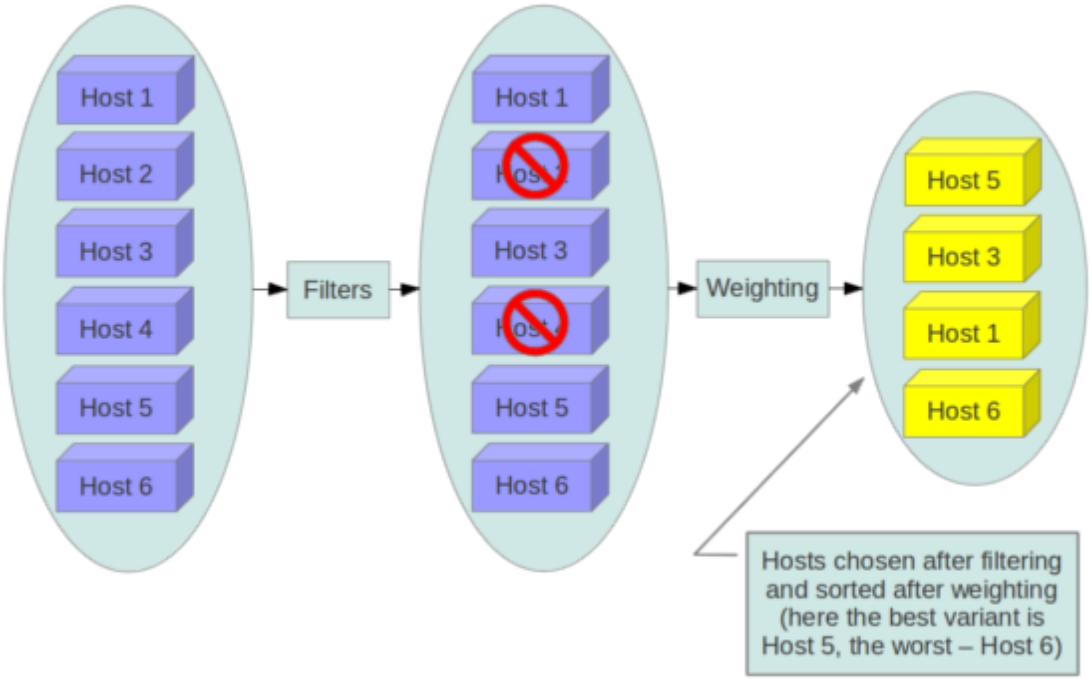



Figure 2.3: NOVA Filter Scheduler, from [Fil]

[2]https://docs.openstack.org/nova/latest/

Mills *et al*. [MFD11] have developed a standard objective method for evaluating various online versions of the initial VM placement algorithms in large clouds. They aimed to find a single standard method for comparing initial on-demand VM placement solutions instead of using multiple disjoint approaches. They declare that the majority of the existing initial VM placement solutions have an incomplete view of CDCs, which leads to inefficient placement decisions. The reason for this incomplete view is that they treat CDCs as an unstructured set of PMs and neglect the fact that PMs are grouped into racks/clusters. Therefore, they considered two levels during the evaluation, namely *cluster level* and *PM/node level*. Cluster-level decisions seek to identify the most appropriate cluster/rack of PMs. Within a selected rack, PM-level decisions select a PM onto which to assign the VMs. They have tested three different criteria for choosing the relevant cluster, namely least-full first (LFF), percent allocated (PAL), and random. Besides, they have evaluated six heuristics for placing the VM on the proper node within the chosen cluster. These algorithms are first fit (FF), LFF, most-full first (MFF), next fit (NF), random, and tag and pack (TP). They conclude that using two-level initial VM placement based on cluster and PM level yields better results than considering the flat architecture that relies only on the PM level. Moreover, they found that placing VMs of the same VM request or VMs belonging to the same customer to the same cluster achieves better network performance during communication among these VMs. However, they concluded that there are no significant quantitative differences among the 18 combinations of the initial VM placement algorithms that they tested. Eventually, they assure the importance of picking the proper criteria for cluster selection as it can significantly increase the cloud provider's revenue.

Using simple heuristics such as first-fit decreasing (FFD) or best-fit decreasing (BFD) for creating an online VM placement solution is a viable option due to their simplicity and strength. For example, Beloglazov *et al*. [BB12] begins with the initial assignment of VMs-to-PMs intending to minimize energy consumption. They propose a power-aware version of the BFD algorithm that considers power consumption during the placement. They sort all VMs in decreasing order based on the current CPU utilization and then allocate VMs to PMs that cause the least possible increase in power consumption due to the allocation. Another example, Shi *et al*. [SBBJ13] solve a variant of the initial VM placement problem with the aim of maximizing the revenue. Each set of VMs is constrained by both the individual VM's characteristics (CPU, memory, and bandwidth) as well as placement constraints. The placement constraints include security and anti-collocation requirements. To solve this problem, they propose two

approaches; one is based on the FFD while the other uses integer linear programming (ILP). The FFD-based solution classifies the VM requests into different categories and satisfies the placement constraints. For the ILP-based solution, they formulate the problem as an ILP problem that deals with the required placement constraints and VM types. The ILP solution provides an optimal VM placement; however, it is not suitable for the online placement of newly arrived VM placement requests and is not beneficial in case of medium and large-sized CDCs. Even though the FFD-based solution offers a sub-optimal solution, it can also provide near-optimal solutions for small-size problems. Most importantly, the FFD-based heuristic is capable of solving an online version of the problem by accepting new VM placement requests. They conclude that adopting the heuristics for the online initial VM-to-PM mapping are practical and more effective than the ILP counterparts.

On the other hand, global search metaheuristics such as ant colony optimization (ACO) and genetic algorithms (GA) have been adopted for the initial VM placement. For example, Ma *et al.* [LL$^+$12] expressed the initial VM placement problem as a multi-objective optimization problem and solved it using the ACO algorithm. They aimed to reduce three conflicting objectives, which are power consumption, SLA violations, and resource wastage. Zhao *et al.* [ZWL$^+$18] adopted ACO for solving the initial VM placement problem striving to find a balance between energy savings and the required performance of VMs. However, their experimental evaluation overlooks the scalability of the solution as they test their proposal using only ten VMs on homogeneous PMs. Ye *et al.* [YYL17] considered four objectives, which are reducing energy consumption and maximizing load balance, PMs' utilization, and robustness. They solved this problem using an energy-aware version of the knee point-driven evolutionary algorithm (KnEA), which they have named as energy-efficient KnEA (EEKnEA). Striving to reduce power consumption and costs, Yousefipour *et al.* [YRJ] formulated the VM placement problem as a mixed-integer nonlinear programming problem and introduced energy- and cost-aware VM consolidation (ECVMC) solution using a GA. On the other side, Quang-Hung *et al.* [QHNN$^+$13] solved an offline version of the static VM placement problem in private clouds using a power-aware genetic algorithm (GAPA). Their private cloud scenario considers a university cloud where students and researchers can request VMs during specific class times. Based on the findings of the solutions that adopt metaheuristics [LL$^+$12, YYL17, YRJ, QHNN$^+$13], they conclude that ACO, GA and KnEA outperform existing heuristics that use FFD, BFD and permutation pack (PP) algorithms. Additionally, it is found that ACO and KnEA

outperform GA [LL+12, YYL17]. However, these metaheuristics-based solutions are more relevant to offline VM placement, which is impractical for real scenarios on public cloud data centres. Using a metaheuristic for the initial VM placement is absurd as the algorithm will run each time a new VM placement request is received, which can have a high probability of changing the current VMs-to-PMs mapping that degrades the overall performance of the VMs. One more downside of adopting metaheuristics is that the majority of them are computationally expensive as opposed to the simple heuristics.

Virtual data centre (VDC) embedding problem, another variant of the VM placement problems, involves an efficient mapping of the VDC to the existing physical infrastructure in CDCs. Rabbani *et al.* [REP+13] propose a static VDC embedding solution, which assigns not only the VMs but also the virtual links as well as the virtual switches. Therefore, they have developed a centralized network-aware approach that allocates VDCs of any topology type by explicitly considering the virtual switches and links in the placement request. They focus on the networking infrastructure, which is usually ignored by many of the previous VM placement solutions. To do so, they allocate VMs of the same request to the same PM whenever possible to reduce network overhead as well as resource fragmentation. Their proposed heuristic solution balances the load while considering communication costs. Wuhib *et al.* [WYS13] have developed an architecture that simultaneously allocates compute and network resources to VDCs. The proposed architecture consists of multiple controllers where each of them handles a different objective such as energy efficiency, load balancing, fair allocation, and service differentiation. They address both the initial and dynamic allocation of the VMs. For the initial placement of the VDC, they allocate one VM at a time. For finding the proper mapping of each VM, they introduced an algorithm that operates as follows; for each VM, it selects $d$ random PMs and then places the VM on the PM that minimizes the value of the objective function. If no feasible solution is found, then the algorithm selects another $d$ random PMs. This algorithm is more viable for large-scale CDCs as opposed to search-based counterparts.

To investigate the effect of the resource consumption of the co-located VMs on the performance of the individual VMs, Corradi *et al.* [CFF14] propose a practical energy-aware VM consolidation solution for OpenStack. Their proposed solution aims to reduce energy consumption and resource wastage while considering networking requirements. Their empirical evaluation indicates that a VM consolidation solution may be theoretically feasible; however, it may not guarantee the SLA requirements

due to the interference among the co-located VMs. The degradation of the VMs' performance depends on the type of services running on the co-located VMs. They conclude that any VM consolidation solution should thoroughly take the interference due to aggregated resource consumption among the co-located VMs into consideration to satisfy the SLA requirements.

Reviewing the VM placement literature confirms that the area of initial VM placement is ripe enough. Moreover, it assures the importance of examining the hierarchical structure of the CDC as opposed to the simplified flat architectures. Finally, online solutions for the initial VM placement problem are practical as they reflect the continuous arrival of VM placement requests on public clouds as opposed to the offline counterparts.

### 2.3.2 Dynamic VM Placement

On the contrary to initial VM placement, a dynamic VM placement solution starts with an existing VMs-to-PMs mapping (which reflects the current state of the system). Then, the dynamic VM placement solution attempts to reach a new mapping (state) that achieves the placement objectives without violating the placement constraints. A dynamic VM placement solution adapts to changes in the cloud environment by migrating VMs among PMs, aiming at a more efficient VMs-to-PMs mapping. On the one hand, dynamic VM placement may result in some unused PMs being turned off or switched to lower power modes, which can save energy [DAGLO12]. On the other hand, dynamic VM placement may result in more SLA violations (SLAVs) due to the temporal unavailability of the VMs during the migration time [LXJ+11] . None of the existing cloud management tools, such as OpenStack, considers the dynamic placement of VMs after the initial allocation [WTAPB15].

Figure 2.4 illustrates how the dynamic VM placement solution migrates VMs from underutilized PMs, and switch unused PMs off to save energy. As an example, the dynamic VM placement controller migrates $VM_7$ from $PM_4$ to $PM_3$ and switches $PM_4$ off, which reduces the running PMs and hence saves energy consumption. The aggressive consolidation of VMs may result in overload situations, which can degrade the quality of the service provided. As a result, the dynamic VM placement controller starts migrating one or more VMs from an overutilized PM until it becomes no longer overutilized. If none of the running (active) PMs is able to host the migrating VM, then the dynamic VM placement controller switches one of the inactive (powered off) PMs on so that it can host the VM as shown in Figure 2.5.

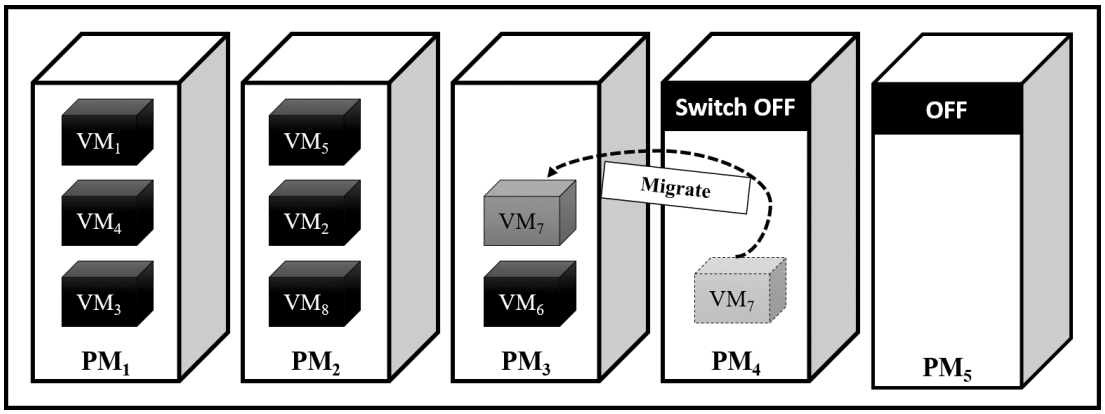



Figure 2.4: Migrating VMs from $PM_4$ (underutilized), and switching $PM_4$ off

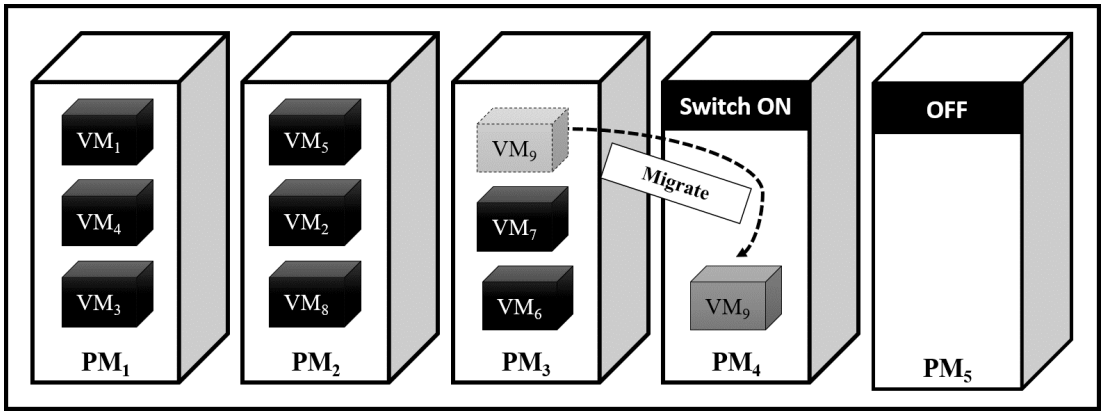



Figure 2.5: Migrating $VM_9$ from $PM_3$ (overutilized), and switching $PM_4$ on

There is a relatively large body of VM placement literature that is concerned with the dynamic reallocation of the VMs following the initial placement of the VMs to adapt to workload changes in the CDC. Examples of these dynamic VM placement solutions include [LLW11, CBHM12, BB12, CD13, XSC13, WYS13, BS14, CMR15, MP16, FAP+15, ZHL16, MR16, KDA+17, WT18, LSWJ18]. The act of changing the system state of the current VMs-to-PMs mapping in the dynamic VM placement solution requires a triggering mechanism. There are various triggering mechanisms for VM reallocation in dynamic VM placement solutions, including periodic, event-driven, hybrid, or threshold-based [PS16]. The following sections review and analyse the current VM placement solutions based on the adopted strategies (Section 2.4), policies (Section 2.5), and objectives (Section 2.6).

## 2.4 VM Placement Strategies

Following the identification of the type of VM placement problem being solved (initial or dynamic or both) and the choice between the online and offline versions, the next fundamental question for the dynamic VM placement solution is: On what basis is it going to reallocate the VMs? The current literature on dynamic VM placement solutions dynamically reallocates VMs based on either the reserved VM size as specified by the user *(reservation-based*) or the actual workload demand (*demand-based*). In this thesis, we refer to the basis on which the VMs are reallocated as the *VM placement strategy*. The adopted VM placement strategy determines the amount of resources that need to be allocated to each VM. Thus, the reservation-based VM placement strategy allocates resources that are equal to the entire VM size, even if the VM is completely idle. However, the demand-based VM placement strategy allocates resources corresponding to the actual demand of each VM, bounded by the VM size. The reservation-based VM placement strategy ensures better availability and durability; however, it might lead to underutilization of PMs if customers do not fully utilize their VMs during the reservation period. The demand-based VM placement strategy resizes the allocated resources for each VM based on the actual workload demand. As opposed to the reservation-based VM placement strategy, the demand-based VM placement strategy can guarantee more efficient utilization of the PMs but may incur additional SLAVs. Recall that, the efficient utilization of the PMs helps using less number of PMs through consolidating VMs into fewer PMs. The remainder of this section reviews reservation- and demand-based placement strategies on the current VM placement literature.

### 2.4.1 Reservation-based VM Placement Strategy

The reservation-based VM placement strategy can be applied to both the initial and dynamic versions of the VM placement problems. All the existing initial VM placement solutions in the literature employ the reservation-based placement to allocate resources to the VMs. However, there is a limited number of research articles that adopt reservation-based VM placement for the dynamic reallocation of the VMs. Compared to the initial VM placement, a dynamic reservation-based VM placement strategy is aware of some of the changes in the cloud environment (such as the deallocation of the VMs) and can migrate VMs accordingly. As a result, a dynamic reservation-based VM placement tends to improve resource utilization when compared to the initial VM placement.

Lin *et al.* [LLW11] have proposed two algorithms to dynamically consolidate VMs in CDCs using a reservation-based VM placement strategy. Both algorithms employ the notion of *retiring* PMs and the *retirement threshold*. Their proposed algorithm sets a PM in retiring state when one of the hosted VMs is deallocated (after finishing its job or by the end of the booking time). When the algorithm marks a PM as retiring, it rejects any new VM placement requests to that same PM so that it can switch the PM off to save energy after deallocating the other hosted VMs. To speed up the consolidation process, they introduced the retirement threshold, which specifies the waiting time before the algorithm forces the migration of the remaining VMs in a retiring PM. The first algorithm is a modification of the original round-robin, which they have called dynamic round-robin (DRR); the second is a hybrid one that employs either the first fit (FF) algorithm or the DRR based on the incoming rate of VMs. The hybrid algorithm uses FF when the incoming rate of VMs is high; otherwise, it uses DRR. They conclude that the hybrid algorithm offers the best results in terms of the best savings rate for both energy consumption and the number of running PMs.

Based on the fact that cloud customers do not book VMs for the same time and therefore, some VMs will end before others (VMs are deallocated), Borgetto *et al.* [BS14] have proposed a dynamic reservation-based VM placement solution. Their proposed algorithm dynamically reallocates VMs intending to save energy consumption while maintaining the resources reserved by the users. They used a modified vector packing algorithm to redistribute VMs in order to efficiently make use of underutilized PMs after deallocating some of the hosted VMs. Their proposal takes the overheads resulting from switching the PMs on and off into consideration. Their proposed scheduler shows energy savings compared to the default OpenNebula's scheduler. Finally, they have conducted small-scale experiments using only six PMs. Thus large-scale experiments need to be conducted to verify the scalability of their solutions in large-scale cloud scenarios.

### 2.4.2 Demand-based VM Placement Strategy

Much of the current literature on dynamic VM placement solutions pays particular attention to the demand-based VM placement strategy (significantly more than the reservation-based VM placement) since it is expected to ensure better energy savings. A dynamic demand-based VM placement controller dynamically reconfigures the VMs-to-PMs mapping in order to optimize the cloud providers' objectives (*e.g.* energy, revenue, or SLAs). The workload demand identification method and the action

taken due to the change in the workload makes the VM placement solution dynamic rather than static. The dynamic demand-based reallocation of the VMs requires run-time monitoring of the PMs' resource utilization as well as the hosted VMs. By monitoring the resource utilization, the dynamic VM placement controller can react to overutilization and underutilization situations by migrating VMs using live migration. Furthermore, a dynamic VM placement controller can predict resource usage by adopting the relevant time series prediction models, which enables it to put the expected changes in the CDC into consideration beforehand. There is a considerable amount of previous studies that focused on creating dynamic VM placement solutions that adopt the demand-based VM placement strategy. The remaining paragraphs of this section describe some of these studies.

To adapt to the ongoing allocation and deallocation of VMs, the work of Calcavecchia *et al*. [CBHM12] consider the continuous optimization of the VMs (after the initial VM placement) to ensure better satisfaction of the VM's demand. They propose a technique called Backward Speculative Placement (BSP), which assumes previous knowledge of historical workload traces of VMs in the data centre. The BSP technique divides the VM placement into two phases. The first phase is called continuous deployment (an example of an online VM placement controller), which only deals with the newly received VM requests, and it assigns VMs-to-PMs using a BFD algorithm. The BFD algorithm places VMs according to a score function, which measures the level of demand dissatisfaction. The second phase is called ongoing optimization (an example of a dynamic VM placement controller), which aims to optimize the current VMs-to-PMs mapping by migrating VMs from the most loaded PMs to others that are likely to ensure a better demand satisfaction.

Aiming to reduce energy consumption (by minimizing underutilization per PMs) while satisfying SLA requirements (by reducing overutilization situations), Beloglazov *et al*. [BB12] developed a dynamic VM placement solution. After the initial allocation of the VMs to PMs using a modified version of BFD (power-aware BFD), they follow a heuristic-based approach to automatically consolidate the VMs onto the least possible number of PMs while reducing the SLAVs. Their proposed approach follows a divide and conquer concept by dividing the dynamic VM placement problem into four subproblems, namely host overload detection, host underload detection, VM selection, and VM placement. The subproblem of host overload detection involves deciding when a host is considered as overloaded. Once the host is deemed to be overloaded, some VMs should be selected for migration from this overloaded host to a non-overloaded

one until that host becomes no more overutilized. They proposed three techniques for solving the host overload detection problem using static or adaptive thresholds. The subproblem of host underload detection aims to detect underloaded hosts. After the discovery of an underloaded host, all VMs in that host should be migrated to another host if possible and switching that host into a power-saving mode. The VM selection subproblem tries to select VMs for migration from the overloaded hosts to decrease the SLA violations (SLAVs). For the VM selection subproblem, they proposed three algorithms, namely minimum migration time (MMT), random selection (RS), and maximum correlation (MC). Eventually, the VM placement subproblem aims to determine the appropriate PMs that should host the VMs migrated from both overloaded and underloaded PMs. Their proposed solution periodically reallocates VMs either to achieve energy savings by turning some PMs off or to minimize the resulting SLAVs by migrating the VMs to non-overloaded PMs or even by switching some PMs on to host the migrating VMs.

Several VM placement proposals aimed to achieve further energy savings while reducing SLAVs by introducing new algorithms for the subproblems of Beloglazov's *et al.* [BB12] proposal. The following list summarizes some examples of these solutions as follows:

- Cao *et al.* [CD13] propose a modified heuristic-based VM consolidation framework. Their proposal is based on the idea that there is no certainty that an overloaded host will eventually lead to SLAVs. An overloaded host may or may not violate the SLA requirements. Therefore, migrating VMs from overloaded hosts that are not expected to generate SLAVs is not beneficial and will result in more energy consumption. Accordingly, their work seeks to enhance the VMs' selection from overloaded hosts to include only hosts that are expected to violate the SLA requirements. To do so, they initially identify overloaded hosts with one overload detection algorithm such as interquartile range (IQR) or local regression robust (LRR) or median absolute deviation (MAD) [BB12]. Then, they classify any overloaded host as either likely or not likely to incur SLAVs. The VM selection algorithm selects VMs from overloaded hosts with SLA violation to migrate them to other suitable hosts; and then selects VMs from overloaded hosts without SLAVs using the proposed SLAV decision.

- Chowdhury *et al.* [CMR15] have introduced a new set of VM placement solutions based on both the first-fit decreasing (FFD) and worst-fit decreasing (WFD) algorithms. They have presented a modified WFD (MWFD), second

WFD (SWDF), and an FFD with a decreasing host. For finding a proper mapping for the VMs selected for migration, they cluster those VMs and start by allocating VMs from high-density clusters first. By so doing, they managed to consolidate the VMs with less energy consumption and SLA violations compared to the existing heuristics.

- Monil *et al*. [MR16] propose a new algorithm for host overload detection and another one for VM selection. Their algorithm for host overload detection aims to foretell whether the host is going to be overloaded or not by making use of the mean, median, and standard deviation of the last $n$ resource demands of the VMs running on that host. For the proposed VM selection algorithm, a VM is going to be selected for migration if the VM's CPU utilization is lower than the migration control threshold; otherwise, the VM with the highest fuzzy value will be chosen first. They conclude that the results from the proposed algorithms outperform the existing heuristics.

- Khoshkholghi *et al*. [KDA+17] have proposed algorithms for host overload and underload detection as well as for VM selection and placement. The host underload detection algorithm uses a vector magnitude squared of multiple resources. With the new algorithms, they managed to reduce both energy consumption and the resulting SLAVs. Moreover, they have proposed an algorithm that uses an iterative weighted linear regression approach to decide whether a host is overloaded or not. They have also introduced three algorithms for selecting VMs from the overloaded hosts. Finally, they have added a two-phase VM placement approach that can effectively map both new VM placement requests as well as the migrating VMs.

- Wang *et al*. [WT18] adopted a modified version of the BFD algorithm to allocate the VMs to the suitable PMs without violating the placement constraints. They named the proposed VM placement solution as space aware best fit decreasing (SABFD). By space-aware, they mean the available CPU resources in a PM after allocating a VM to it. In addition, they have proposed a new approach for VM selection from overutilized PMs. Their approach selects the VM with the highest CPU utilization for migration from the overutilized PMs.

The majority of dynamic VM placement solutions in the literature consider only a single resource (CPU) and ignore other resources such as memory and bandwidth.

However, other dynamic VM placement solutions jointly consider multiple resource types as follows:

- Xiao *et al*. [XSC13] try to dynamically reallocate the VMs in the CDC based on the workload demand. From this reallocation, they aim to minimize the overload situations per the PMs as well as reducing energy consumption. To improve the utilization of the PMs across all resource types (CPU, memory, and bandwidth), they have introduced the skewness concept so that they can appropriately consolidate the VMs efficiently. Furthermore, they have deployed load prediction mechanisms to reduce the number of overload situation whose effectiveness has been confirmed is supported by the conducted experiments.

- To jointly allocate compute and network resources dynamically to VDCs, Wuhib *et al*. [WYS13] have proposed a dynamic VM placement controller that can switch between different management objectives. They have developed a distributed architecture that dynamically estimates VM allocations based on a modified version of a highly-scalable, generic gossip protocol called the generic resource management protocol (GRMP). The GRMP algorithm tries to find a better allocation based on the specified management objectives. The GRMP algorithm enables PMs to share their states (*e.g*. the set of hosted VMs) using a push-pull interaction. Each PM can compute a new VMs-to-PMs mapping and apply it with the help of live migration.

- Using a genetic algorithm, Mosa *et al*. [MS19] propose a VM placement solution that dynamically reallocates VMs based on the actual demand of the individual VMs. The proposed solution considers different resource types (namely, CPU and memory) while aiming to minimize underutilization and overutilization scenarios in the cloud data centre. The conducted experiments highlight the importance of considering multiple resource types. Finally, they conclude that the genetic algorithm outperforms the well-known best-fit decreasing algorithm for dynamic VM placement.

**Reactive, Proactive and Hybrid Demand-based VM Placement**

Commonly, an autonomic demand-based VM placement solution can adapt to changes in the CDC using *reactive*, *proactive* or *hybrid* approaches [HKR$^+$14]. A dynamic VM placement solution that follows a reactive approach responds to the current state

of the system by migrating VMs to meet the cloud provider's objectives. The reactive controller monitors the status of the CDC and makes changes to VMs-to-PMs mappings only based on the current state, and it is agnostic of any future forecasts. On the contrary, a proactive approach employs prediction methods to forecast the anticipated workload and acts accordingly. Predictive methods use historical workload data to predict the expected resource demand with the help of workload prediction models. Proactive VM management works well when the VMs' workload is periodic or seasonal. Therefore, many VM placement solutions assume that VMs present specific repeatable patterns that can be easily predicted (*e.g.*patterns repeated at a particular time of the day, month, or year). However, proactive approaches do not work well when the workload is unpredictable. Finally, hybrid approaches combine reactive and proactive (predictive) approaches to make more efficient VM migration decisions.

A number of dynamic VM placement solutions have adopted various machine learning techniques and statistical prediction methods to achieve better PM utilization ratios as well as reduce SLAVs from the unnecessary frequent migrations and the overutilization scenarios. The list below describes some of the VM placement solutions that adopt prediction methods.

- Zhou *et al.* [ZHL16] proposed an adaptive three-threshold energy-aware algorithm (ATEA) algorithm that makes use of the historical utilization data. They have set three thresholds to determine four classes of hosts based on the actual load (little, light, moderate and heavy). They have proposed adaptive algorithms to decide the threshold values based on the k-Means clustering algorithm. Additionally, they have introduced three approaches for selecting VMs from overloaded PMs based on the memory size and the CPU utilization of the VMs. They conclude that the proposed dynamic thresholds are more efficient than the fixed counterparts.

- Farahnakian *et al.* [FPLP13] have proposed a dynamic VM placement solution that consolidates CDC's VMs on the least number of PMs with the help of a k-nearest neighbor (KNN) supervised learning algorithm. They managed to predict resource utilization, which is used to proactively figure out both underutilization as well as the overutilization situations per each PM. Compared to other reactive approaches, their proactive approach of forecasting underutilized and overutilized PMs enabled them to reduce energy consumption and SLAVs.

- Nguyen *et al.* [NFYJ18] have proposed a new VM placement solution that aims

to improve energy efficiency using multiple usage prediction. They used multiple linear regression to predict the future utilization of multiple resource types. To improve energy efficiency, they sought to reduce both the frequent changes in the power state of each PM and the unnecessary numerous VM migrations. They conclude that the prediction of resource usage enabled them to achieve better energy saving rates with satisfying compliance with the quality of service requirements.

There are relatively few proposals that considered hybrid approaches to allocate resources in data centres efficiently. For example, Gandhi *et al*. [GCG+12] propose a hybrid method for resource allocation that deploys reactive and predictive controllers aiming to reduce SLAVs and energy consumption. Their proposal consists of four controllers, namely base workload forecaster, predictive controller, reactive controller, and the coordinator. Their solution makes use of the periodic patterns in workloads, and hence the base workload forecaster analyses historical data to find patterns that represent the base workload. The base workload forecaster discretizes the patterns using fast Fourier transform (FFT). The predictive controller takes the foretasted patterns as input, estimates the required resources, and allocates the resources accordingly. The coordinator takes the actual workload as input and forwards it to the predictive controller if it does not exceed the base workload. Additionally, the coordinator transmits the actual workload to the reactive controller if the actual workload exceeds the base one. The reactive controller handles the excess demand by adding more servers.

To reduce overload situations of the PMs in CDCs, Xiao *et al*. [XSC13] have used a proactive approach to predict the workload, which results in preventing over 46% of overloaded PMs on their experiments. Bobroff *et al*. [BKB07] deployed a proactive, dynamic VM placement solution that adapts to the changes in the workload demands to minimize the number of running PMs while meeting the specified SLAVs' rate. The solution uses three main modules, namely *measurement*, *forecasting* and *placement*. The measurement module monitors resource utilization; the forecasting module predicts the demand for the next interval, and the placement module allocates VMs to PMs and migrates VMs accordingly. The placement module finds an approximate solution using the well-known first-fit algorithm. However, their proposed solution only considers the CPU and will only work well with offline scenarios.

Regardless of the apparent advantages of deploying proactive solutions for VM placement in CDCs, not all VMs can benefit from them. For ensuring better optimization with low prediction error, the workload should express high variability as

well as strong autocorrelation [BKB07]. Therefore, the proactive VM placement is only beneficial when the VMs' workload exhibits significant variability. VMs with low variability in the workload will barely benefit from dynamic VM reallocation. It is worth mentioning that many proactive VM placement solutions assume predictable workload patterns, which may not always be applicable.

## 2.5 Virtual Machine Placement Policies

The dynamic placement of VMs in CDCs involves developing an *autonomic* (*self-managing*) system where the VM placement solution adapts to various changes by reallocating the VMs accordingly. In 2001, IBM launched the concept of *autonomic computing* to refer to self-managing systems that automatically adapt to the changing requirements of the system and manage upcoming issues with minimum human intervention [SB03]. An autonomic system manages its behavior according to the system's objectives by deploying the appropriate *policy*. A policy causes the self-managing system to take actions that change its current state to a better one. There are three common policy mechanisms for controlling any self-managing system namely *action (rule-based)*, *goal* and *utility function* policies [KD07]. Regardless of the adopted policy type, the self-managing system decides the actions it should take to move from its current state to a better new state, as shown in Figure 2.6.

### 2.5.1 Action-based (Rule-based) VM Placement Policy

The action (*a.k.a* rule-based) policies take the form of If (condition) then (action). An example of a *condition* might be the overutilization or the underutilization thresholds of a PM expressed as a percentage of CPU or memory use. An example of an *action* is the migration of some VMs to mitigate the overutilization and underutilization situations. The action policies only care about what action the system should do whenever it is in any given state. The action policies do not specify the state that the system will reach after taking action. As an illustration, suppose that there is a system that has an autonomic manager which aims to avoid overutilization and underutilization situations of the PMs. If that system adopts an action policy, then it will try to migrate some VMs from the overutilized PMs, or it will try to migrate all VMs from underutilized PMs and switch them to a power-saving mode. Therefore, the possible states might

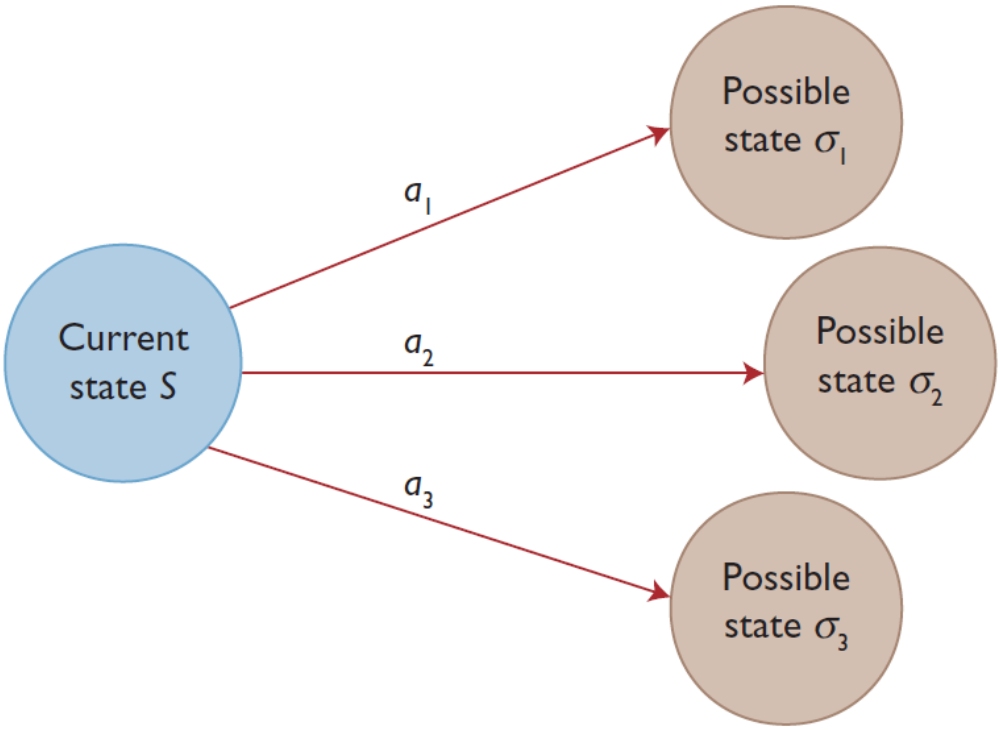



Figure 2.6: States and actions of an autonomic system [KD07]

be {overloaded PMs, underloaded PMs}, and the action is {migrating some VMs, migrating all VMs}. Each state is a result of the current VMs-to-PMs mapping. The following pseudo-code demonstrates an example of a possible action-based (rule-based) manager.

```
1: if (cpuUtilization >= 90%) then                ▷ Overutilized PM
2:     migrateSomeVMs()
3: else if cpuUtilization <= 40% then             ▷ Underutilized PM
4:     migrateAllVMs()
5: else
6:     do nothing
```

A high proportion of the existing VM placement solutions in the literature adopt action-based policies [LLW11, CBHM12, BB12, CD13, XSC13, WLFJ13, BS14, CMR15, ZHL16, MR16, DHC16, KDA+17, WT18].

### 2.5.2 Goal-based VM Placement Policy

On the contrary, goal policies specify the desired state of the system. The goal policy cares about computing the action that moves the system from the current state to the desired state. Put simply, goal policies classify the system states as either desirable or undesirable and take the necessary action to move the undesirable state to a desirable one. The following pseudo-code shows an illustration of an autonomic manager developed using goal policies:

$$40\% <= cpuUtilization <= 90\%$$

The downside of goal policies is that they do not enable a fine-grained expression of the preferences as they only perform a kind of binary classification against the current state of the system. Therefore, a goal policy accepts the current state as long as it is within the range of the desired states, even if there is a better state that should be considered instead. To the best of our knowledge, no VM placement solution in the literature relies on goal policies for the dynamic placement of the VMs.

### 2.5.3 Utility-based VM Placement Policy

Utility function policies compute the desired state by repeatedly choosing the state with the highest utility from the possible ones. Utility functions seek to find the state that maximizes the utility, and they are considered as an extension of goal policies without having to specify the desired state in advance. Utility functions entail selecting the appropriate utility attributes followed by modeling the utility function and finally using a proper search algorithm for optimizing the system. The following pseudo-code shows an example of an autonomic manager developed using a utility function [MP16]:

$$Utility(a,t) = Income(a,t) - EnergyCost(a,t)$$

$$Income(a,t) = \sum_{VM \in a} IncomePerVM(VM, cost) - ViolationCost(a,t)$$

The above utility function aims to maximize the cloud provider's profit from hosting the VMs by minimizing the energy and SLA violation costs. Here, the utility function is defined by the high-level objectives of the cloud provider. Utility function policies are more relevant for creating self-managing systems as they focus on the desired state rather than the current state [KD07]. Utility-based policies offer a holistic view of the decision making process. In addition to that, utility functions are favored

over goal policies as they enable more fine-grained behavior by providing more flexibility in expressing the system's attributes. Examples of dynamic utility-based VM placement solutions include [FMCB14, FAP+15, MP16, YRJ].

Utility-based dynamic VM placement solutions adopt various search algorithms that enable them to find the state with the highest utility. For example, Mosa *et al.* [MP16] have deployed a meta-heuristic search algorithm using GA that explores several VMs-to-PMs mappings to find the one that maximizes the utility. To achieve that goal, they have formulated a utility function that maximizes the profit by increasing the income and decreasing the associated costs. The two primary sources of costs (that they aimed to decrease) were energy costs as well as the SLAVs' penalties. Their proposed solution outperforms existing heuristics by minimizing the amount of energy consumed as well as SLAVs from both overloaded PMs and VMs' migrations. Farahnakian *et al.* [FAP+15] is another example of a utility-based solution that deploys ant colony system (ACS) meta-heuristic to search for a near-optimal VMs-to-PMs mapping that achieves the consolidation goals. They propose a dynamic VM placement approach that aims to consolidate VMs in a way that reduces the energy consumption and satisfies the SLA requirements. They have formulated the VM consolidation problem as a multi-objective optimization problem that considers both the number of inactive PMs as well as the number of VMs' migrations. They conclude that the proposed dynamic VM placement solution using ACS outperforms existing heuristics in terms of energy savings, minimizing the SLAVs as well as minimizing the total number of VM migrations. Simulated annealing (SA) was proven to find efficient solutions for such NP-complete problems [Kir84]. Therefore, Appendix A demonstrates an empirical evaluation of a genetic algorithm (GA) versus simulated annealing (SA) algorithm for the dynamic reallocation of VMs based on the actual workload demand. The results of the empirical evaluation confirm that GA outperforms SA by saving more energy while reducing SLAVs. Additionally, it is found that GA can produce a more desirable solution than SA when limiting the execution time of the experiment.

By reviewing existing utility-based solutions for dynamic VM placement, it is evident that utility functions outperform action policies for creating self-managing systems. Such conclusion conforms with the conclusions previously made by Kephart *et al.* [KD07]. However, one of the major difficulties with utility-based policies is the choice of appropriate parameters for the adopted metaheuristic. The majority of metaheuristics require parameter tuning to produce robust results. Another concern about the utility-based solution is the computational complexity and the total execution time.

## 2.6 Virtual Machine Placement Objectives

Regardless of the nature of the VM placement problem and the deployed strategy and policy, any VM placement solution seeks to achieve distinctive and sometimes contradictory high-level optimization objectives. Common VM placement objectives include *increasing profit or revenue*, *saving energy consumption*, *minimizing the number of powered-on PMs*, *improving PMs' utilization*, *balancing the load among the active PMs*, *minimizing network communication overheads*, *satisfying the SLA requirements*, *reducing the resulting temperature* or *ensuring security requirements*. Furthermore, some VM placement solutions might consider other customer requirements such as anti-co-location where VMs belonging to the same customer should be placed on different PMs to avoid a single point of failure. By analysing the objectives proposed in previous work, one can find that some of them are related and reflect the same objective, and they share the underlying goals. For example, energy efficiency, efficient utilization of the PMs, and minimizing the number of running PMs almost reflect the same goal. Moreover, profit and revenue are in the same category; minimizing SLAVs, ensuring QoS requirements, and minimizing overload situations can also be considered as one class of objectives. In what follows, we classify existing VM placement solutions based on the high-level placement objectives for both initial and dynamic VM placement problems, as shown in Figure 2.7.

### 2.6.1 Energy and Utilization-aware VM Placement

Energy efficiency is one of the most widely considered VM placement objectives due to its economic and environmental impact on data centres. Energy consumption represents nearly 50% of the total operational costs of the data centres [FHG+08]. Additionally, the power usage efficiency is a highly-ranked metric by the majority of data centre operators [Sta15]. Consequently, most of the research carried out on both initial and dynamic VM placement aim to decrease energy consumption in CDCs. Generally, power management techniques are either static or dynamic, and each of them can be achieved at the hardware or software level [BBLZ11]. The most common power-saving techniques in VM placement include dynamic voltage and frequency scaling (DVFS), soft scaling, efficient utilization of the PMs through VM consolidation, and switching unused PMs into one of the power saving modes. DVFS can save energy consumption by lowering the CPU frequency. More *et al.* [MMP+14] propose an energy manager

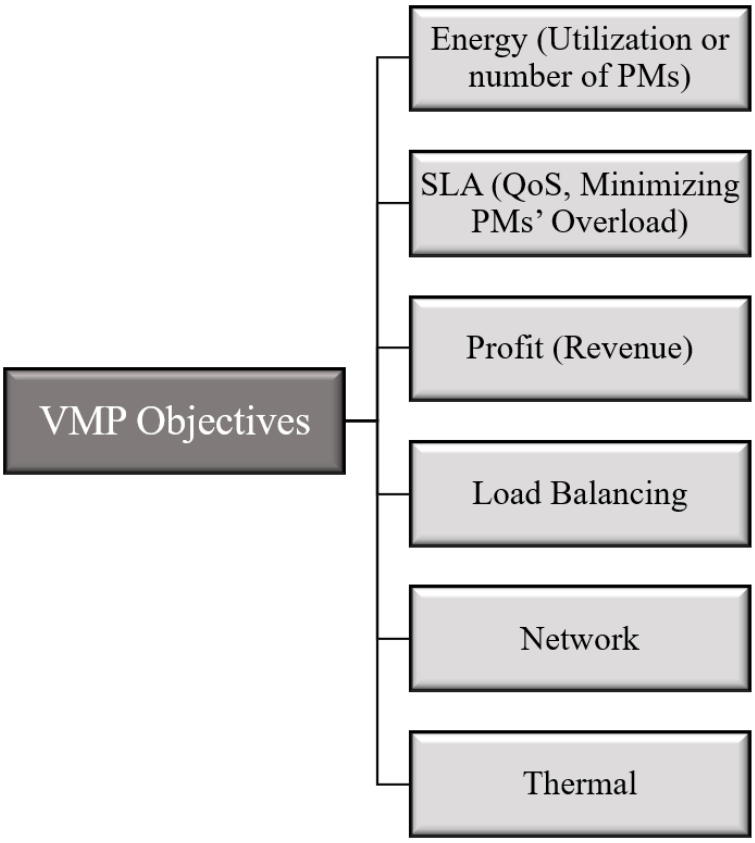



Figure 2.7: High-level classification of VM placement objectives

that applies multiple energy-saving techniques such as soft scaling, DVFS, VM consolidation, and switching idle PMs off. VM consolidation techniques ensure energy efficiency by improving the utilization of the PMs that eventually minimizes the number of active PMs. Jing *et al*. [XF11] propose a centralized energy-efficient cross-layer dynamic VM placement system that simultaneously saves energy, reduces performance degradation, and prevents thermal hotspots. Many initial VM placement solutions aim to reduce energy consumption and resource wastage [LL+12, FMCB14, CFF14]. Moreover, Dynamic demand-based VM placement makes use of VM consolidation to minimize the number of running PMs and hence reduces energy consumption [BAB12, MR16, FAP+15, MP16, DHC16, MM17]. Furthermore, dynamic reservation-based VM placement solutions also consider energy efficiency [LLW11, BS14]. Lucanin *et al*. [LB13] designed a grid-conscious cloud model that is aware of the changes in electricity prices during the day. They analyse real-time electricity prices and optimize energy usage based on the prices. They proposed normal VM instances that should be

available all the time as well as green instances, which can be paused when the electricity prices are high. Finally, they proposed a peak pauser algorithm, which pauses the green instances during the hours of peak electricity prices.

### 2.6.2 SLA and QoS-aware VM Placement

Cloud providers pay particular attention to the SLA to keep the customer satisfied so as not to pay additional penalties due to SLAVs or to lose unsatisfied customers. As an illustration, Amazon loses 1% in sales due to every additional 100ms of latency [GCG+12]. A large proportion of the work in the VM placement literature considers meeting the QoS requirements by minimizing all sources of SLAVs such as overload situations and the aggressive migration of the VMs. The SLA requirements are different among the various cloud service models, which means that there are different methods for defining the SLA for each service model. For example, in the SaaS, the cloud provider might care about minimizing the *makespan*. Makespan is the total time spent from receiving a task for execution until the end of the execution. On the contrary, the way of calculating SLAVs in the IaaS is entirely different, as it neither depends on the applications' response time nor their execution time. The SLA in the IaaS pays extra attention to the availability of the infrastructure. Xiao *et al*. [XSC13] strive to avoid overload situations per the PMs. Examples of SLA-aware VM placement proposals in the IaaS include [CD13, SW12, BAB12, WLFJ13, FAP+15, MP16, ZHL16, MR16].

### 2.6.3 Profit or Revenue-aware VM Placement

A realistic estimation of any VM placement solution in CDCs can be translated into monetary success or failure. Therefore, developing a VM placement solution that can consider the high-level objectives of the cloud provider through profit or revenue is truly useful. Utility functions are of great value in expressing such financial goals, and the utility-based VM placement solution can choose the state that results in higher profit or revenue. Maximizing the revenue or profit can be considered for the initial as well as the dynamic VM placement. For example, Mosa *et al*. [MP16] have developed a profit-aware dynamic VM placement solution that tries to decrease the costs of VM placement to maximize the cloud provider's profit. An example of an initial VM placement solution that is revenue-aware proposed by Shi *et al*. [SBBJ13], where they

try to allocate sets of VMs with different security and anti-colocation constraints. Additionally, Shi *et al*. [SH11] propose a VM placement solution that aims to maximize the profit from the VM placement while maintaining the required SLA and constrained by the power budget.

### 2.6.4 Load Balancing-aware VM Placement

A load-balance-aware VM placement solution can balance the load either inside each PM (intra-PM load balancing) or among the PMs (inter-PM load balancing) [LZLY17]. Intra-PM load balancer balances the load among the different resources types (CPU, Memory, ...) of the same PM. Inter-PM load balancer balances the load among all PMs. Both intra-PM and inter-PM load balancers can be either initial (incremental) or dynamic. Inter-PM load balancing is a critical objective for data centres' operators to avoid hot spots. Hot spots may result from the aggressive consolidation of VMs, which lead to more overload situations. Hot spots violate the SLA, degrade the reliability and availability of the hosted VMs and lessen the lifetime of the PMs [CFF14]. Intra-PM load balancing minimizes resources wastage of each PM. Moreover, balancing the load is essential to ensure the availability and scalability of the CDC [MSY12]. Many static and dynamic VM placement solutions aimed to balance the load among the running PMs [REP+13, WYS13, YYL17, LZLY17].

### 2.6.5 Network-aware VM Placement

VMs can affect the performance of each other due to contention for network bandwidth. Wang *et al*. [WMZ11] proposed an online packing algorithm for VM consolidation considering bandwidth requirements. They formulated the VM consolidation problem as a stochastic bin packing problem and modeled the VMs' bandwidth as probabilistic distributions. Furthermore, they have chosen to place VMs that are intensively communicating with each other in the same or the nearest possible PM for minimizing network overhead [JS14]. Li *et al*. [LLYL15] formulated the multi-tenant VM allocation problem with the goal of reducing the network diameter of all the tenants. They developed an LP-MKP algorithm, which is a layered progressive resource allocation algorithm based on the multiple knapsack problem. Biran *et al*. [BCF+12] propose a network-aware VM placement solution that places CPU, memory in addition to network resources. They formulated a network-aware optimization problem named min cut ratio-aware VM placement (MCRVMP). The MCRVM placement problem is

defined in a data centre with tree-based network topology. The proposed two heuristics for solving the MCRVMP problem, namely 2-phase connected component-based recursive split (2PCCRS) and greedy heuristic (GH). Their solution works well with medium-size CDCs. Ilkhechi *et al.* [IKU15] present a network-aware VM placement solution for a specific scenario where VMs require to exchange intensive network traffic with fixed nodes (called sinks). They aimed to maximize the satisfaction through improving the VMs performance. They introduced an off-line greedy approach and a heuristic-based solution, which assumes that each PM can only host a single VM. Tziritas *et al.* [TLK+18] introduced a communication-aware (aiming to avoid unnecessary network traffic during VM migration) and energy-aware VM placement solution using the graph-coloring algorithm. They migrate VMs in batches as opposed to the current solutions that consider single VM migrations. They managed to optimize network traffic resulting from communication dependencies between VMs, which enabled them to reduce network overheads and energy consumption.

### 2.6.6 Thermal-aware VM Placement

Data centres' PMs, network devices, and other peripherals generate heat, which needs dissipation for better reliability and lower operational costs. Thermal dissipation contributes to the operational costs of the data centres [Kre00]. Thermal-aware VM placement solutions balance the temperature resulting from the PMs. Thermal-aware placement aims to mitigate hot spots and keep the temperature within a range that makes the data centre reliable. Thermal-aware VM placement might contradict with energy-aware VM placement [XF10a]. For example, minimizing the number of running PMs can lead to creating hotspots resulted from heat imbalance. The trade-off among different placement objectives should be taken into account. Many proposals consider thermal-aware placement during workload scheduling in contrast to the limited number in case VM placement. Xu *et al.* [XF10a] developed a thermal-aware VM placement solution that ensures thermal efficiency by making sure that the CPU's temperature is within the safe range.

### 2.6.7 Multiple Objectives-aware VM Placement

In realistic scenarios, an initial or dynamic VM placement solution considers not only a single criterion, but multiple objectives which may be contradictory. Solving a

VM placement problem with multiple objectives can be formulated either as a multi-objective problem or as a single objective that considers all the required objectives in the objective function. For example, Gao *et al*. [GGQ+13] have addressed multiple objectives (resource wastage and power consumption) simultaneously by proposing a multi-objective ant colony solution that tries to find the set of non-dominated solutions that can minimize energy and resource wastage simultaneously. Xu *et al*. [XF10a] solved the VM placement problem using multi-objective optimization aiming to save power consumption and reduce resource waste and thermal costs. Minimizing energy consumption for running the VMs while reducing the resulting SLA violations has been widely considered in the literature(*e.g*. [BB12, WLFJ13, CD13, FAP+15, ZWL+18, LSWJ18]). Zheng *et al*. [ZLD+15] consolidate VMs while considering three conflicting objectives, namely reducing power consumption, balancing the load among the PMs, and minimizing the migration time.

## 2.7 Discussion

Table 2.1 summarizes some of the current VM placement solutions by classifying them based on the type of the problem being solved, the adopted VM placement strategy, policy, and placement objectives. Furthermore, the conducted analytical review resulted in a novel classification that helped in outlining some observations on the current state-of-the-art of VM placement research. These observations describe candidate research areas that can advance the state-of-the-art and guide VM placement research to focus on areas that may have a beneficial impact. Accordingly, the following subsections describe these observations by outlining some of the problems and opportunities in the current VM placement research. These observations are described grouped into three directions namely *exploring new VM placement strategies*, *real-world performance evaluation* of existing solutions, and developing an *evaluation and verification framework*. These observations directed the work done on this thesis.

### 2.7.1 Exploring New VM Placement Strategies

A significant and growing portion of the VM placement literature has investigated reservation-based and demand-based VM placement strategies while solving the initial and dynamic VM placement problems. Such VM placement proposals tended to focus only on the two extremes of the VM placement strategies (reservation-based and

Table 2.1: VM placement solutions based on problem, strategy, policy and objectives

| Research | Problem | Strategy | Policy | Objectives |
|---|---|---|---|---|
| Ma *et al.* [LL+12] | S | r | U (ACO) | E, SLAV & RW |
| Lin *et al.* [LLW11] | D | r | A | E |
| Calcavecchia *et al.* [CBHM12] | S + D | d | A | VMs' DS |
| Beloglazov *et al.* [BB12] | D | d | A | E & SLA |
| Cao *et al.* [CD13] | D | d | A | E and SLA |
| Xiao *et al.* [XSC13] | D | d | A | OA & NPMs |
| Wang *et al.* [WLFJ13] | D | d | A | E & SLA |
| Rabbani *et al.* [REP+13] | S | r | A | LB & CC |
| Quang-Hung *et al.* [QHNN+13] | S | r | U (GA) | E |
| Shi *et al.* [SBBJ13] | S | r | UA | Revenue |
| Wuhib *et al.* [WYS13] | D | r | A | E, LB, FA & SD |
| Farahnakian *et al.* [FPLP13] | D | d | A | E & SLA |
| Ferdaus *et al.* [FMCB14] | S | r | U (ACO) | E & RW |
| Corradi *et al.* [CFF14] | S | r | A | E |
| Borgetto *et al.* [BS14] | D | r | A | E |
| Deng *et al.* [DHC16] | D | d | A | E & SLA |
| Ye *et al.* [YYL17] | S | r | U (EEKnEA) | E, LB, Ut & Ro |
| Zhao *et al.* [ZWL+18] | S | r | U (ACO) | E & P |
| Ilkhechi *et al.* [IKU15] | S | r | A | NS |
| Farahnakian *et al.* [FAP+15] | D | d | U (ACS) | E & SLA |
| Chowdhury *et al.* [CMR15] | D | d | A | E & S |
| Zhou *et al.* [ZHL16] | D | d | A | E & SLA |
| Monil *et al.* [MR16] | D | d | A | E & SLA |
| Khoshkholghi *et al.* [KDA+17] | D | d | A | E & S |
| Monil *et al.* [MM17] | D | d | A | E & SLA |
| Li *et al.* [LZLY17] | D | d | U (BBO) | LB |
| Mosa *et al.* [MP16] | D | d | U (GA) | Profit |
| Wang *et al.* [WT18] | D | d | A | E & S |
| Yousefipour *et al.* [YRJ] | S | r | U (GA) | E & cost |
| Liu *et al.* [LSWJ18] | D | d | A | E & SLA |
| Nguyen *et al.* [NFYJ18] | D | d | A | E & SLA |
| Mosa *et al.* [MS19] | D | d | U (GA) | UU and OU |

- Static or initial (S), Dynamic (D), Reservation (r), Demand (d), Utility-based (U), Action-based (A), Utility and Action (UA), Ant Colony optimization (ACO), Genetic Algorithm (GA), Biogeography-Based optimization (BBO), Energy (E), Service Level Agreement (SLA), SLA Violation (SLAV), Demand Satisfaction (DS), Load Balancing (LB), Utilization (Ut), Robustness (Ro), Performance (P), Network Satisfaction (NS), Communication Costs (CC), Overload Avoidance (OA), Number of PMs (NPMs), Fair Allocation (FA), Service Differentiation (SD), Resource Wastage (RW), underutilization (UU), overutilization (OU).

- In the above table, any solution to the dynamic (D) VM placement problem means that it has considered the initial (static) VM placement problem as well.

demand-based strategies). A reservation-based VM placement can satisfy the SLA requirements at the cost of underutilization of the PMs, which means more energy waste. A demand-based VM placement strategy results in more efficient utilization, but leads to more SLAVs (See the *strategy* column of Table 2.1). To the best of our knowledge, so far, there have been no attempts made to study the middle way between allocating the full reservation and allocating the actual demand. We think that exploring the area between reservation-based placement and demand-based placement could offer various alternative tradeoffs and contribute to better flexibility in the decision-making for cloud providers or private cloud owners. Therefore, the focus of this thesis is to develop a new VM placement solution that makes use of a novel VM placement strategy. That novel VM placement strategy explores the area in-between full-reservation and the actual demand-based allocation.

### 2.7.2 Real-world Performance Evaluation of Existing Solutions

A considerable amount of literature has proposed numerous solutions for both the initial and dynamic VM placement using different policies. The academic literature on VM placement solutions has revealed repetitions and duplications, and many of them are just a reinvention of the wheel. Therefore, conducting experiments that evaluate the existing solutions using a real private cloud data centre or a collaborative research cloud data centre (in collaboration with other institutions) can add significant value to the VM placement research. By such in-vivo experiments, researchers can measure the actual effectiveness of current solutions in practical scenarios and even discover restrictions of applying others. Furthermore, conducting these in-vivo experiments can help researchers conduct a realistic cost-benefit analysis regarding the adoption of a particular VM placement policy as opposed to others.

### 2.7.3 Developing an Evaluation and Verification Framework

If the access to real CDCs to implement VM placement solutions is not possible, then it is vital to develop a component-based evaluation and verification framework that can systematically verify existing solutions. Researchers usually develop and validate their solutions in various simulations environments that are incomparable. A functional area of research is to build an adequate evaluation and verification framework that uses realistic workloads so that it can easily compare alternative VM placement proposals with the same set of practical settings. Additionally, the framework can be offered

through a web-based system linked to a simulated or a real CDC so that researchers only upload their algorithms and run them. Through such a framework, researchers can easily plug their new VM placement solutions, which are evaluated and tested against practical use cases.

## 2.8 Summary

This chapter has introduced a novel classification of the existing VM placement research in four areas, namely VM placement problems, strategies, policies, and objectives. This novel classification and analysis can direct the future researcher to concentrate on areas of higher impact. This chapter started by reviewing the initial (both online and offline versions) and dynamic VM placement problems. Online solutions for the initial VM placement reflect the real scenarios on CDCs; however, dynamic VM placement solutions are usually offline. The review confirms that using two levels (cluster and PM) during the VM placement yields more robust results than only considering one level. Then, the chapter described the basis on which the VMs are reallocated using either the reservation-based or the demand-based VM placement strategies and reviewed the related articles. The demand-based VM placement solution can be reactive, proactive, or hybrid. After that, the chapter has covered the three standard policies adopted for creating self-managing VM placement solutions. It is observed that existing VM placement solutions in the literature are either action-based or utility-based. The prohibitive SLAVs and underutilization costs are roadblocks to consolidation ratios from demand-based and reservation-based VM placement strategies, respectively. Regardless of the VM placement strategy or the policy used, any VM placement solution can consider either a single- or multi-objective (*e.g.* energy, utilization, SLA, profit, load balancing, temperature, or a combination of them). Finally, the chapter shows the lessons learned, and some points out some possible future directions.

To conclude, we think that proposing new solutions for either the initial or dynamic VM placement, using either reservation or demand-based strategies, may not have a high impact. The current VM placement research has already covered dozens of the action-based and utility-based solutions with many repetitions of the deployed algorithms. Therefore, we think that to conduct VM placement research with a high impact, it needs either to be conducted in real cloud environments or through collaboration with cloud providers. If this is not possible, then new cloud evaluation frameworks

that reflect the real nature of the cloud environment are essential and can have a significant impact by measuring the effectiveness of the existing solutions in a standard practical way. Finally, another direction that worth further investigation is exploring the area between reservation-based and demand-based VM placement strategies, which we have decided to be the focus of this thesis.

# Chapter 3

# System Architecture and Simulation Environment

## 3.1 Introduction

This chapter aims to illustrate the principal components of the system architecture of the VM placement solution. This architecture represents the design foundation of the proposed parameter-based VM placement solution. Then, it describes the essential characteristics of the workload running on the VMs and any design assumptions. Due to the limitations of conducting in-vivo (real) experiments, the chapter reviews the most widely used cloud simulation toolkits. Then, it demonstrates in detail the one that has been adopted for conducting the empirical evaluation (CloudSim). Eventually, the chapter outlines and demonstrates the calculation of the performance metrics that are used to measure the performance of the proposed algorithms.

## 3.2 Main Components of the System Architecture

Figure 3.1 exhibits an abstract view of the system architecture and the main components of the proposed parameter-based VM placement solution for cloud data centres (CDCs). The main components are the *CDC*, the *customer requests* (in the form of VM placement requests) and the *VM placement controller* which consists of *initial VM placement controller*, *dynamic VM placement controller* and *parameter estimation*.

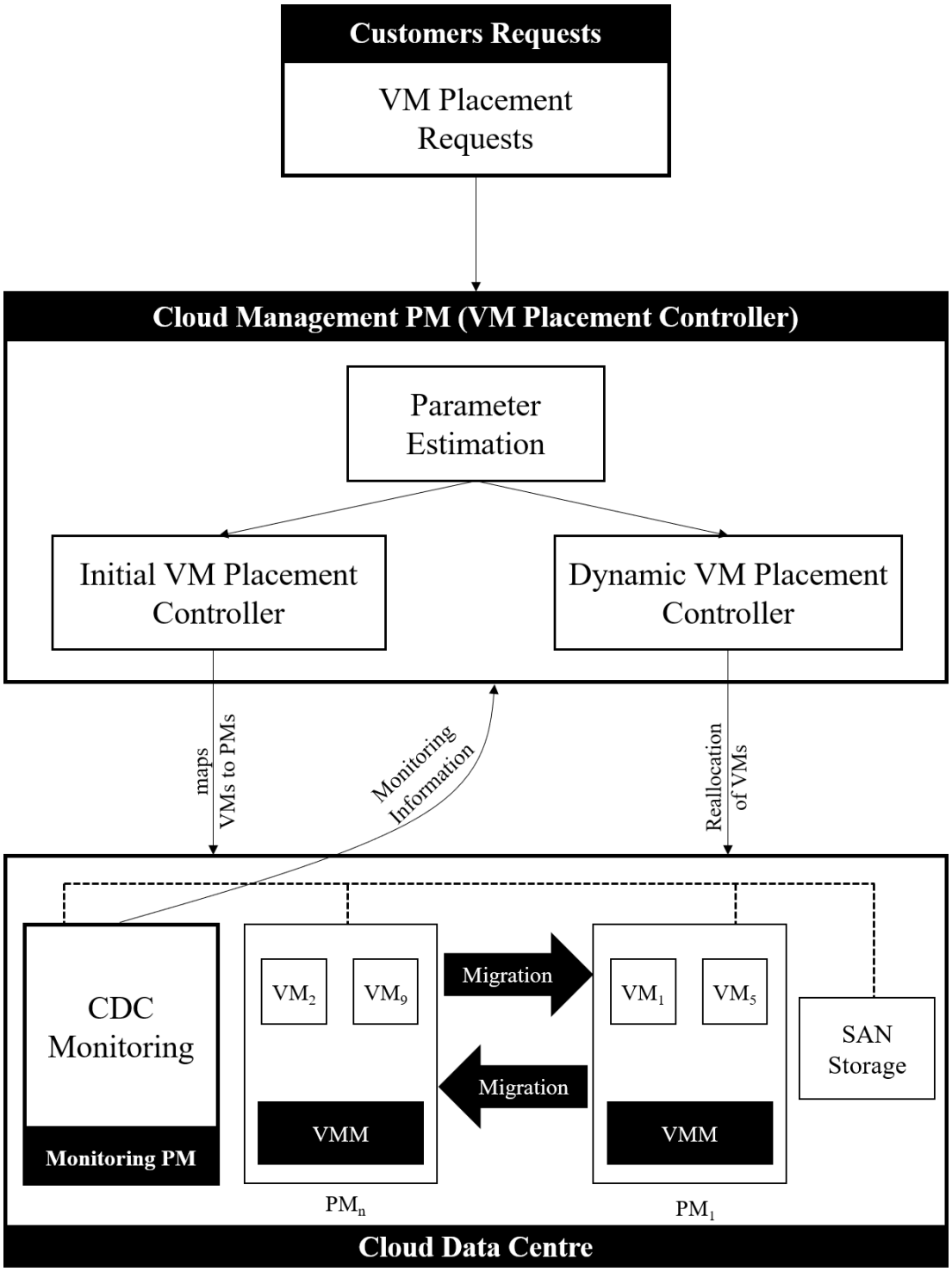



Figure 3.1: Main components of the parameter-based VM placement architecture

### 3.2.1 Cloud Data Centre

The CDC hosts many PMs, storage, network devices, and other infrastructure elements that make it function accurately, such as power sources and cooling systems. The CDC

hosts three types of PMs namely, *compute*, *monitoring*, and *management*. A compute PM is a PM that performs the computational works and hosts VMs; each hosted VM is running on one of the compute PMs. Big cloud providers (or even private cloud owners) usually manage large-scale CDCs that consist of hundreds of thousands of compute PMs. For example, in May 2013, Netcraft[1] reported that the number of PMs in the US East (Northern Virginia) region of AWS EC2 is 93,537 with a growth rate of 7.4% over the last four months. Both the initial and the dynamic VM placement solution require runtime monitoring of the resource utilization of the CDC's PMs' as well as the hosted VMs. Therefore, the cloud monitoring component runs a monitoring software to collect monitoring information of CDC's devices and send it to other components that require it. Examples of the monitoring information collected by the CDC monitoring component include the resource utilization of both the VMs and the PMs, the current SLAVs, the health of PMs and VMs, and the energy consumption. The monitoring information is collected at each scheduling interval. Prometheus[2] is an example of a powerful open-source cloud-native monitoring and alerting toolkit that can generate ad-hoc graphs and alerts. Finally, a management PM is the one that provides many management functions such as controlling the compute PMs, VMs, users' authentication, VM images, storage, and network. In this thesis, we are only concerned with one management function, which is the autonomic placement of VMs to PMs. All the PMs are connected through a hierarchical network architecture; however, we ignore the network details as it is out of the scope of this work. In what follows, we will use the term PM to refer to a compute PM (*a.k.a* host).

The target CDC is assumed to consist of $N$ heterogeneous PMs; each PM has $n$ CPU cores, and each CPU core is defined by the total number of MIPS $x$ (million instructions per second), so the total processing capacity of a single PM is $nx$. To map a VM to a PM, the capacity of the required virtual CPU core should be less than or equal to the available capacity of one of the physical CPU cores. Additionally, all PMs in the CDC support resource *oversubscription*, which means that the resource capacity of all VMs hosted in a specific PM might exceed the capacity of that PM. Each PM is also characterized by other resource types, such as the amount of memory and the available network bandwidth. The CDC runs a virtual machine manager (VMM) that manages the allocation of VMs and enables the live migration of the VMs among the running PMs. The CDC stores the VMs' images and the data on a storage area network

[1] https://news.netcraft.com/archives/2013/05/20/amazon-web-services-growth-unrelenting.html
[2] https://prometheus.io/

(SAN). Moreover, it is assumed that the cloud provider can offer various VM instances of different sizes based on the amount of resources allocated for each VM type. As an illustration, examples of VMs' instances might be extra-large, large, medium, small, and extra small. This architecture assumes that the cloud provider or the private cloud owner has no prior knowledge of the type and the resource usage of the applications running on the customers' VMs. Therefore, this research considers an IaaS service model, which is application-agnostic.

### 3.2.2 VM Placement Requests

The IaaS cloud customers submit requests to lease VMs of a predefined size for a particular time. Each VM placement request might consist of either a single VM or multiple VMs to be hosted by the cloud provider's infrastructure (this also applies to private cloud scenario). The VM placement request specifies the required size of the VM(s) based on the available VMs' instances. VM placement requests are made online and continuously received by the cloud providers. Each VM placement request is initially being assessed by the initial VM placement controller, which decides whether to accept or reject the VM(s) placement request based on resource availability or any other further constraints.

### 3.2.3 VM Placement Controller

The VM placement controller automatically manages the initial placement and the reallocation of VMs in the CDC. A typical VM placement controller should reflect the non-deterministic behaviour of the system. As an illustration, the VM placement requests should be represented as stochastic processes, so that these requests arrive with some probability. This probability can be obtained from solving a stochastic optimization problem. However, constructing a stochastic model for the components of the VM placement controller is not a straightforward task. For example, the utilization of VMs in CDCs is complex and cannot be easily modelled using simplistic statistical distributions. Moreover, simulating the non-deterministic behaviour may not guarantee that the solution will be optimal for a particular case [Bel13]. Therefore, the behaviour of the proposed parameter-based VM placement solution is deterministic due to the complexity and other issues associated with reflecting the non-deterministic behaviour of cloud environments. As illustrated in Figure 3.1, the VM placement controller consists of three components, namely the initial VM placement controller, dynamic VM

controller, and parameter estimation. The details of these components are explained in the following subsections.

### Initial VM Placement Controller

The initial VM placement controller receives monitoring information (from the CDC monitoring component) about the PMs and the VMs in the CDC, as shown in Figure 3.2. If the initial VM placement controller accepts a VM placement request, then it starts mapping the VM(s) to the proper PM(s) based on the adopted initial VM placement solution. In practical scenarios, the initial VM placement controller solves an online optimization problem as the cloud provider continuously receives new VM placement requests over time. The adopted virtualization technology, through the virtual machine manager (VMM), allocates the VMs' resources on the specified PM(s) according to the mapping decision of the initial VM placement controller.

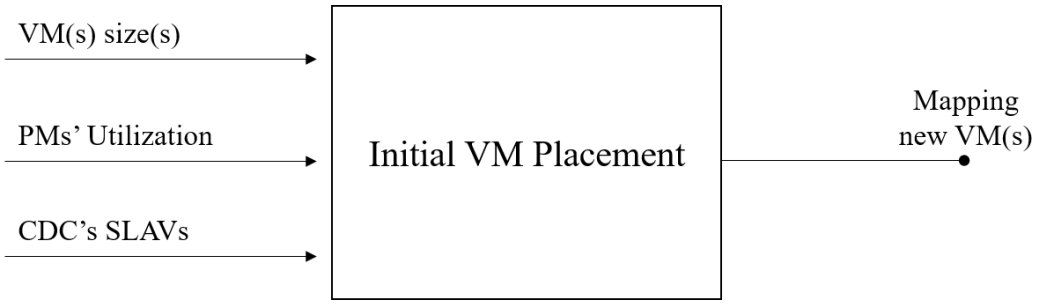



Figure 3.2: Initial VM placement controller

### Dynamic VM Placement Controller

The dynamic VM placement controller reallocates the VMs to improve the current state of the system based on the specified VM placement objectives and constraints. Figure 3.3 indicates that the dynamic VM placement controller uses the current VMs-to-PMs mapping in addition to other monitoring information to create a new VMs-to-PMs mapping that may result in a better state of the system. A key question is: How often does the VM placement controller reallocate the VMs? The answer to this question depends on the adopted triggering mechanism. The triggering mechanisms may be periodic, event-driven, hybrid, or threshold-based [PS16]. In the proposed architecture, the dynamic VM placement controller runs periodically at predefined scheduling intervals (every five minutes of the simulated time).

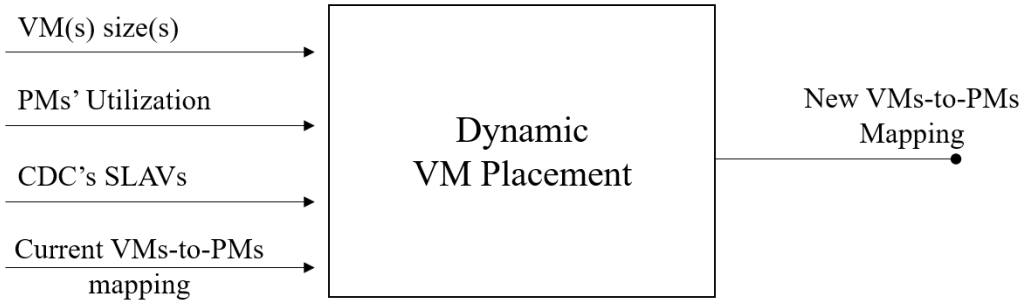



Figure 3.3: Dynamic VM placement controller

During each scheduling interval (and by making use of the monitoring information), the dynamic VM placement reallocates the VMs from the set of *source PMs* to the set of the *target PMs*. A source PM is the one that is the source of migration (requires migrating some or all of the VMs that it hosts). As an illustration, a PM might migrate some of the VMs when it is overutilized, or it might migrate all the hosted VMs when it is underutilized (this is an example of a source PM). On the contrary, a target PM is the one that can accept migrating VMs or can host new VM placement requests. A target PM is a candidate for hosting VMs migrated from source PMs. Target PMs should have sufficient resources and should not be overutilized after hosting the migrated VMs. Solutions of the dynamic VM placement controller make use of offline algorithms as it originally has all the current VMs in the CDC in advance and should redistribute them according to the adopted VM placement strategy, policy, and objectives. Figure 3.3 shows the input to the dynamic VM placement controller, where the current VMs-to-PMs mapping is one of the inputs besides other monitoring information.

**Parameter Estimation**

The parameter estimation component is the final component of the proposed parameter-based VM placement system architecture (shown in Figure 3.1). Parameter estimation is a new component of the VM placement architecture which works only for the proposed parameter-based VM placement solution. Both the initial and dynamic VM placement controllers can make use of the parameter. The parameter can be either static (set by the cloud operator) or dynamic (estimated automatically using various parameter estimation algorithms). Chapter 4 introduces the parameter-based VM placement solution using a predefined static parameter. Furthermore, Chapters (5, 6) propose different algorithms for the dynamic estimation of the parameter based on the monitoring

information received from the CDC monitoring component. The algorithm design and the implementation details of the initial and dynamic VM placement controllers as well as various parameter estimation algorithms are presented on Chapters (4, 5, 6).

## 3.3 Simulation of Cloud Data Centre

Recall that the thesis is about the development and evaluation of a new VM placement solution that adopts a novel parameter-based VM placement strategy. The development and evaluation of this new VM placement solutions is accomplished using either *real-world* or *simulated* experiments. Real-world experiments (*a.k.a.* in-vivo) are conducted using the infrastructure of a real CDC. By and large, the real-world experiments are usually expensive, time-consuming, and maybe tedious to perform [BVWS14]. Conducting experiments in cloud infrastructure based on real CDC is usually not possible for academic researchers as the majority of researchers in academic institutions do not have access to real cloud data centres. Even if researchers managed to access the real CDCs to conduct their experiments, they would usually face two common problems, namely scalability and reproducibility. Scalability problem means that the size of the experiment is limited to the current network design, compute PMs, storage, and network devices of the accessed CDC. The reproducibility problem results from the difficulty of reproducing the results by having to build the same real CDC that is not simple to reproduce. Moreover, testing specific non-common scenarios in real CDC may be difficult and require too much time; for example, testing specific workload patterns or network conditions.

Modelling and simulation technologies can model and simulate real complex environments using either software-only or a combination of software and hardware in case of emulation [ZPXD12]. Simulating cloud computing environments can make the testing and evaluation process easier, cost-effective, and reproducible. Moreover, it can simulate large-scale CDCs without much pain. This means that simulated environments can overcome the reproducibility and scalability problems of real-world cloud experiments. Researchers have been using simulation tools extensively in computer networks, grid, and distributed systems. Examples of research-focused computer

networks' simulators include NS-3[3], OPNET[4] and OMNeT++[5]. Additionally, examples of simulators for simulating large-scale distributed systems include GridSim[6] and SimGrid[7]. Finally, there are many existing cloud simulation tools for simulating cloud computing environments such as CloudSim [CRB+11a], GreenCloud [Uni10], CloudSched [TZX+15], EMUSIM [CNDRB13], DCSim [TKBL12], MDCSim [LSN+09], and iCanCloud [NnVPC+12].

Due to the previously stated advantages of using simulation toolkits, this thesis makes use of a cloud simulation tool for the development and evaluation of the proposed VM placement solution. However, typical questions that arise from using simulated environments include: shall researchers develop their own simulators or make use of existing ones, and which one of the current simulators should they use and why? The following subsection briefly describes the commonly used cloud simulation tools and our choice of the simulation toolkit for the implementation and evaluation of the proposed VM placement solution.

### 3.3.1 Common Cloud Simulators

This subsection examines three of the commonly used cloud simulation toolkits, namely CloudSim, GreenCloud, and iCanCloud. Then, Section 3.3.2 justifies the choice of the specific cloud simulator based on three criteria.

CloudSim[8] is a framework for modelling and simulating CDCs and cloud services for various service models. It is an event-driven open-source simulation toolkit programmed in Java and developed by the CLOUDS Laboratory at the University of Melbourne. CloudSim can simulate energy-aware computing resources in large-scale CDCs. Additionally, it can simulate network topologies and allow live migration of VMs for supporting the dynamic reallocation of the VMs [CRDB09, GSA12]. Furthermore, it supports simulating federated clouds and allows users to develop and test their allocation policies. CloudSim does not have a graphical user interface (GUI); however, the CloudAnalyst extension supports a GUI. Several new projects add extensions to the core CloudSim to enable the simulation of new environments or to add

[3]https://www.nsnam.org/ [Accessed: 23-Jan-2018]
[4]https://www.riverbed.com/gb/products/steelcentral/opnet.html [Accessed: 23-Jan-2018]
[5]https://omnetpp.org/ [Accessed: 23-Jan-2018]
[6]http://www.buyya.com/gridsim/ [Accessed: 23-Jan-2018]
[7]http://simgrid.gforge.inria.fr/ [Accessed: 23-Jan-2018]
[8]http://www.cloudbus.org/cloudsim/ [Accessed: 23-Jan-2018]

new functionality. As an illustration, iFogSim[9] enables simulating resource management and scheduling policies in Fog computing. CloudSimEx[10] extends CloudSim by adding more utilities, allow running experiments in parallel and simulating MapReduce. EdgeCloudSim[11] simulates Edge computing scenarios. More further extensions of CloudSim include WorkflowSim, Cloud2Sim, SimpleWorkflow, DynamicCloudSim, RealCloudSim, CloudReports, CloudAuction, CloudMIG Xpress, and CloudAnalyst. These various extensions and the common use of CloudSim makes it one of the most widely used cloud simulators in academic research.

GreenCloud [Uni10] can be a handy alternative cloud simulator when the experiments focus on the communication aspects of the cloud environment. GreenCloud is an open-source packet-level cloud simulation toolkit that models and simulates energy-aware cloud data centres by extending the NS-2 network simulator. GreenCloud is programmed using C++ and Tool Command Language (TCL) scripts. GreenCloud pays more attention to communication networks in the cloud. Additionally, it models energy consumed by the data centre's entities, including servers, switches, and communication links. It simulates different computing resources, including CPU, memory, and network with various energy models for each of these resources. Besides, it supports VM migration in addition to a simple graphical user interface (GUI). Cloud researchers can use GreenCloud to develop their algorithms for resource allocation, workload scheduling, and to optimize network communication. GreenCloud is pretty powerful for simulating network-aware resource scheduling algorithms, and it considers a three-tier hierarchical network architecture of the CDC. To ease the deployment and usage, the GreenCloud is also available as a standalone VM so that it can be easily used and redistributed without requiring any additional configurations. Finally, GreenCloud provides a dashboard that presents graphs and statistics of the simulation results.

Finally, iCanCloud[12] is another open-source C++ based scalable cloud simulation toolkit. The underlying platform for the iCanCloud simulator is OMNET and MPI [NnVPC+12]. It has built-in methods to compute energy consumption for each computing resource in the cloud data centre. Like other simulators, users can build and test their allocation policies and find the trade-off between cost and performance. iCanCloud supports parallel experiments and provides models for public cloud providers such as Amazon. Moreover, it has a full GUI for managing and configuring PMs,

[9] https://github.com/Cloudslab/iFogSim
[10] https://github.com/Cloudslab/CloudSimEx
[11] https://github.com/CagataySonmez/EdgeCloudSim
[12] http://www.arcos.inf.uc3m.es/ icancloud/Home.html [Accessed: 23-Jan-2018]

VMs, experiments, and generating graphical results. However, the downside of iCanCloud is that it does not support VM migration algorithms at the moment.

### 3.3.2 Choosing the Proper Simulation Toolkit

Tian *et al.* [TXC+15] compared four cloud simulators, namely, CloudSim, GreenCloud, iCanCloud, and CloudSched. Their comparison concluded that none of these simulators is perfect for all scenarios. Additionally, the choice of the appropriate simulator is based on the problem type and the optimization objectives. So far, no cloud simulation toolkit outperforms all the others in all aspects, so it is recommended to choose the one that better suits the experimental requirements of the problem that is being solved. This thesis adopts CloudSim for the implementation and empirical evaluation of the proposed VM placement solution for the following reasons:

- **Ease of reproducibility:**
  Reproducible research is highly valuable as it enables others to verify the results and conclusions. Commonly, using popular open-source cloud simulators is better than creating a customized simulation tool regarding the reproducibility of the results. By using open-source cloud simulation toolkits, other researchers may be able to replicate the conducted experiments and reproduce the results.

- **Widely used and facilitates comparison and evaluation:**
  CloudSim simulation toolkit is one of the most widely used cloud simulators by the research community in the area of VM placement and resource management in cloud environments [MS17a]. Some examples of research conducted with the help of CloudSim for workload scheduling and VM placement research areas include [CRB11b, BB12, WGB11, TCTB11, BB10, QHNN+13, MP16, MS17a]. Using a widely used cloud simulator can facilitate the evaluation, verification, and comparison tasks. Furthermore, CloudSim comes with the prepacked real dataset (PlanetLab) besides the ability to generate random traces. Moreover, CloudSim has different test drivers representing different solutions using the available dataset for the VM placement.

The primary weakness point of CloudSim is related to the limitations and inaccuracy of its communications models [NnVPC+12]. Because communication models are out of the scope of this work, such limitations and restrictions are insignificant for the conducted research. One final reason for using CloudSim is experience reuse as

CloudSim has many extensions for containers, edge, and fog computing (it acts as a Swiss knife for multiple jobs). The following subsection demonstrates the architecture of the CloudSim simulation toolkit.

### 3.3.3 CloudSim Architecture

The CloudSim toolkit follows a layered architecture consisting of three main layers namely, *core simulation engine*, *CloudSim* and *user code*. Figure 3.4 (adapted from [CRB+11a]) exhibits the layered architecture of the CloudSim simulator.

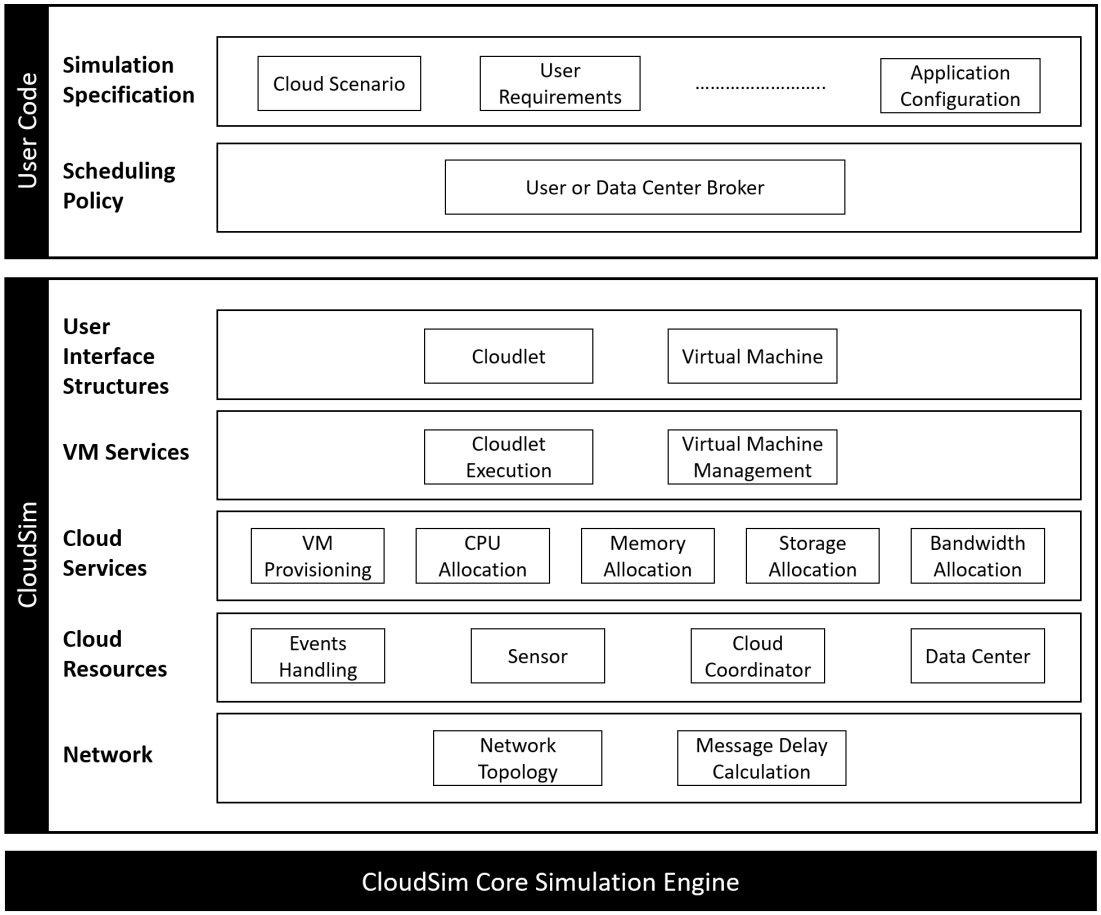



Figure 3.4: CloudSim layered architecture, adapted from [CRB+11a]

The core simulation engine (the bottom layer of the architecture) supports core simulation functionalities, including the creation of different entities (data centres, PMs, VMs, brokers, Cloudlets), queuing, events' processing, managing, and communication among entities. The CloudSim layer (the middle layer) is responsible for modelling

and simulating the virtualized cloud data centre. This layer can define all network-related functions such as network topology and the calculation of message delay. The CloudSim layer configures the data centre, VMs, brokers, Cloudlets, and allocates the required resources such as CPU, memory, storage, and bandwidth for both the VMs and PMs. The Cloud Services sublayer (of the CloudSim layer) is the most critical layer to the context of this work. For example, the VM provisioning component in the Cloud Services sublayer involves allocating VMs to the PMs using the appropriate VM placement strategies and policies. To develop our proposed parameter-based VM placement solution, we have extended the basic functionality of the VM provisioning by implementing the required algorithms. Finally, the user code layer (the top layer of the CloudSim architecture) enables CloudSim users to specify the configuration of the desired cloud data centre. For example, it enables CloudSim users to determine the number of data centres, the number of PMs, the number of VMs, and the applications' workload represented by the Cloudlets. Therefore, the user code layer enables CloudSim users to test different scenarios which involve specific application requirements as well as various data centre settings.

## 3.4 Settings of the Simulated Cloud Data Centre

The proposed parameter-based VM placement solution is implemented and evaluated using a single CDC. The specific settings of the simulated CDC, including number and types of VMs and PMs, are described in what follows. The CPU of each PM or VM in the CDC is defined by the number of cores; the capacity of each core is defined in million instructions per second (MIPS). Tables 3.1 and 3.2 illustrates the characteristics and the types of the PMs and VMs used in the simulated CDC. In Tables 3.1 and 3.2, memory is defined in Megabit (MB), and bandwidth in Megabit per second (Mbps). The settings of the PMs are the default ones that are embedded in CloudSim. The energy consumption is estimated based on these exact types of PMs. The number of VMs and PMs in the CDC depends on the type of the used cloud traces (*e.g.* synthetic or real PlanetLab traces). The number of PMs is 800, and the number of VMs is usually more than 1000 VMs. The resource requirements of VMs varies over time due to the changes in the workload demand of the applications of each single cloud customer. The cloud provider aims to utilize the PMs' resources efficiently while satisfying the SLA requirements. The details of the workload traces are demonstrated in the upcoming section.

Table 3.1: PM Types

| PM Type | Number of CPU cores | Core Capacity | Memory(MB) | Bandwidth(Mbps) |
|---|---|---|---|---|
| HP ProLiant G4 | 2 | 1860 | 4096 | 1024 |
| HP ProLiant G5 | 2 | 2660 | 4096 | 1024 |

Table 3.2: VM Types

| VM Type | Number of CPU cores | Core Capacity | Memory(MB) | Bandwidth(Mbps) |
|---|---|---|---|---|
| High-CPU Medium | 1 | 2500 | 870 | 100 |
| Large | 1 | 2000 | 1740 | 100 |
| Small | 1 | 1000 | 1740 | 100 |
| Micro | 1 | 500 | 613 | 100 |

## 3.5 Workload Traces

This section describes the workload traces that represent the resources requirements of the applications on the running VMs. Different workload traces are crucial for evaluating the proposed algorithms to have a better understanding of their effectiveness. The number of publicly available workload traces is limited, and even the currently available traces hide some details for privacy concerns (for example, using relative values of the actual sizes of the servers and the VMs rather than the real ones). PlanetLab traces are one of the most commonly used real cloud traces, and they are embedded in CloudSim. What we care about in such traces is the resource utilization (CPU, memory and bandwidth utilization) of the VMs or the PMs. Due to limitations of the current real traces, this thesis also makes use of synthetic workload traces. The following subsection describes the characteristics of both the real and the synthetic workload traces in more detail.

### 3.5.1 Normal PlanetLab Traces

The first source of real workload traces is from the CoMon project[13], which has been used to generate monitoring statistics from the PlanetLab[14] platform. The workload data represent CPU utilization recorded every five minutes and collected in the period between March and April 2011 from over a thousand VMs that are hosted by PMs located at different places all over the world. These normal PlanetLab traces are already embedded with CloudSim and available for public use. Figure 3.5 exhibits the

[13]http://comon.cs.princeton.edu/
[14]https://www.planet-lab.org/

Gaussian kernel density estimate (KDE) of the CPU utilization for a total of 1033 VMs on the $20^{th}$ of April 2011. As can been seen from Figure 3.5, the average utilization of PlanetLab traces is low (less than 13%), which confirms the fact that servers are underutilized most of the time. As the average CPU utilization of all VMs does not exhibit the change in workload over time, so Figure 3.6 shows the change of the CPU utilization of two selected PlanetLab machines over the day. Figure 3.6(a) reveals that the workload is fluctuating over the course of the day between 0% utilization to almost 100% utilization. On the other hand, Figure 3.6(b) indicates that VM 2 is entirely idle (as the CPU utilization of VM 2 is zero) for more than eight hours.

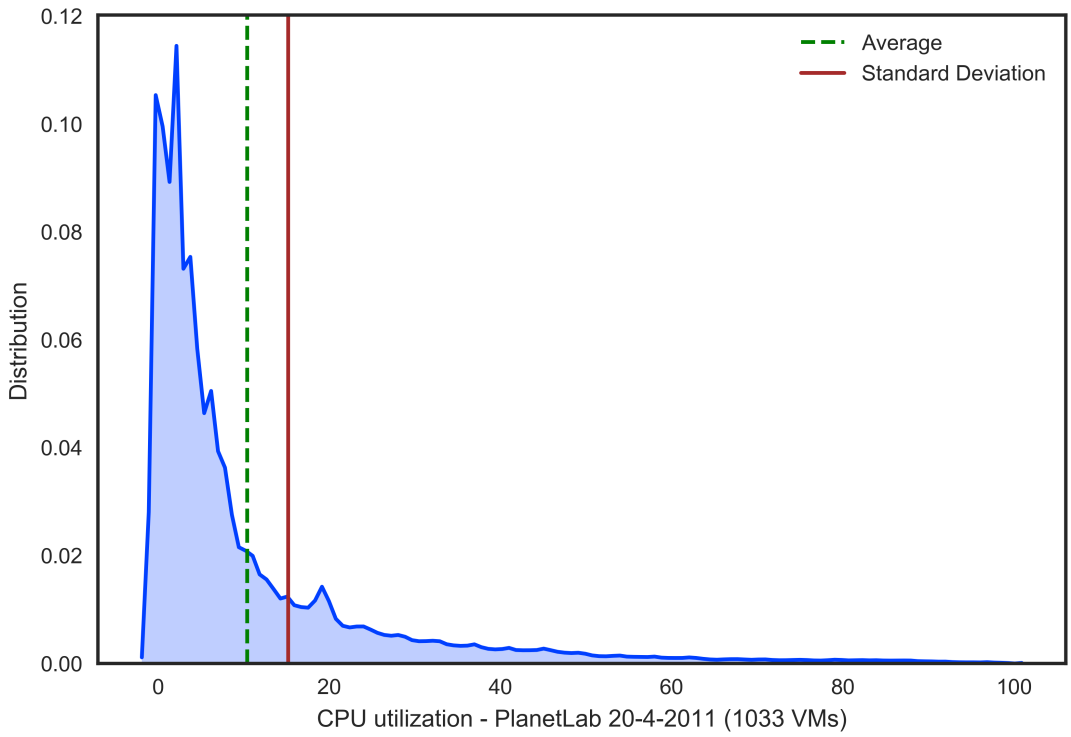



Figure 3.5: Distribution of normal PlanetLab VMs' utilization

### 3.5.2 Skewed PlanetLab Traces

Because of the low utilization nature of the real normal PlanetLab traces, we have created another skewed version of these traces. With this skewed version of the normal PlanetLab traces, the aim is to have higher utilization of the VMs so that we test the behavior of the proposed algorithms against VMs that are low and highly utilized. The skewed PlanetPlab traces skews the original PlanetLab traces by multiplying the values

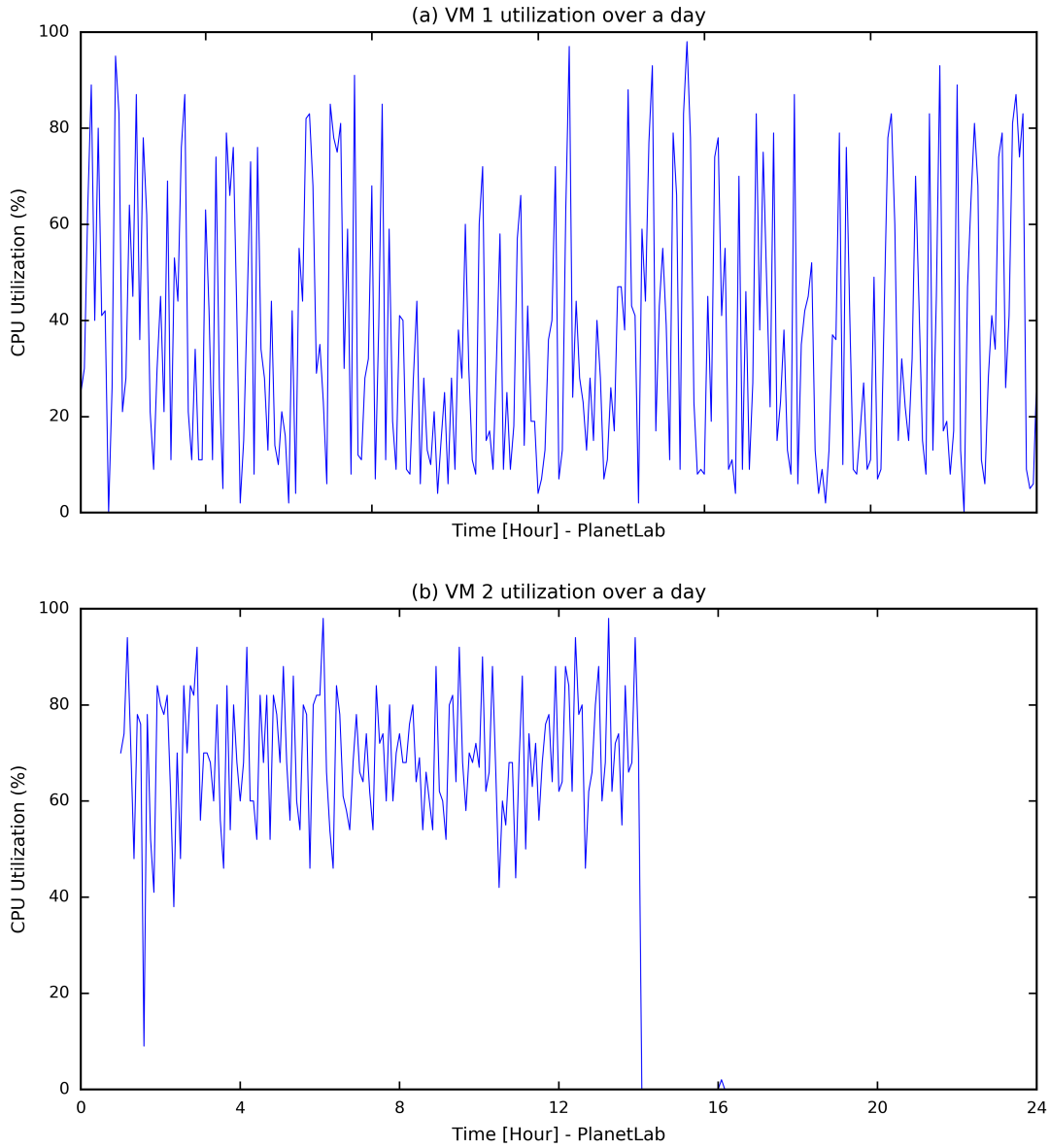



Figure 3.6: CPU utilization of two selected PlanetLab machines over a day

of CPU utilization by five bounded by the VM size. Because we are multiplying by a fixed number, this means that VMs with 0% utilizations are going to have the same utilization percentage in both the normal and skewed versions of PlanetLab traces.

### 3.5.3 Synthetic (Random) Traces

CloudSim offers a way to generate synthetic (random) workload traces for testing various VM placement solutions. The synthetic workload traces represent artificial CPU

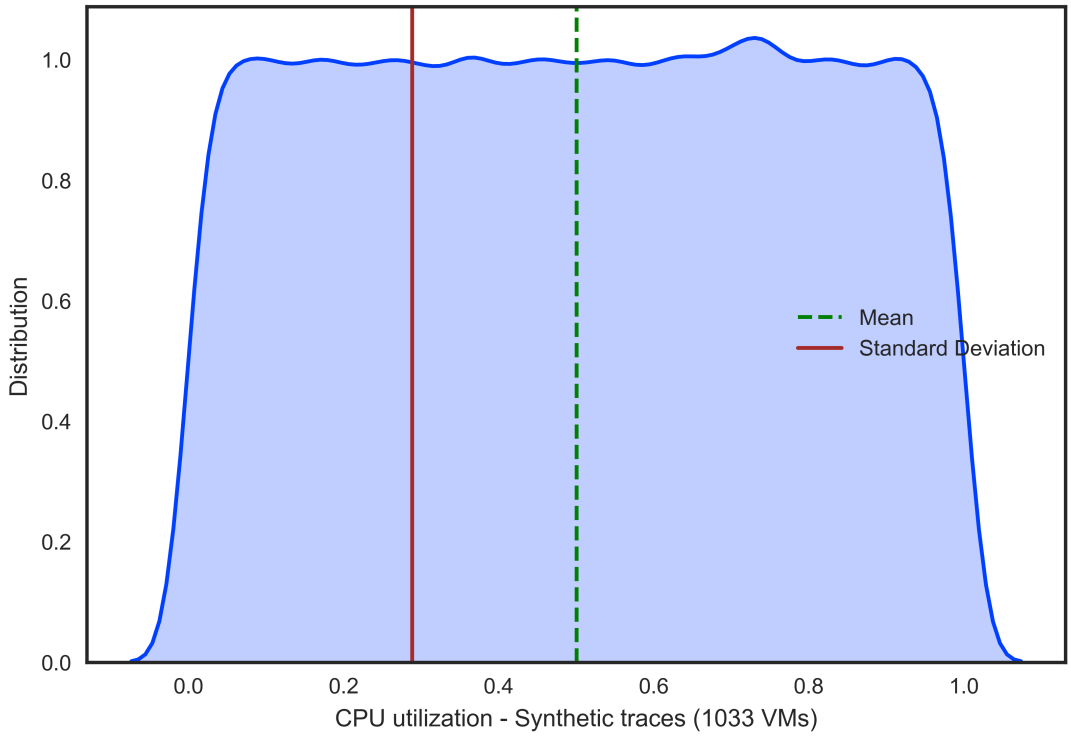



Figure 3.7: Distribution of the VMs' utilization using the synthetic traces

utilization values (with a sampling rate of five minutes over a day) that have been generated based on a uniform random distribution with values in the range between zero and one. These workload traces represent the CPU utilization of the tasks or applications running on each VM. Figure 3.7 presents the Gaussian kernel density estimate (KDE) of the synthetic the CPU utilization of the 1033 VMs. The average utilization of the synthetic traces is higher than the average utilization of the normal PlanetLab traces (50% as opposed to 13%).

## 3.6 Performance Metrics

1. **Energy consumption:**
   The total power consumption of a PM is determined by the utilization of various resource types, including the CPU, memory, storage, and cooling system. However, several studies indicate that there is a linear relationship between energy consumption and CPU utilization of the PMs [FWB07]. Moreover, the linear relationship between energy consumption and CPU utilization exists even

when the CPU frequency is dynamically adjusted to save power; for example, by using techniques such as dynamic voltage and frequency scaling (DVFS). Consequently, and due to the complexity of modelling power consumption in multi-core systems, this thesis estimates energy consumption using real data of the power consumption of two specific servers. CloudSim uses the SPECpower[15] benchmark for the estimation of the energy consumption by the PMs. Table 3.3, from [BB12], exhibits the energy consumption, in Watts, at different CPU utilization levels of two server configurations with dual-core CPUs. The selected servers, shown in Table 3.3, are HP ProLiant ML110 G4 and HP ProLiant ML110 G5.

Table 3.3: Energy consumption (in Watts) at different CPU utilization levels [BB12]

| Server Name | 0% | 10% | 20% | 30% | 40% | 50% | 60% | 70% | 80% | 90% | 100% |
|---|---|---|---|---|---|---|---|---|---|---|---|
| HP ProLiant G4 | 86 | 89.4 | 92.6 | 96 | 99.5 | 102 | 106 | 108 | 112 | 114 | 117 |
| HP ProLiant G5 | 93.7 | 97 | 101 | 105 | 110 | 116 | 121 | 125 | 129 | 133 | 135 |

2. **Absolute normalized number of running PMs:**
The absolute normalized number of running PMs (ANPMs) calculates the absolute normalized total number of active PMs in the CDC by considering the CPU capacity and the running time of each PM. The CPU capacity of a PM is normalized to the largest CPU capacity of all active PMs, and the running time of a PM that is calculated relative to the total simulation time of the experiment. As an illustration, if there is a single PM running during the whole simulation time and its CPU capacity is equal to the capacity of the PM with the largest CPU, then the *ANPMs* will be increased by one. Similarly, if the CPU capacity of a PM is equal to the CPU capacity of the largest PM and the PM runs only for half of the simulation time, then the *ANPMs* is going to be increased by 0.5; and so forth. Equation 3.1 demonstrates the computation of the absolute normalized number of PMs running in the CDC.

$$ANPMs = \sum_{i=1}^{n} (norm_i^{cpu} \times norm_i^{ta}). \tag{3.1}$$

In Equation 3.1, $norm_i^{cpu}$ is the normalized CPU capacity (in million instructions per second (MIPs)) of $PM_i$ and $norm_i^{ta}$ is the normalized active time of

[15] http://www.spec.org/power-ssj2008

PM$i$. Equations (3.2 and 3.3) shows the calculations of both $norm_i^{cpu}$ and $norm_i^{ta}$ respectively.

$$norm_i^{cpu} = \frac{c_i^{cpu}}{c_{max}^{cpu}}. \tag{3.2}$$

$$norm_i^{ta} = \frac{t_{ai}}{t_s}. \tag{3.3}$$

Here, $c_i^{cpu}$ is the CPU capacity of PM$_i$ and $c_{max}^{cpu}$ is the CPU capacity of the PM with the largest CPU capacity in the CDC. Additionally, $t_{ai}$ is the running (active) time of PM$_i$, and $t_s$ is the total simulation time of the experiment.

3. **Number of migrations and migration Time ($RT_m$):**
The number of migrations defines the total number of VM migrations. Too many unnecessary migrations can result in more energy consumption [DMKP16]. The relative migration time $RT_m$ is the percentage of time required for all VMs' migrations which is calculated in Equation 3.4, where $t_{mj}$ is the time required to migrate VM$_j$, and $t_{aj}$ is the total time during which VM$_j$ was active.

$$RT_m = \frac{\sum_{j=1}^{m} t_{mj}}{\sum_{j=1}^{m} t_{aj}}. \tag{3.4}$$

4. **SLA violations (SLAVs):**
Service level agreement (SLA) defines the quality of service requirements (QoS) as a contract between the service provider and the customer. In a cloud infrastructure, SLA is considered to be violated when the resource requirements of a VM (bounded by the maximum VM size) are not sufficiently provided at any point of time. Therefore, estimating SLA violations (SLAVs) is commonly determined by the amount of under-allocated resources per VM at any time. There are two primary sources of the SLAVs in the IaaS service model [BB12]:

   (a) The first source of SLAVs is the one that results from the *overutilization of PMs* due to the aggressive consolidation of VMs to ensure better utilization ratios. The SLAVs resulting from the overutilization of PMs ($SLAVO$) is estimated as follows:

   $$SLAVO = \frac{1}{n}\sum_{i=1}^{n}\frac{t_{oi}}{t_{ai}},$$

   where $n$ is the number of running PMs, $t_{oi}$ is the time period during which PM$_i$ is overutilized, and $t_{ai}$ is the active (running) time of PM$_i$.

(b) The second source of SALVs is the one which results from the *migration of VMs* among various PMs. The VMs can be migrated when the hosting PM is overutilized or underutilized due to the change in workload demand of VMs over time. The SLAVs resulting from the migration of VMs among PMs ($SLAVM$) is estimated as follows:

$$SLAVM = \frac{1}{m}\sum_{j=1}^{m}\frac{d_{dj}^{cpu}}{d_j^{cpu}},$$

where $m$ is the total number of running VMs, $d_{dj}^{cpu}$ is the amount of under-allocated CPU demand per $\text{VM}_j$ due to VM migration, and $d_j^{cpu}$ is the total CPU demand by $\text{VM}_j$.

Therefore, SLAVs is estimated as a composite metric that considers the two previously sated sources of violations as shown in Equation (3.5).

$$SLAV = SLAVO \times SLAVM. \tag{3.5}$$

The $SLAVO$ estimates the SLAVs resulting for the overutilization of PMs, and $SLAVM$ calculates the SLAVs resulting from the migration of VMs among the CDC's PMs. Further details on the two sources of SLAVs can be found in Section 4.3.

5. **Total number of SLA violations (NSLAVs):**
Another way for estimating the SLAVs is to count the number of times during which the allocated resources per PM are less than the requested resources. Therefore, we can say that there is a violation per $\text{PM}_i$ at time $t$, when the amount of allocated CPU $a_i^{cpu}(t)$ is less than the actual CPU demand $d_i^{cpu}(t)$ regardless of the time of the violation.

$$NSLAV_{it} = \begin{cases} 1 & \text{if: } a_i^{cpu}(t) < d_i^{cpu}(t) \\ 0 & \text{otherwise} \end{cases}$$

6. **Average CPU utilization:**
By and large, the CPU utilization of a PM refers to the actual usage of the CPU resource to run the required workload. In the conducted experiments, the CPU utilization of $\text{PM}_i$ at time $t$ is estimated by dividing the CPU demand by the PM's

CPU capacity. However, if there is an SLA violation per PM (*i.e.* the allocated resources are less than the actual demand), then the CPU utilization is calculated by dividing the allocated CPU by the CPU capacity of that PM. The following Equation summarizes the two cases of estimating the CPU utilization of $PM_i$ at time $t$:

$$u_i^{cpu}(t) = \begin{cases} \frac{a_i^{cpu}(t)}{c_i^{cpu}(t)} & \text{if: } a_i^{cpu}(t) <= d_i^{cpu}(t) \\ \frac{d_i^{cpu}(t)}{c_i^{cpu}(t)} & \text{otherwise} \end{cases}$$

The CPU utilization of a single PM is the average utilization of the PM's CPU over the simulation time for each scheduling interval. As an illustration, if the simulation time is one hour, and the scheduling interval is five minutes, then CPU utilization is the average of the utilizations across the hour for each interval (*i.e.* the average of 12 CPU utilizations). Accordingly, the CPU utilization of $PM_i$ across all the scheduling intervals is calculated as follows:

$$u_i^{cpu} = \frac{1}{q} \cdot \sum_{l=1}^{q} u_{il}^{cpu},$$

where $q$ is the number of scheduling intervals and $u_{il}^{cpu}$ is the CPU utilization of $PM_i$ at interval $l$. The average CPU utilization of all PMs across the entire CDC is defined as follows:

$$U_{cdc}^{cpu} = \frac{1}{n} \cdot \sum_{i=1}^{n} u_i^{cpu}.$$

7. **Number of PM shutdowns:**
   The frequent shutting down of PMs incurs cost as there is a setup cost to get the powered off PM up and running. For example, powering on an application server takes 260 seconds, during which the server is using its peak power [GHBRK12]. Turning a PM on also could take extra time for updating the application and/or the operating systems, which adds extra cost. Therefore, the number of shutdowns should not be too much so as not to lead to more SLA violations.

## 3.7 Summary

In this chapter, we have presented the main components of the architecture of the proposed parameter-based VM placement solution. These components are the CDC, the

VM placement requests, the initial VM placement controller, monitoring, dynamic VM placement controller, and the parameter estimation component. The parameter estimation component is introduced to work with the proposed parameter-based VM placement strategy. This parameter can be either statically defined or dynamically estimated on the fly. Then, the chapter discussed the importance of using cloud simulators due to the costs, limitations, and difficulties associated with the in-vivo experiments. After that, the chapter reviewed three of the most widely used cloud simulators and outlined the criteria for choosing the proper one. This work uses CloudSim simulation toolkit for the empirical evaluation of the proposed algorithms. The layered CloudSim architecture has been introduced, and the settings of the simulated CDC has been demonstrated. After that, the chapter described both the real (such as normal PlanetLab and skewed PlanetLab) and the synthetic workload traces. Finally, the chapter defined and demonstrated the estimation of the various performance metrics that are going to be used throughout the thesis to measure the efficiency of the proposed algorithms.

# Chapter 4

# Parameter-based Virtual Machine Placement

## 4.1 Introduction

As pointed out in Chapters 1 and 2, the reservation-based and demand-based VM placement strategies represent the two extremes for allocating VMs in CDCs. Additionally, both the initial (static) and the dynamic reservation-based VM placement strategies do not utilize the CDC's PMs efficiently, which results in more energy waste and $CO_2$ emissions. However, the reservation-based VM placement strategy can satisfy SLA requirements (here, we ignore hardware and network failures). On the other hand, the dynamic demand-based VM placement strategy utilizes the infrastructure efficiently by ensuring better consolidation ratios, which minimize underutilization and hence saves energy, $CO_2$ emissions and operational costs. However, the demand-based VM placement results in more SLA violations (SLAVs) from both the PMs' overload situations and the excessive migrations of VMs, which degrade the quality of service (QoS). Therefore, there is a trade-off between the efficient utilization of PMs and the resulting SLAVs.

This chapter introduces a new parameter-based VM placement solution based on a novel VM placement strategy by investigating the middle way between the demand-based and reservation-based VM placement strategies. We have named the proposed VM placement strategy as *parameter-based* because it exploits the space between the full reservation and the demand-based allocation with the help of a single parameter ($\alpha$). The operators of public or private clouds can set the value of that parameter based on the business preferences. The work on this chapter is based on the paper [MS17b].

## 4.2 VM Placement Problem

The primary goal of the VM placement is to find a proper mapping of the VMs to the PMs that achieves better consolidation ratios while reducing SLAVs. An effective VM placement solution should find a good trade-off between utilization and SLAVs by reducing underutilization and overutilization situations of the CDC's PMs. A VM placement controller is usually embedded in a cloud management solution such as OpenStack. The VM placement in CDCs is a complex problem due to the heterogeneity of resource types, the variability, and unpredictability of the applications' workload, the conflicting objectives, and the large-scale of the current CDCs. The relationship between the VM placement controller and the other components of the parameter-based VM placement solution has been described earlier in Chapter 3. The following section presents the mathematical formulation of the VM placement problem, and Section 4.2.2 defines the objective function.

### 4.2.1 Mathematical Formulation

This section presents the mathematical formulation of the VM placement in a single CDC that is managed by a public cloud provider or a private cloud owner. Supposing that the CDC houses $n$ physical machines (PMs) defined as $P = \{p_1, p_2, \ldots, p_n\}$; each $p \in P$ is defined as consisting of three distinct resource types, namely CPU, memory (Mem) and bandwidth (BW). The types of resources of any PM are defined as $r \in R$, where $R = \{CPU, Mem, BW\}$. The capacity $c$ of resource type $r$ of PM$_i$ is defined as $c_i^r \in C^r$, where $C^r = \{c_1^r, c_2^r, \ldots c_n^r\}$. Accordingly, $c_i^{cpu}$, $c_i^{mem}$ and $c_i^{bw}$ represent PM$_i$'s total capacity of CPU, memory and bandwidth respectively. At any time $t$, a PM$_i$ could be either *active (running)* or *inactive* (switched off or in any of the power saving modes).

$$p_i = \begin{cases} 0, & \text{if: PM}_i \text{ is not active} \\ 1, & \text{if: PM}_i \text{ is active} \end{cases}$$

The cloud provider offers various VMs instances to IaaS customers; the CDC can run $m$ VMs on the available PMs. Each VM is defined as consisting of CPU, Mem and BW resources, and as $v \in V$ where $V = \{v_1, v_2, \ldots v_m\}$. The initial VM placement controller maps the set of the VMs ($V$) to a subset of the active PMs ($P$) using the mapping function; $f : V \rightarrow P$ such that $\forall v \in V, \exists p \in P : p = f(v)$ as shown in Figure 4.1. The placement of VM$_j$ on PM$_i$ at time $t$ is defined as $x_{ij}(t) = 1$. Equation 4.1 ensures

that, any active VM is only placed on a single PM at time $t$.

$$\sum_{i=1}^{n} x_{ij}(t) = 1. \tag{4.1}$$

Additionally, the initial VM placement controller ensures that the resource capacity of each PM should exceed the total capacity of the hosted VMs.

The dynamic VM placement controller adaptively reallocates VMs according to the cloud provider's objectives during each scheduling interval $l \in L$, $L = \{l_1, l_2, \ldots l_q\}$. The total resource demand (of the applications running on the hosted VMs) for each resource type $r$ of $\text{PM}_i$ at time $t$ is defined as $d_i^r(t) \in D^r$, where $D^r = \{d_1^r, d_2^r, \ldots d_n^r\}$. Therefore, $d_i^{cpu}(t)$, $d_i^{mem}(t)$ and $d_i^{bw}(t)$ represent the CPU, memory and bandwidth demand of all the VMs running on $\text{PM}_i$ at time $t$. The resource demand of $\text{VM}_j$ at time$_t$ $d_j^r(t)$ is bounded by the reserved capacity of the $\text{VM}_j$ ($c_j^r(t)$); therefore, $c_j^r(t) \geq d_j^r(t)$. The allocated resources to $\text{VM}_j$ at time $t$ is defined as $a_j^r(t)$; therefore, $a_j^r(t) \geq d_j^r(t)$ satisfies resource requirements per VMs by ensuring that the allocated resources for $\text{VM}_j$ are greater than or equal to the requested resources (actual demand). At any time $t$, if $a_j^r(t) < d_j^r(t)$, this means that $\text{VM}_j$ is likely to incur SLAVs. To avoid SLAVs while ensuring better utilization for any $\text{VM}_j$ at any time $t$, the relationship among the VM's capacity (full reservation), the actual resource demand, and the allocated resources should be as follows:

$$c_j^r(t) \geq a_j^r(t) \geq d_j^r(t).$$

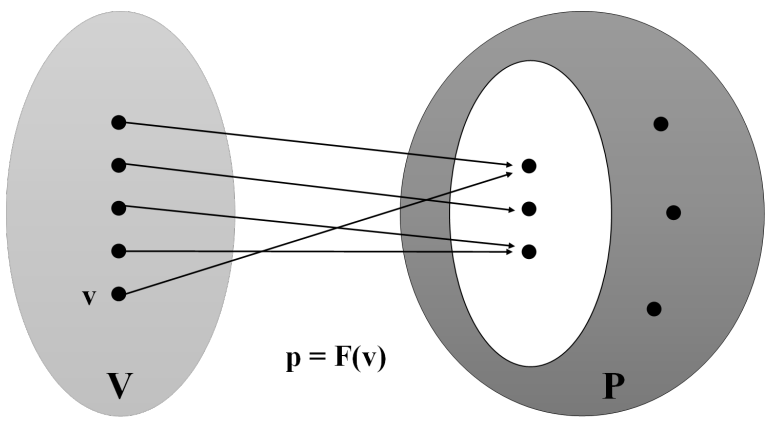



Figure 4.1: VMs-to-PMs mapping

At any time, the utilization state of a PM could be in one of the following five utilization states; namely, *fully-utilized*, *partially-utilized*, *under-utilized*, *over-utilized* or *non-utilized*. A PM is called *non-utilized* when it is either switched off or in one of the power saving modes which indicates that the total resource utilization is 0%. A PM is considered as *over-utilized* when the total demand from the hosted VMs is greater than the PM's capacity of any resource type $r$ (*i.e.* $d_i^r(t) > c_i^r(t)$) or when the maximum utilization of any resource type is greater than the overutilization threshold. A PM is *underutilized* when it is active, and the maximum utilization of any resource type is less than the underutilization threshold. A partially-utilized PM is the one that is neither overutilized nor underutilized. Assuming that each PM is defined by $z$ resource types (CPU, Mem, BW); a summary of the five utilization states of any PM are formally specified below, where $underThreshold$ and $overThreshold$ represent the PM's underutilization and overutilization thresholds respectively.

$$state(p_i(t)) = \begin{cases} \textit{non-utilized} & \text{if: } \sum_{r=1}^{z} u_i^r(t) = 0\% \text{ or switched off} \\ \textit{fully-utilized} & \text{if: } u_i^r(t) = 100\% \\ \textit{partially-utilized} & \text{if: } underThreshold < u_i^r(t) < overThreshold \\ \textit{under-utilized} & \text{if: } 0\% < max_{r=1}^{z}(u_i^r(t)) \leqslant underThreshold \\ \textit{over-utilized} & \text{if: } max_{r=1}^{z}(u_{it}^r) \geqslant overThreshold \end{cases}$$

The percentage of underutilization per resource $r$ of PM$_i$ at time $t$ is $w_i^r(t) \in W^r$, where $W^r = \{w_1^r, w_2^r, \ldots, w_n^r\}$. Accordingly, the underutilization of CPU, memory, bandwidth per PM$_i$ at time $t$ is $w_i^{cpu}(t)$, $w_i^{mem}(t)$ and $w_i^{bw}(t)$ respectively. The underutilization per PM$_i$ at time $t$ ($w_i^r(t)$) is zero when the PM's utilization is 100% ($u_i^r(t) \geqslant 100\%$); the $w_i^r(t)$ is calculated as shown in Equation 4.2.

$$w_i^r(t) = (100 - u_i^r(t)). \tag{4.2}$$

Any PM can become overutilized per any resource type due to the aggressive consolidation of VMs which may violate the Quality of Service (QoS) requirements defined in the service level agreement (SLA). The overutilization per resource of type $r$ per PM$_i$ at time $t$ is defined as $o_i^r(t) \in O_i^r$. Accordingly, the overutilization of CPU, Mem and BW per PM$_i$ at time $t$ is defined as $o_i^{cpu}(t)$, $o_i^{ram}(t)$ and $o_i^{bw}(t)$ respectively.

Equation 4.3 shows the calculation of $o_i^r(t)$ in condition that $d_i^r(t) > c_i^r(t)$:

$$o_i^r(t) = \frac{d_i^r(t) - c_i^r(t)}{c_i^r(t)}. \tag{4.3}$$

An efficient VM placement solution aims to efficiently utilize the PMs by minimizing the average underutilization of active PMs. The estimation of average underutilization per resource type $r$ for all active PMs in a single scheduling interval $l$ is shown in Equation 4.4; where, $k$ defines the number of currently active partially-utilized or underutilized PMs.

$$\bar{W}_l^r = \frac{1}{k} \cdot \sum_{i=1}^{n} w_i^r. \tag{4.4}$$

Equation 4.5 calculates the average underutilization per resource $r$ of all PMs across all $q$ scheduling intervals:

$$\bar{W}^r = \frac{1}{q} \cdot \sum_{l=1}^{q} \bar{W}_l^r. \tag{4.5}$$

An efficient VM placement solution aims to reduce overutilization situations in order to avoid SLAVs penalties and customer dissatisfaction. Assuming that there are $y$ overutilized (overbooked) PMs, the average resource overutilization for all the overutilized PMs in the scheduling interval $l$ is computed as shown in Equation 4.6.

$$\bar{O}_t^r = \frac{1}{l} \cdot \sum_{i=1}^{n} o_i^r. \tag{4.6}$$

Finally, Equation 4.7 calculates the average resource overutilization per resource type $r$ of all running PMs across all the $q$ scheduling intervals.

$$\bar{O}^r = \frac{1}{q} \cdot \sum_{l=1}^{q} \bar{O}_l^r. \tag{4.7}$$

The following subsection defines the objective function of the proposed VM placement solution based on this mathematical formulation.

### 4.2.2 Objective Function of Proposed Solution

The main objective of the VM consolidation is to run the VMs on the least possible number of the PMs. By monitoring the current resource utilization, the objective function seeks to utilize the PMs efficiently by minimizing the underutilization per resource

type, while reducing SLA violations by minimizing overutilization. Equation (4.8) defines the objective function for each scheduling interval $l$ supposing that there are $z$ resource types.

$$\text{Minimize } (\sum_{r=1}^{z}(S_{rw}.\bar{W}_l^r + S_{ro}.\bar{O}_l^r)) \tag{4.8}$$

Here, $\bar{W}_l^r$ is the average underutilization per resource $r$ of all PMs running in interval $l$. Minimizing the underutilization per PM inherently means reducing the number of running PMs and hence improves the average utilization of the CDC. The $\bar{O}_l^r$ is the average overutilization per resource $r$ for PMs running in interval $l$. Minimizing overutilization reduces the SLAVs resulting from the aggressive consolidation of the VMs. The $S_{rw}$ and $S_{ro}$ are weights representing the relative significance of the underutilization and overutilization of each resource $r$ respectively. These weights are between zero and one as follows:

$$0 \leqslant S_{rw} \leqslant 1, \; and \; 0 \leqslant S_{ro} \leqslant 1$$

To ensure the SLA requirements are met, the capacity of any single $\text{PM}_i$ should be greater than the total allocated resources to all the hosted $h$ VMs.

$$c_i^r > \sum_{j=1}^{h} a_{ji}(t).$$

## 4.3 Sources of SLA Violations in IaaS

A service level agreement (SLA) defines the quality of service (QoS) requirements in the form of a contractual document between the service provider and the customer [LS14]. An SLA violation (SLAV) means that the previously agreed-upon SLA requirements have not been met. The SLA requirements are different for each type of cloud service models. For example, in the software as a service (SaaS) model, the response time is an essential requirement of an SLA. On the other hand, in the infrastructure as a service (IaaS) model, response time may not the most critical attribute; availability is usually an essential requirement. A proper definition of the SLA should be workload independent; therefore, an SLA is expressed as meeting the resource requirements of each VM over the time. In this thesis, we consider that SLA requirements of a VM are met when 100% of the resources requested by that VM are delivered at any time bounded by the maximum resource capacity of that VM. There

are many sources of SLA violations in the IaaS service model, such as *live migration* of VMs, *overutilization of the PMs*, hardware, or network failure, system outage and interference among co-located VMs. Regardless of the source of the SLA violations, at any point of time, there are SLAVs whenever the resources allocated to a VM are less than the requested resources bounded by the VM size, as shown in Figure 4.2.

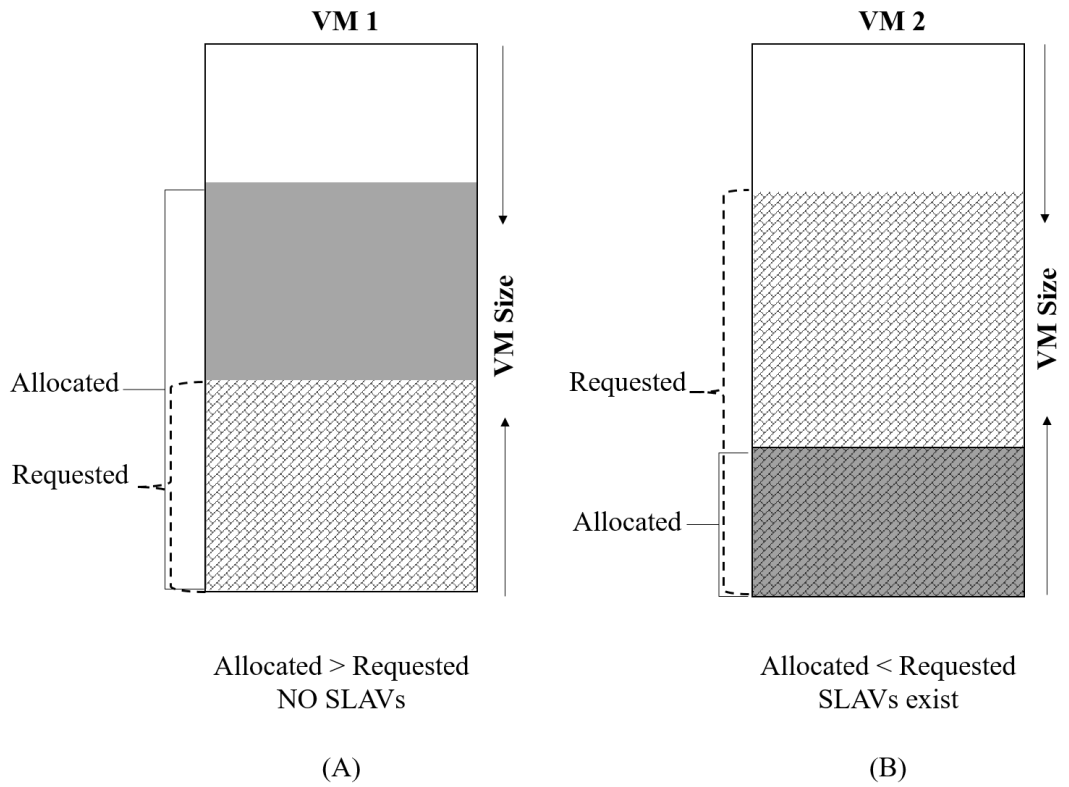



Figure 4.2: SLAVs based on the requested and allocated resources

The proposed VM placement solution considers only two sources of SLAVs, namely live migration and overutilization of the PMs. Overutilization of the PMs results from the aggressive consolidation of VMs through demand-based placement. Live migration causes performance degradation as the VM is not available during the migration time. The following subsections 4.3.1 and 4.3.2 detail violations from live migration and PM overutilization.

### 4.3.1 Violations due to Live Migration (SLAVM)

Live migration moves running VMs from one PM to another with no or minimal impact on the availability of VMs to users. Live migration is a valuable tool for the

administrators of cloud data centres as it facilitates multiple tasks such as balancing the load between physical servers, fault management, and low-level system maintenance [CFH+05]. The majority of the live migration process is completed while a VM is running, and hence it improves the efficiency of the cloud data centres with minimum downtime. The process of copying pages is performed while the VM is running. For example, if we are migrating $VM_j$ from $PM_1$ to $PM_2$ then, the downtime of $VM_j$ is only the time required to suspend $VM_j$ on $PM_1$ plus the time to generate address resolution protocol (ARP) request to redirect traffic to $PM_2$ in addition to the time to synchronize the state of $VM_j$ on $PM_2$ [CFH+05]. ARP is a communication protocol that finds the data link layer address (*e.g.*media access control (MAC) address) of the given logical address (*e.g.*internet protocol (IP)). The migration time depends on the available network bandwidth as well as the memory utilization of the running VM [Dar14]. There is a linear relationship between memory capacity and migration time. However, there is an inverse correlation between migration time and network bandwidth. Equation (4.9) estimates the time required to migrate a VM $j$ supposing that the images and data of VMs are stored on a storage area network (SAN).

$$t_{mj} = \frac{d_j^{mem}}{av_j^{bw}}. \tag{4.9}$$

Here, $t_{mj}$ is the time required to migrate VM $j$; $d_j^{mem}$ and $av_j^{bw}$ represent $VM_j$'s CPU demand and the available network bandwidth respectively.

Adopting live migration of the VMs might lead to more efficient utilization of PMs in a cloud data centre by migrating the VMs from under-utilized PMs and switching them into one of the power-saving modes or even turning them off. Moreover, live Migration might also assist in improving the quality of service (QoS) requirements by migrating VMs from over-utilized (overloaded) PMs. However, the number of VM migrations should be controlled in a way that helps strike a good balance between efficient utilization of the PMs and the resulting SLAVs due to the migration process. Too many migrations degrade the performance and increase the SLAVs, while too few migrations may lead to less efficient utilization of the PMs. Hence a balance is required to consider this trade-off. The reasons for migration depend upon the current state of the VMs-to-PMs mapping and the adopted VM placement policy.

The SLA violations from migrations (SLAVM) is a performance metric that estimates the performance degradation resulting from the migration of VMs among PMs. The *SLAVM* for a specific VM is estimated by dividing the underallocation per CPU

due to migration by the VM's total CPU demand. Equation 4.10 shows the calculation of the *SLAVM* for all VMs in the CDC.

$$SLAVM = \frac{1}{m}\sum_{j=1}^{m}\frac{d_{dj}^{cpu}}{d_{j}^{cpu}}, \tag{4.10}$$

where $d_{dj}^{cpu}$ is the VM$_j$'s under-allocated CPU demand due to migration, $d_{j}^{cpu}$ is the total CPU demand by VM$_j$ and $m$ is the total number of VMs.

### 4.3.2 Violations due to PMs' Overutilization

SLAs are satisfied when 100% of the VM's demand is met at any time. The SLAVO metric computes the SLA violations from PM's *overutilization*, which is the time during which the PM is 100% utilized in any of the resource types. SLAVO happens when the resource demand is higher than the allocated amount bounded by the VM's capacity. The SLAVO of PM$_i$ is calculated, as shown in Equation 4.11.

$$SLAVO_i = \frac{t_{oi}}{t_{ai}}. \tag{4.11}$$

Here, $t_{oi}$ is the time during which PM$_i$ is overutilized in one of its resource types (CPU, memory, or bandwidth) and $t_{ai}$ is the total time during which PM$_i$ is active (running). Equation 4.12 estimates the average SLAVO for all PMs in the CDC.

$$SLAVO = \frac{1}{n}\sum_{i=1}^{n}\frac{t_{oi}}{t_{ai}}. \tag{4.12}$$

## 4.4 Action Policy for Parameter-based VM Placement

Any VM placement solution is an example of an autonomic (self-managing) system that can adapt (by either reacting or proacting) to changes in the CDC without requiring any human intervention. Developing such an autonomic parameter-based VM placement solution can be achieved using one of the widely used policies, as previously discussed in Chapter 2. By reviewing the VM placement literature, the two most commonly used policies for creating such a self-managing system are action (rule-based) or utility function-based policies. The goal policies are not preferred for creating self-managing systems as the utility function is considered as a more flexible

and fine-grained extension of the goal policies [KD07]. The action (rule-based) systems take the form of *if* (condition) *then* (action); action policies only care about what action the system should take whenever it is in any given state. Action policies do not specify the state that the system will reach after taking action; it is all about what the system should do in the current state. The proposed autonomic parameter-based VM placement solution solves both initial and dynamic VM placement. It allocates the VM resources based on the proposed parameter-based VM placement strategy using an action-based policy. The reason for adopting an action policy is that it makes the presentation of the proposed parameter-based solution simpler to perceive; however, other policies can be used to achieve the same goal. The remainder of this section presents a summary of the adopted action-based framework for creating the proposed parameter-based VM placement solution.

Following the initial mapping of VMs to suitable PMs, the proposed dynamic parameter-based VM placement solution seeks to reallocate the VMs to achieve cloud provider's objectives. Figure 4.3 exhibits the adopted action-based policy for the dynamic reallocation of VMs in the CDC. Figure 4.3 illustrates how the system moves from the current state (current VMs-to-PMs mapping) to a new state (a more efficient VMs-to-PMs mapping) to either ensure more efficient utilization of PMs or reduce SLAVs whenever possible. The action-based policy for the dynamic reallocation of the VMs [XSC13, BB12, MR16] divides the dynamic VM placement problem into four subproblems, namely, *cold PM detection*, *hot PM detection*, *VM selection* and *VM placement*. Any active PM is classified as normal (partially-utilized) when it is neither overutilized nor underutilized. However, an active PM is considered hot or cold if it is either overutilized or underutilized, respectively. The details of how the system detects hot or cold PMs are described as follows:

- **Hot PM Detection:**
  The aggressive consolidation of VMs into CDC's PMs might make a PM overutilized when the total workload demands of the hosted VMs exceed the capacity of the hosting PM. The parameter-based VM solution tags a PM as *hot* (*i.e.* overutilized) when the CPU utilization of that PM exceeds a predefined overutilization threshold. The proposed parameter-based VM placement solution aims to satisfy the SLA requirements by minimizing the overutilization situations, which in turn reduces SLAVs. Accordingly, if the PM's *state* is *hot*, then the action should be *migrating some VMs* until the PM becomes normal (*partially-utilized* as defined in Section 4.2.1). There are various approaches for detecting hot PMs, such as

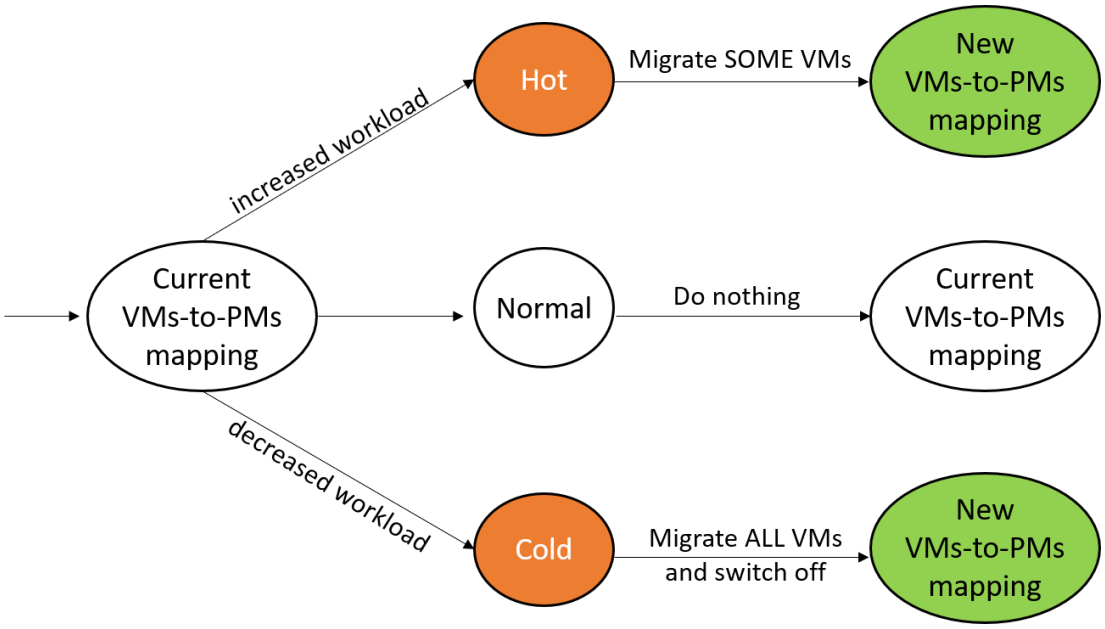



Figure 4.3: Action-based policy for the dynamic reallocation of VMs in CDC

using static threshold, local regression, robust local regression, or median absolute deviation [BB12, XSC13]. In this work, the proposed solution adopts a static threshold for hot PM detection. The static threshold identifies a PM as *hot* when the resource utilization of that PM is greater than the predefined static overutilization threshold. For example, if the overutilization threshold is set to 90%, then the system will tag the PM as hot whenever the PM's utilization is $\geq$ 90%.

- **Cold PM Detection:**
  The proposed parameter-based VM placement solution seeks to save energy by improving PMs' utilization through the reduction of the underutilization ratio per PMs. As a result, when the dynamic parameter-based VM placement controller identifies a cold (*i.e.*underutilized) PM, it attempts to correct the current state of the system by *migrating all the hosted VMs* to other PMs and switch that PM off or into one of the power saving modes. There are various approaches for detecting cold PMs, such as using a static threshold to define cold PMs based on the utilization percentage. However, the proposed solution selects the least utilized PM and try to migrate the hosted VMs into other PMs and turn that PM off whenever possible.

Recall Figure 4.3, at each scheduling interval (typically every five minutes), the

dynamic parameter-based VM placement controller uses the current state of the system to decide the next possible state based on the predefined placement objectives and constraints. If all PMs on the current VMs-to-PMs mapping are normal, then the dynamic VM placement controller does nothing and retains the current state of the system. However, if the state of some PMs is cold, then the dynamic VM placement controller will try to migrate all VMs and switch the cold PM off. On the other hand, when the dynamic VM placement controller detects a hot PM, then it should migrate one or more VMs from the hot PM until it becomes normal (partially-utilized).

The next question that arises is *which VMs to select for migration and why*. The answer to this question is based on the adopted *VM selection* approach. There are different approaches to decide which VMs to choose for migration from the hot PMs; these VM selection approaches include a random selection of the VMs, selecting VMs based on specific criteria such as memory or CPU utilization, or using fuzzy logic as in [MR16]. In the proposed parameter-based VM placement solution, we have chosen to select the VM with the minimum memory utilization to ensure minimum migration time per VM, which can help in reducing the performance degradation per VM due to migration. Regardless of the adopted VM selection approach, the migration of VMs from both hot and cold PMs results in a new state of the system, which is characterized by a new VMs-to-PMs mapping.

The final question is where (on which PMs?) to place the new VM placement requests (using the initial VM placement), and how much resources should be allocated for each new VM placement request. Additionally, during the dynamic VM placement, where to place the VMs selected for migration from both cold and hot PMs, and how much resources should be allocated for each of these VMs? The proposed parameter-based VM placement solution answers these questions about both the initial and the dynamic VM placement controllers in the following subsection.

### 4.4.1 Parameter-based VM Placement

The principal purpose of the VM placement is assigning VMs to PMs to achieve the cloud provider objectives without violating the constraints; this is accomplished using both the initial and dynamic VM placement as introduced beforehand. Consequently, the proposed parameter-based VM placement should consider both the initial and dynamic controllers. The initial VM placement controller receives VM placement requests from the cloud customers, and then either accepts or rejects the requests based on resource availability. Then, it assigns the accepted VM placement requests to the

appropriate PMs based on the resource capacity defined by the VM types (reservation-based placement). The initial VM placement is a kind of N-dimensional bin packing problem where PMs represent the bins, VMs represent the items to be packed, and the capacity of the resource types defines bin size. On the other hand, the dynamic VM placement controller starts with an existing VMs-to-PMs mapping and tries to generate a better mapping (if possible) based on the current one. The dynamic placement strives to find suitable PMs for allocating the VMs that have been selected for migration from hot and cold PMs. Finding a proper PM for VM migration is crucial as it dramatically affects the efficiency of the whole VM placement solution. The dynamic reallocation of the VMs runs periodically for finding a more efficient allocation that meets the cloud provider's objectives.

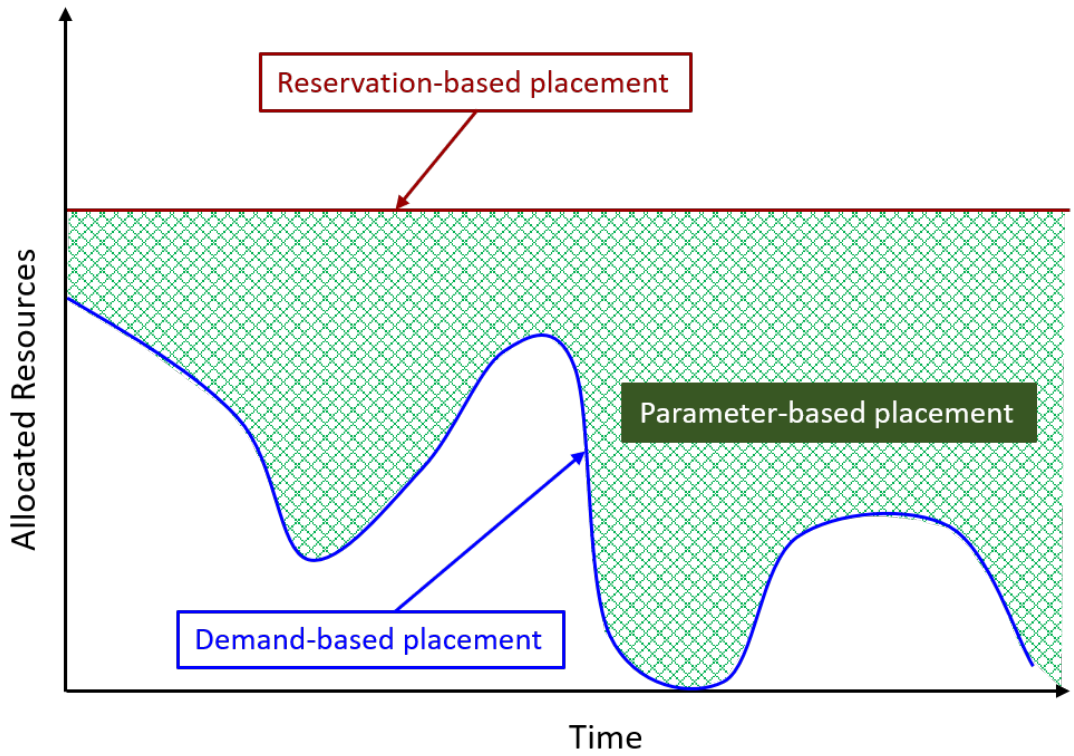



Figure 4.4: Reservation vs demand vs parameter-based placement

The proposed VM placement solution makes use of a novel parameter-based VM placement strategy, which explores the space between the resources required for the actual workload (demand-based allocation) and the resources needed for a VM at full capacity (reservation-based allocation). Figure 4.4 shows the area in which parameter-based placement operates compared to both the reservation-based and demand-based

placement approaches. Figure 4.4 demonstrates that the amount of resources allocated for a VM in case of reservation-based placement is constant over time, even if the VM is completely idle. However, the amount of allocated resources in case of demand-based placement is changing over time based on the workload demand.

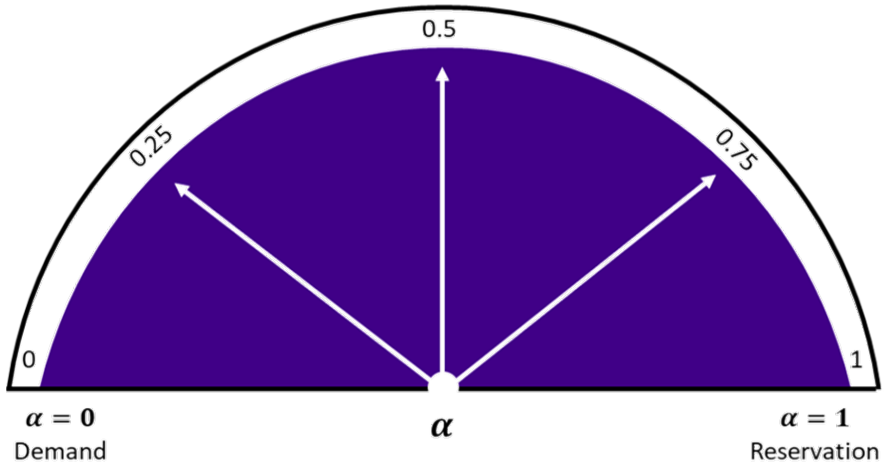



Figure 4.5: Parameter-based VM placement solution uses $0 <= \alpha <= 1$

The parameter-based VM placement strategy adds an extra margin (slack) to the allocated resources for meeting the actual demand requirements to accommodate some later sudden increase of the workload. The parameter-based VM placement assigns VMs to PMs based on a predefined parameter ($\alpha$) aiming to overcome the problems associated with both the reservation-based and demand-based VM placement strategies. It enables the cloud provider to fine-tune the trade-off between utilization (and energy accordingly) and SLAVs. The amount of resources allocated for a VM using the parameter-based VM placement strategy is defined in Equation (4.13), which allocates resources per VM by utilizing the space between the actual demand and the VM's size (total reservation). Equation 4.13 estimates how much resources the parameter-based VM placement strategy is going to allocate to $\mathrm{VM}_j$ at time $t$.

$$\mathrm{a}_j^r(t) = \alpha \cdot (c_j^r - d_j^r(t)) + d_j^r(t), \tag{4.13}$$

where $\mathrm{a}_j^r(t)$ is the amount of resources (between full reservation and actual demand) that should be allocated to $\mathrm{VM}_j$ at time $t$, $c_j^r$ is the capacity of resource $r$ per $\mathrm{VM}_j$ (reservation) and $d_j^r(t)$ represents the actual demand of resource $r$ at time $t$. The parameter $\alpha$ represents a knob that specifies the desired scale between the actual demand and the full reservation. As an illustration, when the parameter $\alpha$ is set to zero ($\alpha = 0$),

this means that we are adopting demand-based placement. On the other hand, when the parameter $\alpha$ is set to one ($\alpha = 1$), this corresponds to reservation-based placement. Figure 4.5 shows that the parameter $\alpha$ of the parameter-based strategy acts as a knob in the range $0 \leqslant \alpha \leqslant 1$ to give different allocation options for the cloud provider.

The cloud operator can set the parameter $\alpha$, which is an input to the parameter-based VM placement controller, as shown in Figure 4.6. Besides the parameter, other inputs to the initial and dynamic VM placement controllers are sent to the parameter-based VM placement controller, as introduced in Chapter 3). The details of the parameter-based VM placement algorithm are presented in the remainder of this section.

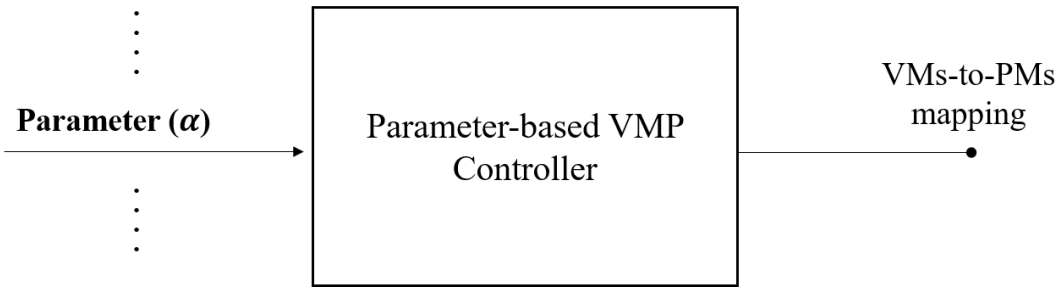



Figure 4.6: The parameter is an input to the parameter-based VM placement controller

**Parameter-based VM Placement Algorithm**

Algorithm 1 presents the proposed parameter-based VM placement solution which works for both initial and dynamic placement based on the "*type*" parameter. The "type" parameter is either *"INITIAL"* (for the initial placement of the VMs) or *"DYNAMIC"* (for the dynamic reallocation of the VMs based on the proposed parameter-based VM placement strategy). The *vmList* represents either the list of the VM that need to be initially allocated or the set of the VMs that requires migration to attain a satisfying VMs-to-PMs mapping state. The list of the VMs are sorted decreasingly based on the VMs' CPU capacity when the *type* is "*INITIAL*" or based on the real CPU demand when *type* is "*DYNAMIC*". For each VM on the VMs' list, the parameter-based VM placement algorithm needs to know the reserved size (capacity) of CPU per the VM ($c_{vm}^{cpu}$) as well as the actual CPU demand of the VM ($d_{vm}^{cpu}$). An active PM is a PM that is currently running while a PM that is switched off or in a power-saving mode is called an inactive PM. Both active and inactive PMs are sorted increasingly based on the PM's available CPU. Then, the algorithm decides the value of the parameter ($\alpha$):

**Algorithm 1** Parameter-based virtual machine placement

```
1:  vmList ← sort(vmList, type, "DESC");
2:  c_vm^cpu ← the CPU capacity of the VM;
3:  d_vm^cpu ← the CPU demand of the VM;
4:  inactivePMs ← sort(inactivePMs, "available^cpu", "ASC");
5:  vmsToPmsMapping ← Empty;
6:  if (type is "INITIAL")) then
7:      α ← 1;
8:  else
9:      α ← set the parameter (α);
10: for all (vm in vmList) do
11:     activePMs ← sort(activePMs, "available^cpu", "ASC");
12:     for all (activePm in activePms) do
13:         if (av_activePm^cpu > α.(c_vm^cpu − d_vm^cpu) + d_vm^cpu) then
14:             vmsToPmsMapping.add(vm, activePm)
15:     if the vm is still not allocated then
16:         for all (inactivePm in inactivePms) do
17:             if (av_inactivePm^cpu > α.(c_vm^cpu − d_vm^cpu) + d_vm^cpu) then
18:                 vmsToPmsMapping.add(vm, inactivePm)
19: return vmsToPmsMapping;
```

for initial VM placement, the VMs are allocated based on the reserved size, and therefore $\alpha$ is set to 1; otherwise, $\alpha$ is set based on the preferences of the cloud operator. After that, Algorithm 1 attempts to map each VM to a fitting PM based on the value of the parameter ($\alpha$). A PM is fitting for hosting a VM if the available CPU per the PM ($av^{cpu}_{activePm}$) is higher than the amount of CPU required for allocation based on the value of the parameter ($\alpha$). The algorithm will switch one of the suitable inactive PMs on when there is no fitting active PM to host the VM. Once a fitting PM found to host the VM, the VM and the PM are appended to the *vmsToPmsMapping*. The algorithm returns the VMs-to-PMs mapping when all VMs have been mapped to the appropriate PMs. The computational complexity of Algorithm 1 depends on the way of estimating the parameter $\alpha$ (statically or dynamically). Having $n$ PMs and $m$ VMs, the worst-case complexity of Algorithm 1 happens when the allocation of every VM requires activation of one of the inactive PMs. In case of a static value of the parameter $\alpha$ as in this chapter, the worst-case computational complexity of Algorithm 1 is going to be $O(n+m/2)m$. However, in the case of the dynamic estimation of the parameter, the computational complexity depends on the adopted approach for parameter estimation, as shown in Chapter 5.

## 4.5 Experimental Evaluation

Using the proposed parameter-based VM placement solution, the following experiments aim to explore the range between the demand-based placement and reservation-based placement. The experiments examine the effect of changing the amount of the allocated CPU resources by using various values of the parameter $\alpha$ as previously defined in Equation 4.13.

### 4.5.1 Experiments Setup

The experiments were conducted using the CloudSim[1] (version 3.0.3) simulation framework. The simulated cloud data centre (CDC) consists of 800 PMs, which are hosting 1033 VMs (the reason for choosing that specific number of VMs and PMs because the PlanetLab traces were extracted from an environment that same number of VMs and PMs). The exact details of the characteristics and types of VMs and the PMs have been previously described in Tables 3.1 and 3.2. The VMs in the following experiments runs three types of workload traces; namely, random, normal PlanetLab traces and skewed PlanetLab traces (the details of the workload traces, have been previously described in Section 3.5.

Algorithm 1 is used for both the initial and the dynamic placement of the VMs in the CDC. For the initial VM placement, the parameter $\alpha$ is set to one. For the dynamic VM placement, the amount of allocated resources for each VM ranges between the demand-based VM placement ($\alpha = 0$) to reservation-based VM placement ($\alpha = 1$). The experiments test the effect of changing the value of the parameter $\alpha$ from 0.0 to 1.0 with an increment of 0.1.

For the dynamic VM parameter-based placement, the conducted experiments react to both the hot and cold PMs by migrating VMs whenever possible:

- For *hot PM detection*: The experiments make use of a static overutilization threshold of CPU utilization. As an illustration, a PM is considered hot, when CPU utilization is 100%. Additionally, In Experiment 4, we have also explored other overutilization thresholds such as 80% and 90%.

- For *VM selection*: The following experiments migrate VMs from hot PMs to minimize the resulting SLAVs by reducing the overutilization situations. The

[1] https://github.com/Cloudslab/cloudsim/releases/tag/cloudsim-3.0.3

conducted experiments chose to start by migrating VMs with the minimum memory for the hot PMs so that it can reduce the overall migration time.

- For *cold PM detection*: After migrating VMs from hot PMs, the dynamic VM placement controller tries to migrate all VMs from the least utilized PM without resulting in any hot PMs whenever possible. If all VMs have been migrated from a cold PM, then the dynamic VM placement controller switches that PM off to save energy.

The simulated CDC runs for a whole day, and the dynamic parameter-based VM placement solution takes place every five minutes of simulated time. To measure the performance of the parameter-based VM placement solution, the conducted experiments measure the following performance metrics:(1) energy consumption; (2) absolute number of PMs (NPMs); (3) number of SLAVs (NSLAVs); (4) percentage of SLAVs (SLAVs); (5) average CPU utilization; (6) Number of VM migrations. The estimation of these performance metrics has been described in Section 3.6.

The academic literature on VM placement has revealed the permanence of SLA violations in the case of demand-based placement (*i.e.* setting $\alpha$ to 0); even when adopting highly sophisticated algorithms (for VM placement, PM overload and underload detection, and VM selection) [RCTY19, XHL19, KA19, ARMMA18, RKT19, WT18]. This confirms that all demand-based VM placement solutions suffer from SLA violations, and therefore there is always an area to explore between the demand- and reservation-based strategies. We have conducted three experiments, where each experiment runs using one of the three different workload traces. The details of the conducted experiments and the results are described below:

### 4.5.2 Experiment 1: Using Synthetic Traces

Experiment 1 analyses the results of the proposed parameter-based VM placement solution using the randomly generated CPU usage traces. In this experiment, various performance metrics have been calculated such as the absolute number of VMs, SLAVs, energy consumption and the average CPU utilization in relation to the change of the parameter $\alpha$.

**Absolute number of PMs (NPMs) to number of SLA violations (NSLAVs):**

Figure 4.7(a) shows the change in the number of SLAVs (NSLAVs) to the absolute number of the running PMs as the value of the parameter $\alpha$ changes from zero to one.

Each point in the figure represents a distinct value of the parameter $\alpha$ as defined in Equation 4.13. In the beginning, a little increment in the value of the parameter $\alpha$ results in a significant and gradual reduction in the number of SLAVs. As shown in Figure 4.7(a), using the demand-based VM placement (by setting $\alpha = 0$), the number of PMs is reduced by around 50% compared to the reservation-based VM placement when $\alpha$ is set to one. Looking at Figure 4.7(a), it is apparent that the demand-based VM placement introduces more SLA violations. The most interesting aspect of Figure 4.7(a) is that the cloud provider can reduce more than 37% of the total number of SLAVs with only around 13% increment in the number of running PMs by setting the value of the parameter $\alpha$ to 0.2, which authenticates the gained flexibility and benefit of adopting the parameter $\alpha$.

**Energy consumption to percentage of SLAVs:**

Figure 4.7(b) shows energy consumption to the percentage of SLAVs along with the change in the parameter $\alpha$. Figure 4.7(b) demonstrates that adopting the parameter-based VM placement strategy can enable the cloud provider to reduce energy consumption by around 28% (as opposed to reservation-based VM placement) while having a near 0% SLAVs (when setting the value of $\alpha$ to 0.6). The inverse relationship between energy consumption and SLAVs while changing the parameter $\alpha$ enables the cloud provider to find a suitable trade-off between the cloud service cost and violation costs. A closer inspection of Figure 4.7(b) suggests that adding an extra slack to the demand-based VM placement (*e.g.* setting $\alpha = 0.2$) can significantly reduce SLAVs with a moderate increase in the amount of energy consumed.

**Average CPU utilization:**

The efficient utilization of PMs is of great importance to the operators of CDCs. Therefore, the dynamic reallocation of VMs tries to ensure more efficient utilization by reducing the underutilization scenarios. It is apparent from Figure 4.7(c) that the demand-based placement ($\alpha = 0$) improves the average CPU utilization by more than 50% (70.7% as opposed to 47% in case of reservation-based VM placement). Figure 4.7(c) shows that setting the value of the parameter $\alpha$ in the range [0.1, 0.3] improves PMs' utilization. Additionally, it also shows that the highest CPU utilization is achieved when the values of the parameter $\alpha$ are set between 0.1 and 0.2. Besides reducing the SLAVs (as shown in Figure 4.7(a)), adjusting the value of the parameter $\alpha$ may improve the average utilization of the PMs.

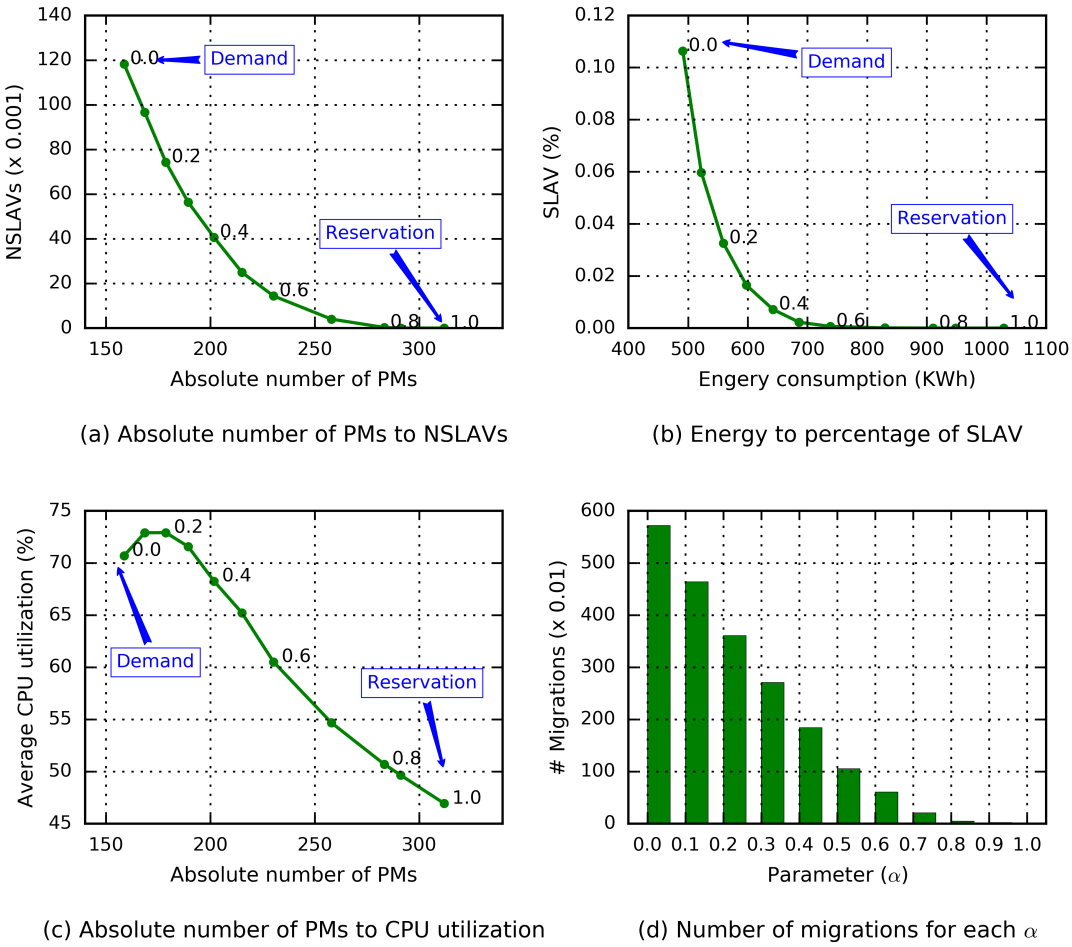


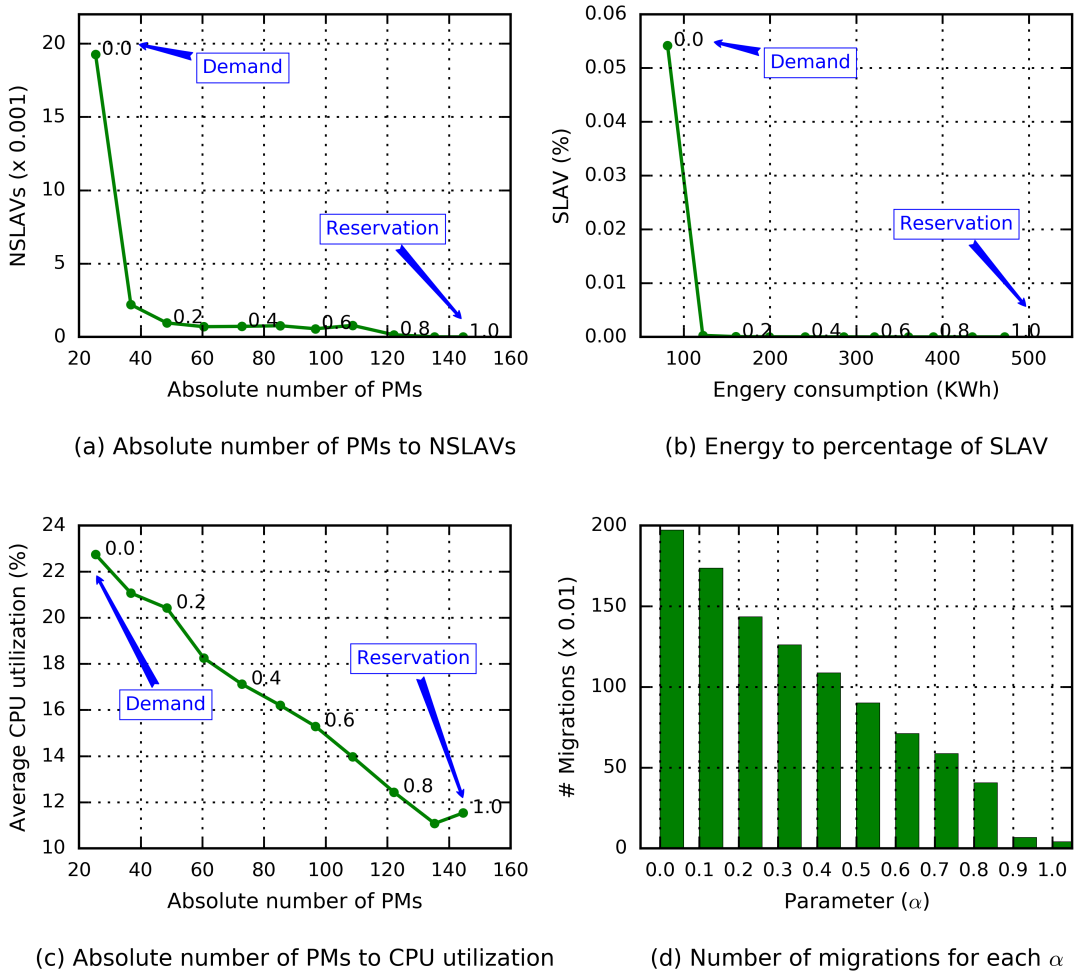

Figure 4.7: Experiment 1: using synthetic workload traces

**Number of VM migrations:**

Figure 4.7(d) shows the change in the number of VM migrations for each value of the parameter α. From Figure 4.7(d), it is clear that increasing the value of parameter α between zero and one results in a significant reduction in the number of VM migrations. Additionally, when the parameter α is set to one (reservation-based placement), there are no migrations (number of migrations is zero). Therefore, using the parameter-based VM placement, the cloud provider can control the number of VM migrations within the CDC to reduce the overall SLAVs.

### 4.5.3 Experiment 2: Using Normal PlanetLab Traces

In contrast to Experiment 1, this experiment evaluates the proposed parameter-based VM placement solution using real workload traces from PlanetLab. Recall from Chapter 3, the average CPU utilization of the normal PlanetLab traces is low, which validates the fact that most servers are underutilized most of the time. The effect of this low utilization can be seen in the following results from the parameter-based VM placement solution using normal PlanetLab traces.




(a) Absolute number of PMs to NSLAVs

(b) Energy to percentage of SLAV

(c) Absolute number of PMs to CPU utilization

(d) Number of migrations for each $\alpha$

Figure 4.8: Experiment 2: using normal PlanetLab traces

**Absolute number of PMs (NPMs) to number of SLA violations (NSLAVs):**

Figure 4.8(a) presents the results obtained from using the normal PlanetLab traces by displaying the NSLAVs to the absolute number of PMs as the value of the parameter

$\alpha$ moves from zero to one. What stands out in Figure 4.8(a) is that it exhibits a sharp decrease in the number of SLAVs when the value of parameter $\alpha$ increases from 0 to 0.2. This rate of decrement in the number of SLAVs is even more apparent when compared to the results from the synthetic workload traces shown in Experiment 1. On the other hand, there is no significant change in the number of SLAVs for values of parameter $\alpha \geq 0.3$ due to the low utilization nature of the normal PlanetLab traces.

**Energy consumption to percentage of SLAVs:**

In Figure 4.8(b), there is a clear trend of decreasing the percentage of SLAVs when increasing the parameter $\alpha$ from 0.0 to 0.2. Compared to reservation-based placement (when $\alpha$ is set to one); setting the parameter $\alpha$ to 0.2 reduces the percentage of SLAVs to near 0% (the exact value is 0.00005%) while reducing the energy consumption by around 66%.

**Average CPU utilization:**

Figure 4.8(c) shows the average CPU utilization to the absolute number of PMs as the parameter-based VM placement solution moves between the demand-based and reservation-based placement. For the normal PlanetLab workload, it did not reveal precisely the same pattern that previously appeared in the random workload experiment. This is due to the nature of the normal PlanetLab traces, as the actual utilization of many VMs is 0%. As demonstrated in Equation (4.13), when the CPU demand of a VM is zero, then the value of the allocated CPU resources for the VM is going to be a percentage of the reservation. Accordingly, the CPU utilization is based only on the reservation (VM size) when the VM is entirely idle.

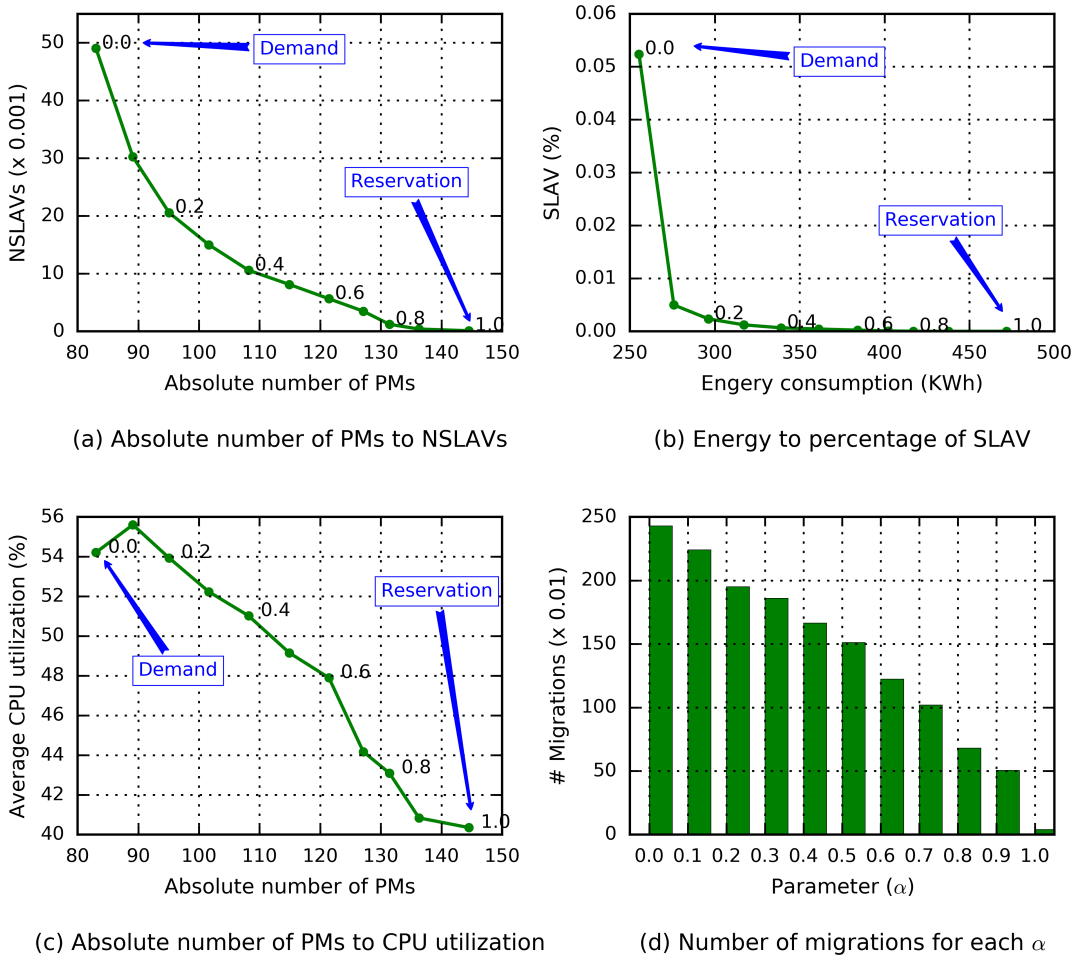
**Number of VM migrations:**

Figure 4.8(d) demonstrates that increasing the value of the parameter $\alpha$ from zero to one will significantly reduce the number of VM migrations. The number of VM migrations is reduced by 27% when setting the parameter $\alpha$ to 0.2. What is apparently counter-intuitive in Figure 4.8(d) is that when the parameter $\alpha$ is set to one (reservation-based), the number of VM migrations is not zero (the number of migrations is precisely 399). The reason for this is that many of the PlanetLab VMs have 0% utilization which makes some PMs underutilized, and hence VM migration happens to improve

the utilization of these PMs. If the cloud provider decided to prevent VM migrations when α is set to one, then he can achieve this by disabling the cold PM detection.

### 4.5.4 Experiment 3: Using Skewed PlanetLab Traces

Experiment 3 reports the results from adopting the parameter-based VM placement solution using the skewed workload traces from PlanetLab (more details about the traces can be found in Chapter 3).




(a) Absolute number of PMs to NSLAVs

(b) Energy to percentage of SLAV

(c) Absolute number of PMs to CPU utilization

(d) Number of migrations for each $\alpha$

Figure 4.9: Experiment 3: using skewed PlanetLab traces

**Absolute number of PMs (NPMs) to number of SLA violations (NSLAVs):**

Figure 4.9(a) proves that the skewed PlanetLab traces exhibits a similar behaviour to both the normal PlanetLab and the synthetic traces. All the tested traces confirm

that a little increase in the value of the parameter $\alpha$ results in a noticeable reduction in the number of SLAVs. In Figure 4.9(a), the decrease in the number of SLAVs shows a gradual behaviour (as opposed to the more sudden behaviour in case of normal PlanetLab traces in Experiment 2). The reason for this gradual decrease is that the average utilization of the VMs is higher in the skewed PlanetLab traces, which has also be seen on the synthetic traces of Experiment 1.

**Energy consumption to percentage of SLAVs:**

Figure 4.9(b) shows that increasing the value of the parameter $\alpha$ results in an apparent reduction in the percentage of SLAVs. Furthermore, Figure 4.9(b) confirms that there is also a significant decrease in the percentage of SLAVs as the parameter $\alpha$ approaches 0.2. Comparing Figure 4.9(b) to the results from the synthetic and normal PlanetLab traces demonstrates the same conclusion which confirms that it is a potential profit for the cloud provider to make use of the parameter $\alpha$.

**Average CPU utilization:**

Figure 4.9(c) reveals that a cloud provider could ensure better utilization of the PMs by a little increase in the value of the parameter $\alpha$ (setting $\alpha = 0.1$). Furthermore, when the parameter $\alpha$ is set to 0.2, the cloud provider could almost get the same utilization as in demand-based placement while reducing the number of SLAVs by 58% with only around 15% increase in the number of running PMs.

**Number of VM migrations:**

Figure 4.9(c) shows the same behaviour appeared in case of the synthetic and normal PlanetLab traces; increasing the value of the parameter $\alpha$ from zero to one results in a significant reduction in the number of VM migrations. As in case of the normal PlanetLab traces, it is clear that there are VM migration even when the parameter $\alpha$ is set to 1. The reason for these migrations, when $\alpha = 1$, is that some PMs are entirely idle (as the utilization of the hosted VMs is 0%) and hence some VM migrations will happen to improve the utilization of PMs. Disabling the cold PM detection mechanism will enable the CDC operator to prevent these migrations when setting $\alpha$ to one.

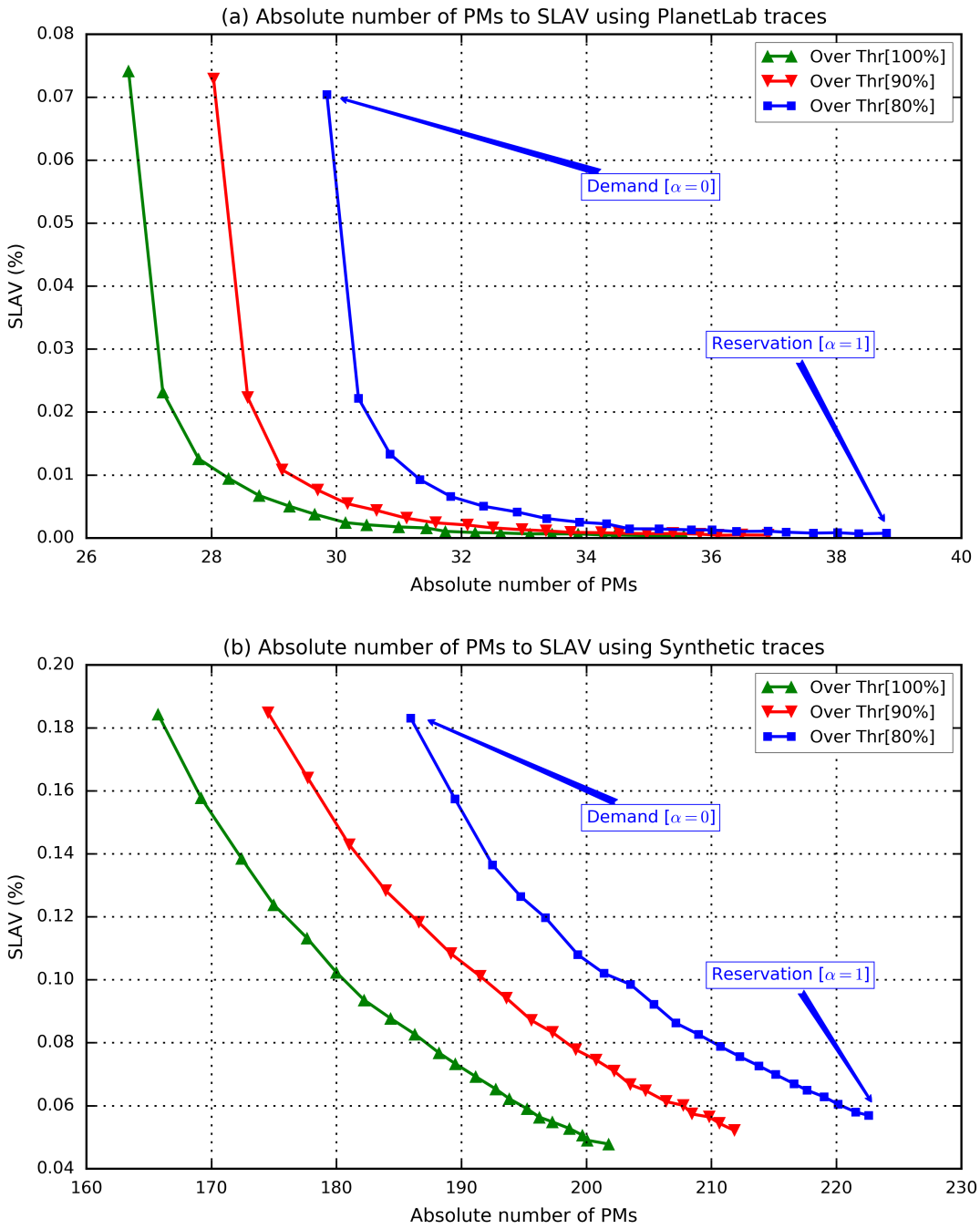



Figure 4.10: Experiment 4: PMs' Overutilization threshold versus parameter α using normal PlanetLab (a), and synthetic traces (b)

### 4.5.5 Experiment 4: Overutilization threshold versus Parameter α

The goal of Experiment 4 is to examine the effect of changing the PMs' overutilization threshold to the change in the value of parameter (α) for the hosted VMs.

In Figure 4.10, the X-axis exhibits the absolute number of PMs used for hosting the VMs, while the Y-axis represents the percentage of SLAVs. Additionally, each point represents a different value of the parameter $\alpha$ starting from 0 (demand) at the top left of the figure, and increasing by 0.1 till reaching 1 (reservation) at the bottom right of the figure. Figures 4.10(a),(b) present results using PlanetLab and synthetic traces respectively; the green, red and blue lines represent the number of PMs to SLAVs under the overutilization thresholds of 100%, 90% and 80% respectively. From Figure 4.10, it is apparent that there is a tradeoff between increasing the value of the parameter $\alpha$ and the decrease of the overutilization threshold using both normal PlanetLab traces and synthetic ones. Figure 4.10 confirms that reducing the overutilization threshold per individual PMs along with the increase in the value of the parameter $\alpha$ results in an apparent rise in the number of running PMs and a reduction in the resulting of SLAVs. On the other hand, increasing the PMs' overutilization threshold along with a decrease in the value of parameter $\alpha$ improves the PMs' utilization (shown in the lower number of used PMs) at the cost of an increase in the SLAVs. Overall, Experiment 4 indicates that the operators of CDCs can combine the PM's overutilization threshold and the parameter $\alpha$ of the VMs to get more viable options for the tradeoff between utilization and SLAVs.

## 4.6 Summary

This chapter started by introducing the mathematical formulation of the VM placement problem and the objective function as well as the placement constraints. Then, it explained the sources of SLAVs in the IaaS service model which commonly result from both the overutilization situations per PMs due to the aggressive consolidation of the VMs as well as the migrations of the VMs among PMs. The VM migration time depends on both the memory utilization and the available network bandwidth. Next, the chapter introduced the new parameter-based VM placement strategy which represents the middle way between the existing VM placement strategies (demand-based and reservation-based VM placement strategies).

The proposed parameter-based VM placement strategy makes use of a single parameter ($\alpha$) that allows investigating the entire area between full-reservation and demand-based allocation based on a custom equation. For solving the proposed parameter-based VM placement strategy, the chapter presented an action (rule-based) policy for both the initial and the dynamic VM placement based on the type of placement.

The empirical evaluation of the proposed parameter-based VM placement using three types of workload traces revealed exciting insights as well as viable options for the cloud operators. The results confirm that the parameter-based VM placement strategy can give the cloud operators more flexibility in choosing between different alternatives. For example, a little increase in the value of the parameter ($\alpha$) results in a significant change in the energy consumption and SLAVs which enables the cloud provider to find a better balance between the conflicting objectives. Additionally, it is found that combining the PM's overutilization threshold in relation to the parameter $\alpha$ of the VMs results in more options for the tradeoff between utilization and SLAVs. Such viable alternatives are not available when using both demand-based and reservation-based VM placement strategies. To conclude, the proposed parameter-based VM placement can open new doors for various opportunities for public cloud providers or private cloud owners to strike an appropriate balance between energy consumption and SLAVs.

# Chapter 5

# Dynamic Tuning for Parameter-based Virtual Machine Placement

## 5.1 Introduction

In Chapter 4, we have investigated the middle way between allocating VMs to PMs based on the reserved VM size (reservation-based placement) and the actual workload demand of VMs (demand-based placement). Examining this middle way resulted in the proposed parameter-based VM placement solution that offers more flexibility in CDC management through enabling cloud providers to specify the optimization ratio of business-related metrics such as PM utilization or SLA violations (SLAVs), among others. For this purpose, the parameter $\alpha$ of our solution is set statically to the desired ratio and as an input to the VM placement algorithm. In our evaluation of our solution in Chapter 4, the value of the parameter $\alpha$ remained constant throughout each entire experiment and, in industrial settings, would remain constant until explicitly changed.

This chapter extends the capabilities of the parameter-based VM placement solution by introducing a dynamically adjusted value for the parameter $\alpha$. The chapter first demonstrates the need and benefits of employing such dynamic adjustment. This is followed by an introduction to the proposed dynamic algorithms that can automatically fine-tune the value of the parameter $\alpha$ on the fly based on the cloud provider's objectives and any other placement constraints. Finally, an empirical evaluation is presented to evaluate the proposed algorithms for the dynamic estimation of the parameter $\alpha$. The work on this chapter is based on the paper [MS18].

## 5.2 The Need for a Dynamic Parameter

One of the main characteristics for the wide adoption of cloud computing is the rapid elasticity where the cloud services are easily provisioned, released and scaled up and down based on the user request. There are two common scaling approaches in cloud environments namely, *vertical* and *horizontal* scaling [VRMB11]. Vertical scaling resizes each VM by scaling a VM's allocated resources (CPU, memory, bandwidth) up when the workload ramps up or down otherwise. Horizontal scaling involves adding more PMs or VMs for running the hosted applications.

The parameter-based VM placement scales the allocated resources per each VM up or down based on the cloud provider objectives. It considers vertical scaling by adding an extra margin (slack) to the resources required to meet the actual demand to accommodate some later sudden increase of the workload. This margin is calculated based on the current value of the parameter $\alpha$ as shown in Equation 4.13 which introduced in Chapter 4 and recalled here for convenience. The parameter $\alpha$ in Equation 4.13 specifies the extra slack to the demand-based placement based on the difference between the full reservation and the actual demand. The changes in the value of the parameter result in many possible VMs-to-PMs mappings which reflect different trade-offs between the utilization and SLAVs. The extreme values of the parameter (0 and 1) represent the two extremes of the VM placement strategies.

$$\alpha = \begin{cases} 0, & \text{Demand-based VM Placement} \\ 1, & \text{Reservation-based VM Placement} \end{cases}$$

There are some questions that cloud providers may raise to ensure efficient placement of VMs in the CDC, for example:

- What value of the parameter $\alpha$ should the cloud provider set so that the resulting SLAVs do not surpass a specific SLAVs threshold?
- Which value of the parameter $\alpha$ the cloud provider should set to ensure that the consumption is below a specific threshold?

Adopting a parameter-based VM placement solution that keeps using the same value of $\alpha$ (static parameter) will not enable the cloud provider to set a SLAVs threshold that should not be surpassed. Such a statically predefined value of the parameter $\alpha$ cannot guarantee a number of SLAVs less than a predefined SLAVs threshold (unless the parameter $\alpha$ is set to 1, or close to 1). The reason for that is because the number

of SLAVs depends on both the current value of the parameter $\alpha$ (and hence the VM-to-PM mapping) and the specific characteristics of the workloads running on each single VM. Therefore, it becomes a challenge to dynamically estimate the value of the parameter $\alpha$ at each scheduling interval so that the resulting number of SLAVs does not exceed the SLAVs threshold (without underutilizing the resources to save energy consumption). Adopting a dynamic parameter can also enable the cloud provider to set other thresholds for energy consumption, the number of active PMs or the number of VM migrations and whatever he cares about. The following sections describe the proposed algorithms for estimating a dynamic parameter in more detail.

## 5.3 Dynamic Parameter-based VM Placement

We can dynamically tune the value of the parameter $\alpha$ either per (1) scheduling interval or (2) per VM. Optimizing the value of $\alpha$ for each scheduling interval means that the parameter $\alpha$ may be adjusted per interval based on the specified thresholds, but all VMs will have the same value of $\alpha$. Tuning $\alpha$ per VM means that the value of $\alpha$ may be modified per each VM for each scheduling interval. In this chapter, we are considering the online estimation of $\alpha$ per scheduling interval. Chapter 6 considers estimating $\alpha$ per VM for each interval.

In this work, we assume that there is a predefined SLAVs *threshold* (set by the cloud provider) that defines the number of SLAVs that should not be exceeded in order to meet the SLA requirements derived from some business goals. Similar thresholds can be managed for different business goals (*e.g.* energy consumption, cost ...). The proposed dynamic parameter-based VM placement solution observes the predefined threshold while actively seeking to optimize resource utilization. On the one hand, the proposed dynamic parameter-based VM placement algorithms can scale up a VM's allocated resources (bounded by the VM's maximum size) when the number of SLAVs exceeds the SLAVs threshold. On the other hand, the proposed algorithms can scale down the allocated resources when the number of current SLAVs is lower than the SLAVs threshold to ensure better utilization of the PMs. Figure 5.1 demonstrates one of the possible changes in the value of the parameter $\alpha$ over the scheduling intervals according to the relationship between the current SLAVs and the SLAVs threshold. Exceeding the SLAVs threshold results in an increase in the value of the parameter and vice versa.

In what follows, two generic approaches that can dynamically estimate the value of

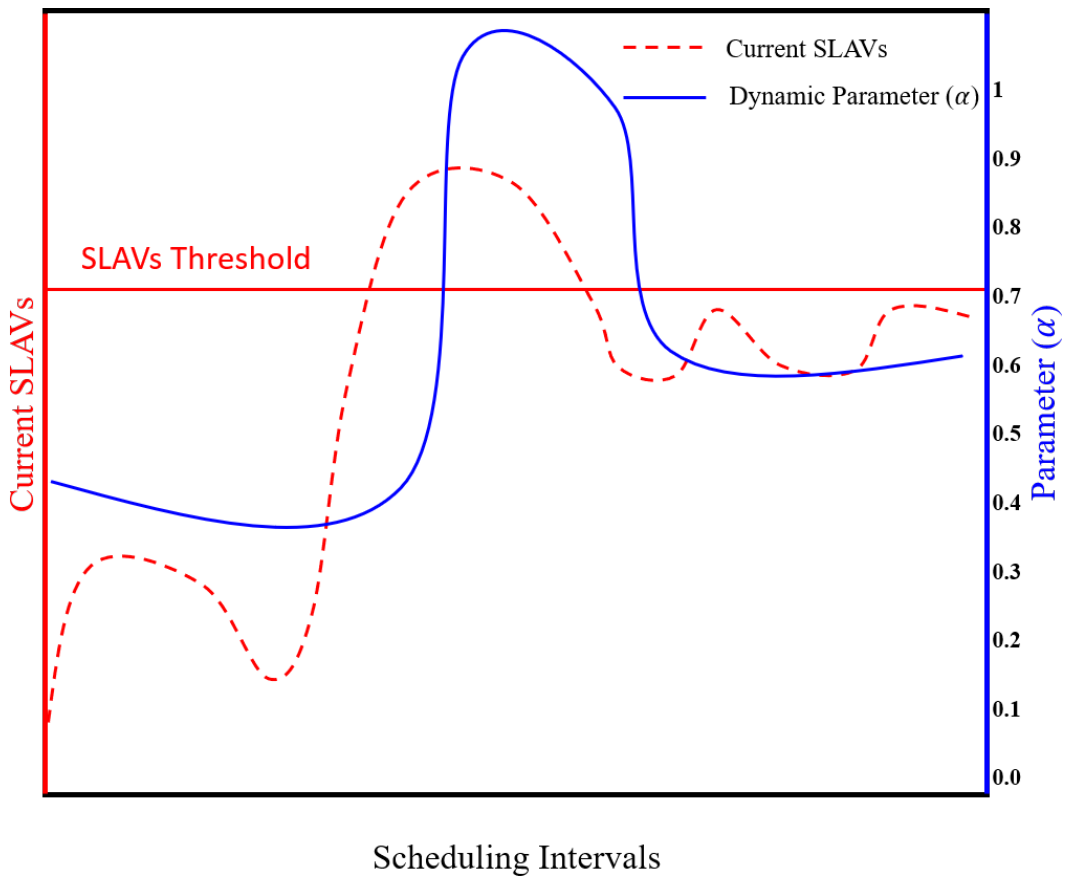



Figure 5.1: Dynamic parameter in relation to SLAVs threshold and current SLAVs

the parameter α are presented; namely, *instantaneous* and *window-based*. These two approaches correspond to different methods to increase or decrease the value of the parameter α at each scheduling interval. The rate of adjusting the value of the parameter α can be either conservative or non-conservative. By conservative we mean that the rate of increase or decrease is low, while non-conservative means the change rate of α is high. The non-conservative approach ensures a quick reaction to the changes in the CDC, while the conservative yields in a slower response to the changes. Making use of the benefits of both the conservative and non-conservative approaches, we also adopted a *flexible* approach which reflects both the conservative and the non-conservative together. The window-based approach is an example of history-aware algorithms that are flexible and can make use of previously known data. Figure 5.2 shows a summary of the proposed algorithms for the online estimation of the parameter.

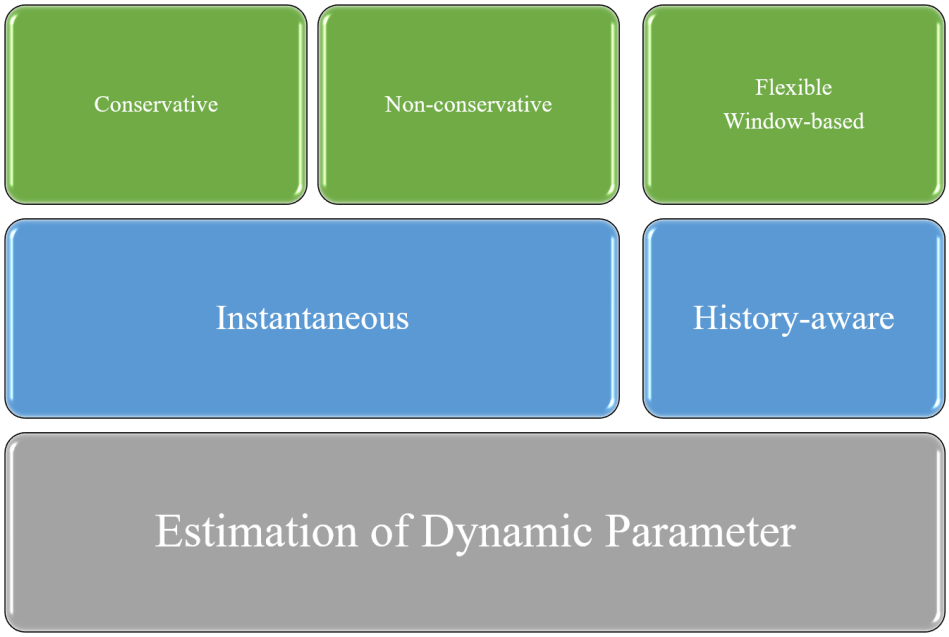



Figure 5.2: Algorithm classification for dynamic parameter estimation

## 5.4 Instantaneous Approach

The instantaneous approach estimates the value of the parameter $\alpha$ based on the current value of the number of SLAVs and its difference from the SLAVs threshold. The conducted experiments show that the conservative version of the instantaneous approach is inefficient in the environments where the workload indicates very high levels of fluctuations which results in higher levels of SLAVs. Using only a conservative approach that slowly adjusts the value of the parameter $\alpha$ won't enable the cloud provider to achieve his goals efficiently. Therefore, we ignored the conservative instantaneous approach and considered a non-conservative one that can promptly adjust the rate of increase or decrease of the allocated resources (in this work, we are only considering CPU resources) to adapt to high workload fluctuations. Thus, we hope to reduce the number of SLAVs whenever this number exceeds the SLAVs threshold by increasing the value of $\alpha$. Conversely, the value of the parameter $\alpha$ is reduced when the number of current SLAVs is lower than the SLAVs threshold. The amount of change in the value of $\alpha$ should drive efficient PM utilization without exceeding the predefined SLAVs threshold.

The instantaneous approach estimates a new value of the parameter $\alpha$ based on the difference between the current number of SLAVs and the SLAVs threshold (which we call it the *distance*). The distance is the absolute value of the difference between the

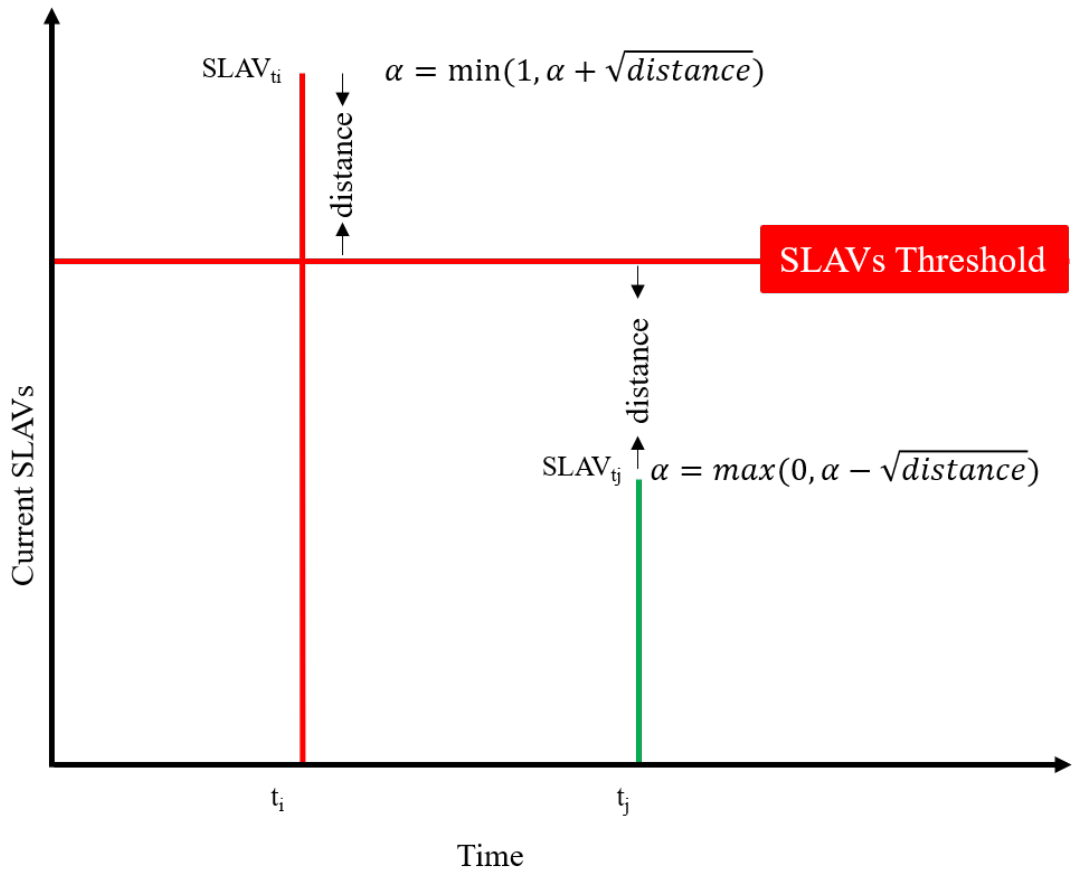



Figure 5.3: Non-conservative instantaneous approach to estimate the parameter (α)

SLAVs threshold and the current SLAVs as shown in Equation 5.1. To give a slight boost when this difference is close to zero, we take the square root of the difference; this seems to enable the algorithm to adapt promptly and makes it less sensitive to the initial value of the parameter α.

$$distance = |thresholdSlavs - currentSlavs| \tag{5.1}$$

Figure 5.3 shows how does the non-conservative instantaneous approach estimates the value of the parameter α based on the distance between current SLAVs and the SLAVs threshold. At time $t_i$ in Figure 5.3, the current SLAVs are higher than the SLAVs threshold, and therefore the instantaneous approach increases the value of α using the square root of the distance bounded by one (as the amount of allocated resources is limited by the reserved VM size). On the other hand, at time $t_j$, the current SLAVs ($SLAV_{tj}$) are less than the SLAVs threshold, and hence the value of the parameter α is

decreased bounded by zero (as the minimum allocated resources should be based on the demand) to reduce resource wastage and improve energy consumption.

**Algorithm 2** Estimating $\alpha$ for use by a non-conservative instantaneous approach

1: currentSlavs $\leftarrow$ getCurrentSlavs();
2: thresholdSlavs $\leftarrow$ The non-exceeding SLAVs threshold;
3: $\alpha \leftarrow$ get current value of $\alpha$;
4: distance $\leftarrow |thresholdSlavs - currentSlavs|$
5: **if** (currentSlavs $\geq$ thresholdSlavs) **then**
6: $\quad \alpha = \min(1, \alpha + \sqrt{distance})$;
7: **if** (currentSlavs $<$ thresholdSlavs) **then**
8: $\quad \alpha = \max(0, \alpha - \sqrt{distance})$;
9: return $\alpha$;

Algorithm 2 shows the computation of the value of the parameter $\alpha$ using the square root of the absolute distance between the current number of SLAVs and the SLAVs threshold. Algorithm 2 is an example of a non-conservative approach as it tries to promptly change the value of $\alpha$ with the help of the square root ($distance \in (0,1)$, so $\sqrt{distance} > distance$). Line 1 of Algorithm 2 computes the current value of SLAVs, and Line 2 defines the SLAVs threshold. Then, the algorithm reads the current value of the parameter $\alpha$ and computes the distance as the absolute difference between the SLAVs threshold and the current value of SLAVs. In Lines 5 to 8, the algorithm adjusts the value of the parameter $\alpha$. It increases $\alpha$ if current SLAVs are greater than the SLAVs threshold and vice versa (as shown in Lines 6 and 8 respectively). Finally, the algorithm returns the estimated value of $\alpha$ as shown in Line 9. Algorithm 2 takes a constant time regardless of the number of inputs, so the computational complexity is $O(1)$.

## 5.5 Window-based Approach

As previously introduced, the instantaneous approach only cares about the current value of the SLAVs as opposed to the SLAVs threshold. This means that we might be missing useful information that may lead to a better adjustment to the value of the parameter $\alpha$. For example, the frequency of adjusting the parameter $\alpha$ in the instantaneous approach is high. Additionally, the adjustment decision is triggered based on the current value of SLAVs as opposed to the SLAVs threshold.

On the contrary, the window-based approach is history-aware in that it makes use

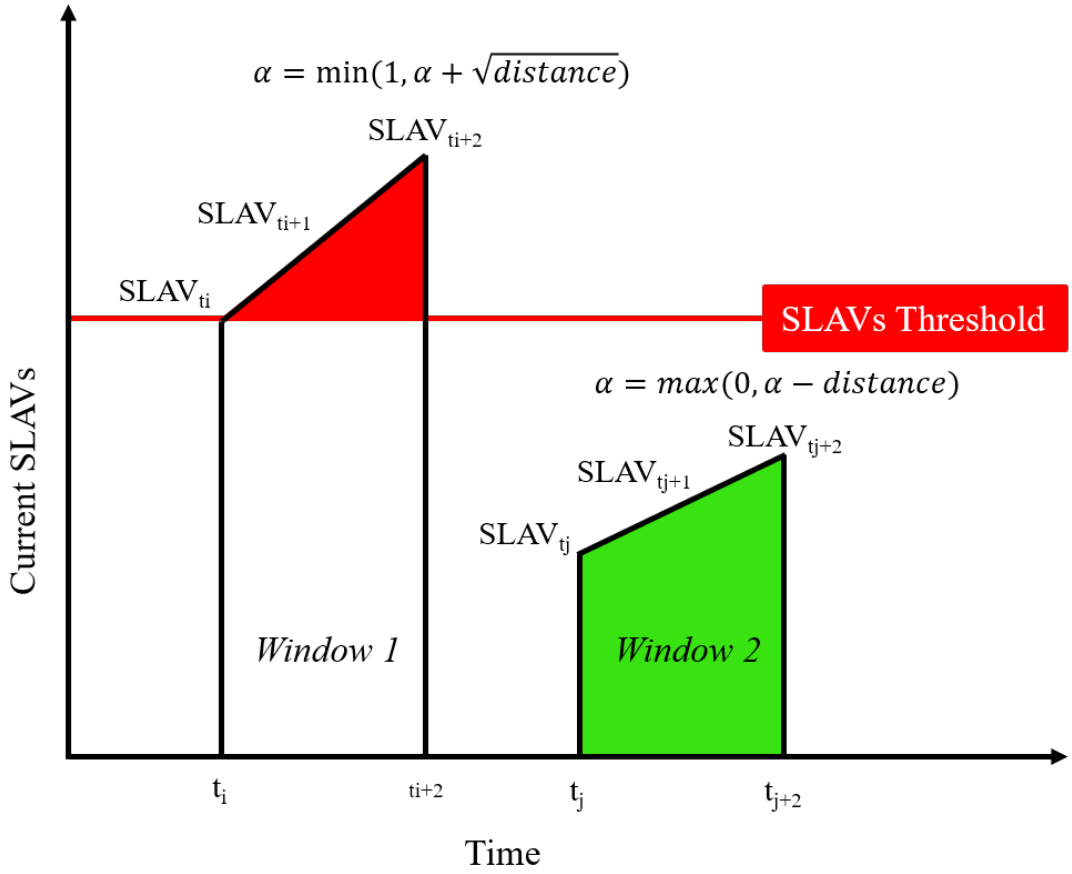



Figure 5.4: Increasing trend - using a window of SLAVs history to estimate α

of some of the previous values of the number of SLAVs in addition to the current value of the SLAVs. Thus, it looks through a window of a specified size into the history of SLAVs to identify a trend (if there is one) with the preceding SLAVs (for example, whether the number of SLAVs is increasing or decreasing) in order to help estimate a more informed and fine-grained value of the parameter α. Using such a window entails that the parameter α is only adjusted (if deemed necessary) after a certain amount of time determined by the window size and the interval time). This span between assessments to take actions tend to make the solution more robust so as not to react to sudden changes that may not last for a long time. As an illustration, in Figure 5.4, the number of SLAVs at $t_{i+2}$ ($SLAV_{ti+2}$) is higher than the SLAVs threshold, and the number of SLAVs is increasing over Window 1, which is the area between $t_i$ and $t_{i+2}$. Consequently, we should increase the allocated resources by increasing the value of α at a high rate to reduce the number of SLAVs as soon as possible. Nevertheless, when the current number of SLAVs is lower than the SLAVs threshold, and the number of

SLAVs is increasing, as shown over Window 2 in Figure 5.4, then we should decrease the value of the parameter α by a small rate to ensure less energy consumption through better utilization without exceeding the SLAVs threshold. Figure 5.4 confirms that the window-based approach will not take action before a specific time (the time imposed by the window size). For example, the value of α is not changed at time$_{i+1}$ and time$_{j+1}$ even though the SLAVs are increasing.

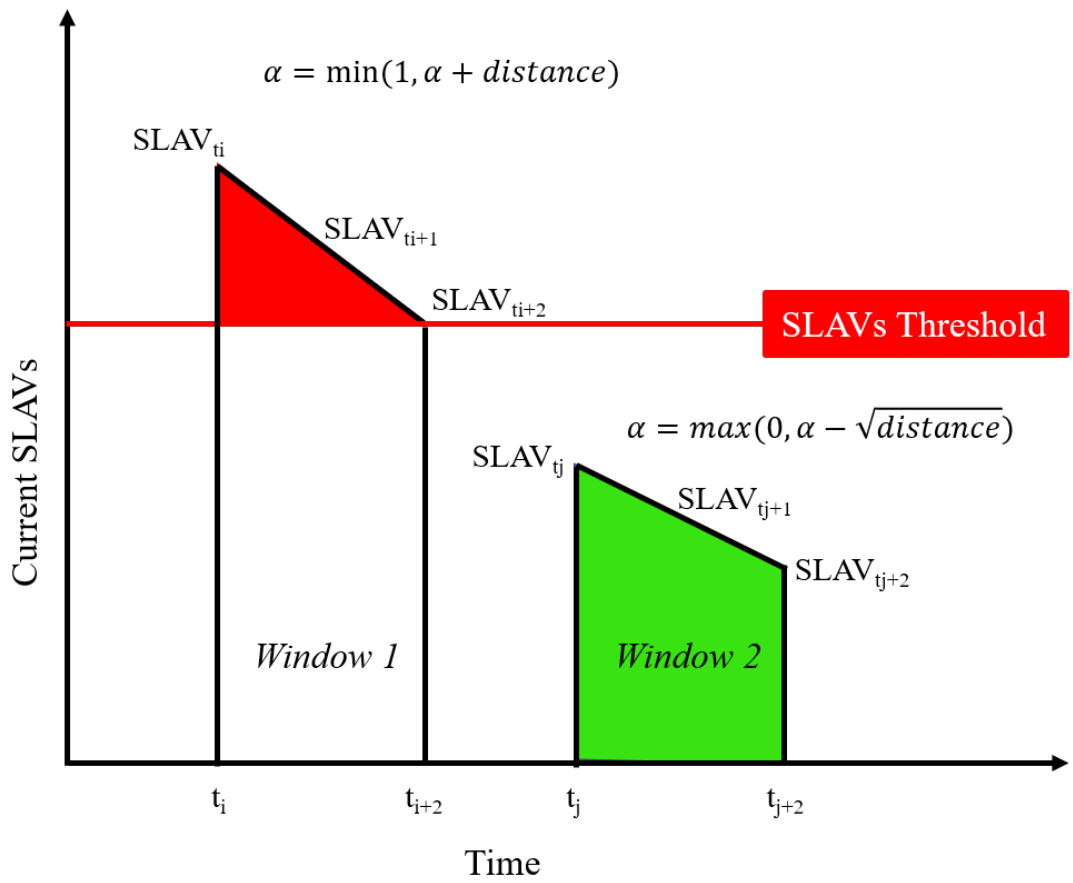



Figure 5.5: Increasing trend - using a window of SLAVs history to estimate α

Figure 5.5 shows a decreasing trend over the specified windows and how does the value of the parameter α is estimated accordingly. For example, when the current SLAVs are higher than the SLAVs threshold and the window is decreasing, then the value of α is increased by a smaller amount (as shown over Window1). However, when the current SLAVs are less than the SLAVs threshold, and the window is decreasing, then the value of the parameter α might be reduced by a high amount to ensure efficient utilization (as shown over Window2).

Algorithm 3 describes the window-based approach that makes use of the previous

**Algorithm 3** Estimating $\alpha$ using a window-based approach

```
1:  currentSlavs ← getCurrentSlavs();
2:  thresholdSlavs ← The non-exceeding SLAV threshold;
3:  α ← get current value of α;
4:  distance ← |thresholdSlavs − currentSlavs|
5:  slavHistory[] ← stores previous SLAVs values;
6:  slavHistory.add(currentSlavs);
7:  windowSize ← set window size;
8:  window[] ← selectedHistory(slavHistory, windowSize);
9:  if (currentSlavs ≥ thresholdSlavs) then
10:     for (i = 0; i < len(window); i++ ) do
11:         if (window[i] > window[i+1]) then
12:             α = min(1, α + distance);
13:             return α;
14:         else
15:             continue;
16:     α = min(1, α + √distance);
17: else
18:     for (i = 0; i< length(window); i++) do
19:         if (window[i] < window[i+1]) then
20:             α = max(0, α − distance);
21:             return α;
22:         else
23:             continue;
24:     α = max(0, α − √distance);
25: return α;
```

values of SLAVs to calculate the parameter $\alpha$. Algorithm 3 starts by calculating the current value of SLAV, then it adds it to the history of SLAVs by making use of the *slavHistory* array. After that, the algorithm saves the selected history in the *window* array, which will include the previously stored values of SLAVs for the specified *windowSize*. Then, the algorithm checks whether the number of current SLAVs is greater or less than the SLAVs threshold. When the number of current SLAVs is greater than or equal to the SLAVs threshold, and the number of SLAVs is not increasing, then the algorithm increases the parameter $\alpha$ by a small value as shown in Lines 9 - 12. However, the algorithm increases the parameter $\alpha$ by a higher amount when the number of SLAVs is increasing (in which case we use the square root, as shown in Line 16). On the other hand, when the number of the current SLAVs is less than the SLAVs threshold, and the number of SLAVs is not decreasing, then we reduce $\alpha$ by a small value as shown in Lines 18 - 20. Same as before, the algorithm reduces the parameter $\alpha$ by

a higher value when the number of current SLAVs is less than the SLAVs threshold, and the number of SLAVs is decreasing as shown in Line 24. Algorithm 3 is flexible as it uses both conservative and non-conservative rates for adjusting the current value of the parameter $\alpha$. The computational complexity of Algorithm 3 is $O(n)$ where $n$ is window size (the number of previous SLAVs values to investigate).

## 5.6 Experiments Setup

The details of the simulated CDC are the same as the one previously introduced in Section 4.5.1. In the following experiments in Section 5.7, the initial placement of the VMs as well as the dynamic reallocation of the VMs are based on Algorithm 1 which has been introduced in Chapter 4. The experiments then use the instantaneous and window-based approaches to dynamically estimate the value of the parameter $\alpha$ that guarantees the different SLA requirements instead of using a static value of the parameter $\alpha$.

We have conducted experiments to evaluate the estimation of the parameter $\alpha$ using both the instantaneous and window-based approaches. For the window-based dynamic parameter estimation approach (shown in Algorithm 3), we have used a window of size two to capture the trend of previous SLAVs and check whether the number of SLAVs is increasing or decreasing. For each of the two approaches, we have calculated two performance metrics:

- The amount of energy (in KWh) consumed by all PMs to allocate the VMs without exceeding the predefined SLAVs threshold.
- The resulting value of SLAVs.

To run the two algorithms for estimating an online value of the parameter $\alpha$, we have to set an initial value for the parameter $\alpha$ in both algorithms. The choice of the initial values of $\alpha$ is as follows:

- The experiments run using four different initial values for the parameter $\alpha$, from the following list.

$$\text{Initial values of } \alpha : [0.2, 0.4, 0.6, 0.8].$$

  The reason for choosing such values is to make sure they can cover the majority

of the range between zero (demand-based), and one (reservation-based). Choosing these different values enable us to test the sensitivity of each algorithm to each different value of the initial parameter $\alpha$.

As in Chapter 4, the two algorithms for the online estimation of $\alpha$ have been tested using the same three types of workload traces (synthetic, normal PlanetLab, and skewed PlanetLab). From the cloud provider's point of view, we need to set SLAVs thresholds that should not be exceeded; therefore, we have tested six different values of SLAVs thresholds as follows:

- SLAVs thresholds for the synthetic traces:
  The list of the chosen SLAVs thresholds for synthetic traces are as follows:

  SLAVs thresholds: [0.1, 0.05, 0.01, 0.005, 0.003 and 0.001].

  The reason for starting the SLAVs thresholds with the value 0.1 is because the SLAVs resulting from the demand-based placement (i.e., $\alpha = 0$, when the number of SLAVs is expected to be highest) is 0.1063.

- SLAVs thresholds for both normal and skewed PlanetLab traces:
  The list of the chosen SLAVs thresholds for the PlanetLab traces are as follows:

  SLAVs thresholds: [0.05, 0.03, 0.01, 0.005, 0.003 and 0.001].

  The reason for starting with a value of 0.05 as the SLAVs resulting from demand-based VM placement is 0.0542.

The dynamic VM placement uses algorithms to detect hot (overutilized) and cold (underutilized) PMs to take the proper action. For the detection of hot PMs, these experiments have used a static overutilization threshold of CPU utilization; a PM is considered to be hot (overutilized) when CPU utilization is 100%. Whenever the parameter-based VM placement algorithm detects a hot PM, it chooses to migrate VMs with the minimum memory from the hot PM to minimize the total migration time which reduces the overall SLAVs. After migrating VMs from hot PMs, the algorithm tries to migrate all VMs from cold PMs by starting by the least underutilized PMs whenever possible. The simulation of the experiments runs for a full day, and the dynamic reallocation of the VMs takes place every five minutes of simulated time (scheduling interval).

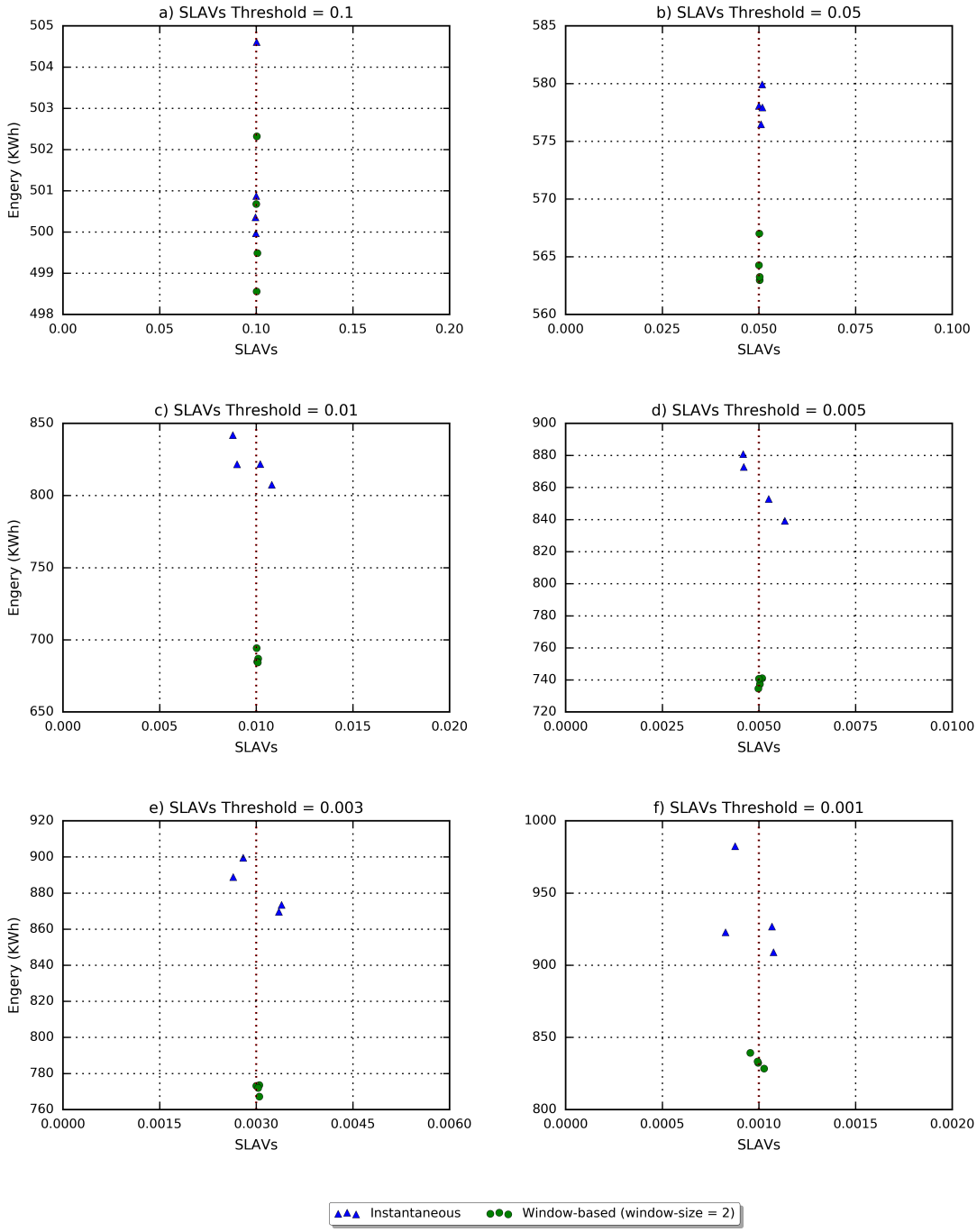



Figure 5.6: Energy to SLAVs using the synthetic traces

## 5.7 Experimental evaluation

The following three subsections describe and analyse the results from three experiments. Each one of these conducted experiments evaluates the instantaneous and

window-based approaches using one of the three types of workloads (synthetic, normal PlanetLab, and skewed Planetlab traces).

### 5.7.1 Experiment 1: Using Synthetic Traces

Figure 5.6 shows the amount of energy consumed and the resulting SLAVs under the specified SLAVs threshold by running both the instantaneous and the window-based approaches using the synthetic workload traces. In each subgraph of Figure 5.6, the X-axis represents the resulting SLAVs, and the Y-axis represents the amount of energy consumed to allocate the VMs to comply with the specified SLAVs threshold. The red dotted vertical line represents the SLAVs threshold; this experiment and the next ones assume that the cloud provider has specified a different SLAVs threshold in each of these subgraphs. Each point represents energy consumption to the number of SLAVs (cf. Equation 4.13 in Chapter 4) using a different initial values of the parameter $\alpha$ (initial $\alpha$: 0.2, 0.4, 0.6 and 0.8).

By and large, Figure 5.6 shows that the window-based approach (green dots) adheres to the various specified SLAVs thresholds in all the six subgraphs with more savings of energy consumption as opposed to the instantaneous approach (blue rectangles). Nevertheless, Figure 5.6(d), (e) surpassed the predetermined SLAVs threshold with an insignificant value. Additionally, Figure 5.6 demonstrates that the window-based approach performs better than the instantaneous approach as it can meet the SLAVs threshold while consuming less energy; which becomes more obvious as the cloud provider decreases the predefined SLAVs threshold. However, for high values of SLAVs thresholds, as shown in Figure 5.6(a), both approaches are somewhat sensitive to the initial value of the parameter $\alpha$ as shown in the spread of the plotted points across the Y-axis. Finally, Figure 5.6(b), (c), (d), (e) and (f) show that the window-based approach becomes less sensitive to the initial value of $\alpha$ as the SLAVs threshold gets smaller, in contrast to the instantaneous approach.

### 5.7.2 Experiment 2: Using Normal PlanetLab Traces

Experiment 2 aims to examine the behaviour of both the instantaneous and window-based approaches using real workload traces from PlanetLab instead of the synthetic ones. Figure 5.7 shows the consumed energy and the resulting SLAVs from adopting the two approaches using normal PlanetLab traces. As concluded from Experiment 1, Figure 5.7 still confirm that the window-based approach usually performs better than

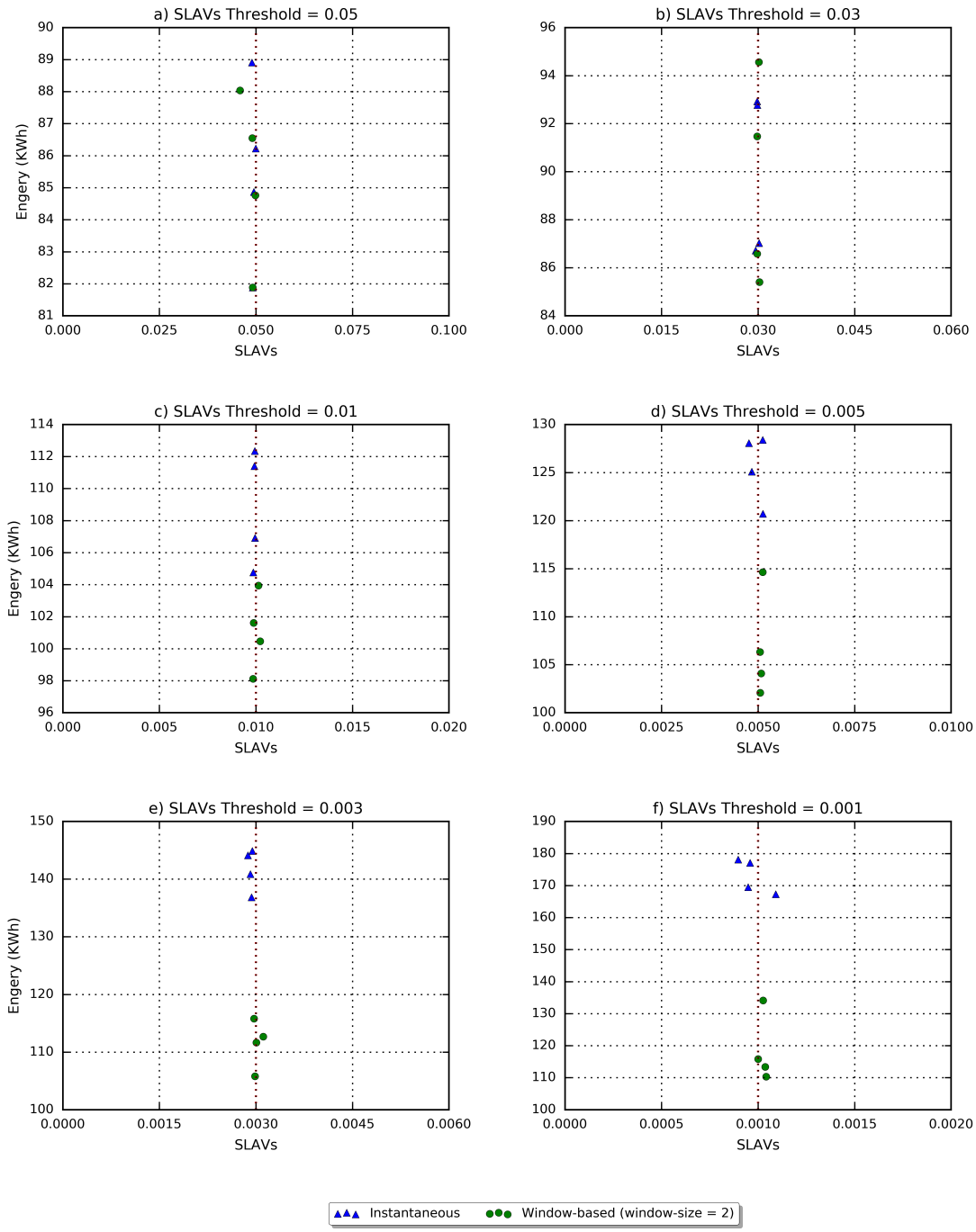



Figure 5.7: Energy to SLAVs using normal PlanetLab traces

the instantaneous one as it can adhere to the SLAVs threshold with lower energy consumption. Moreover, both approaches do not exceed the SLAVs threshold except for the insignificant value of exceeding the SLAVs threshold in Figure 5.7(f). As opposed to the synthetic traces, the normal PlanetLab traces show somewhat higher sensitivity

to the initial value of the parameter $\alpha$ as the PlanetLab traces have higher variability in the data. Furthermore, the results reported from Experiment 2 suggest that the window-based approach produces better outcomes; by showing less sensitivity to the initial value of the parameter $\alpha$ when the SLAVs threshold is set to smaller values. However, it shows higher sensitivity for higher values of SLAVs thresholds. All in all, both experiments demonstrate that, as expected, the window-based approach can estimate the parameter $\alpha$ in a way that provides better results.

### 5.7.3 Experiment 3: Using Skewed PlanetLab Traces

Finally, Experiment 3 examines the results from the non-conservative instantaneous approach and the window-based one using the skewed PlanetLab traces. Generally, the window-based approach can estimate a parameter that results in more efficient PM utilization while adhering to the specified SLAVs threshold, as shown in Figure 5.8. Furthermore, Figure 5.8 confirms that the window-based approach becomes less sensitive to the initial value of the parameter $\alpha$ when the required SLAVs threshold is reduced (becomes near to zero). Experiments 1, 2, and 3 confirm that a cloud provider can adopt both instantaneous or window-based approaches to dynamically estimate the value of the parameter $\alpha$ based on a predefined SLAVs threshold. In summary, these experimental results show that adopting a window-based approach for the online estimation of the parameter returns a more fine-tuned parameter that efficiently utilize the PMs while not exceeding the specified SLAVs thresholds.

## 5.8 Discussion

Both the instantaneous and window-based approaches proved to dynamically estimate the value of the parameter $\alpha$ on the fly. This online estimation of the parameter $\alpha$ offers more flexibility for CDC management in achieving business-related objectives such as setting an SLAVs threshold that should not be exceeded. One of the issues that emerges from adopting these approaches is that both methods use the same value of the parameter $\alpha$ for all VMs (a single $\alpha$ for the whole CDC for each scheduling interval). Setting $\alpha$ the same for all VMs could waste energy as some VMs might have a higher value of $\alpha$ even though that VM may be idle or low utilized. If the parameter-based VM placement solution can use a small value of the parameter for the underutilized VMs, then this might lead to more efficient utilization (decreasing the value of $\alpha$ reduces

the added slack to the demand which minimizes underutilization). Consequently, the following chapter (Chapter 6) tries to ensure more efficient utilization by adjusting the value of the parameter per each VM whenever it is expected to be useful.

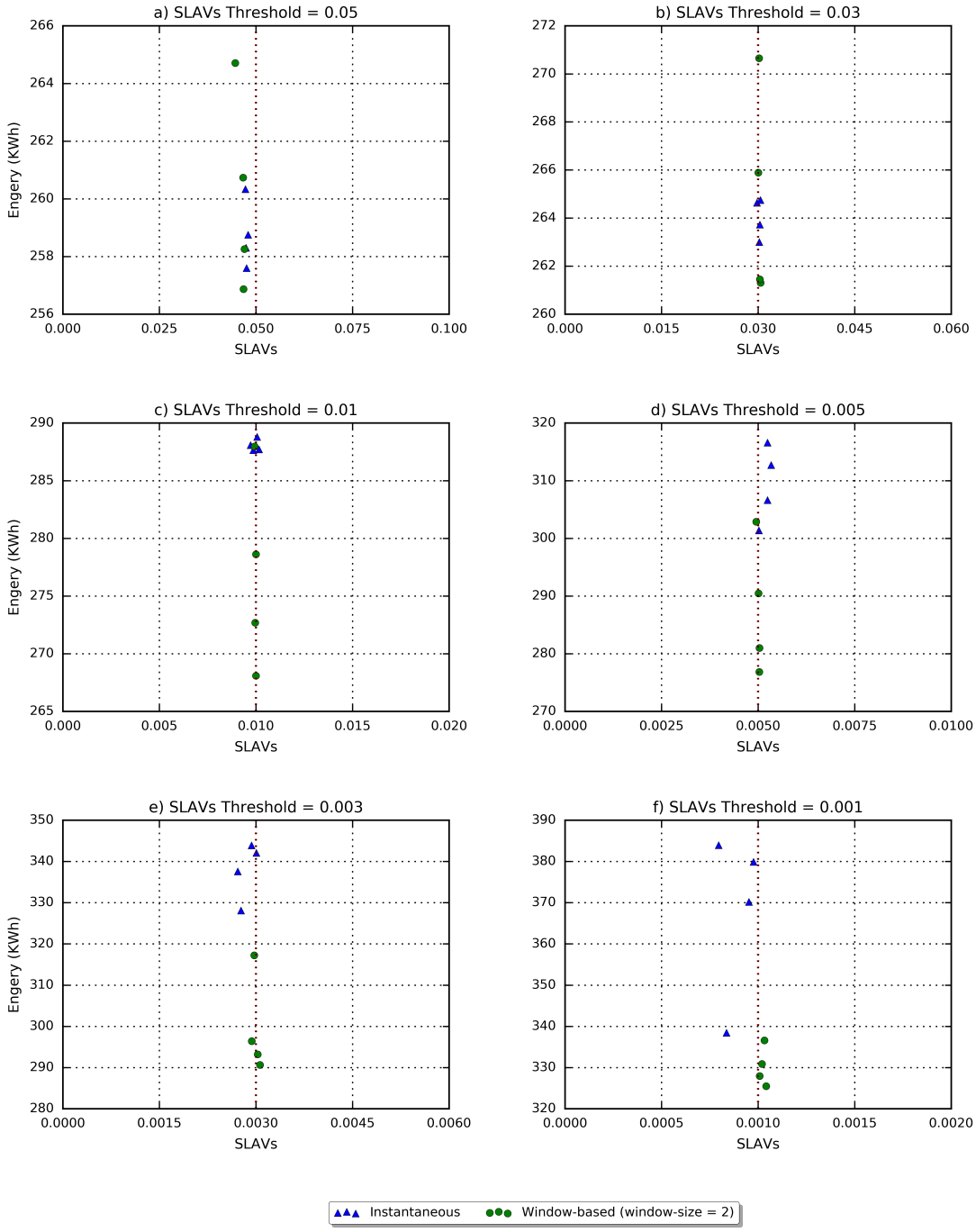



Figure 5.8: Energy to SLAVs using skewed PlanetLab traces

## 5.9 Summary

This chapter demonstrated the need for dynamically estimating the value of the parameter $\alpha$ of the proposed parameter-based VM placement solution on the fly. The reason for this is that a statically predefined, fixed value of the parameter $\alpha$ cannot guarantee, due to workload fluctuations, that the number of SLAVs is less than a predefined SLAVs threshold (unless the parameter $\alpha$ is set to one, or close to one). The estimation of the parameter $\alpha$ on the fly has the objective of not to exceed a certain SLAVs threshold while achieving better utilization.

Next, the chapter explained that there are two general approaches for the online estimation of the parameter, namely, instantaneous and window-based. The instantaneous approach adjusts the value of the parameter $\alpha$ immediately based on the distance between the number of current SLAVs and the predefined SLAVs threshold. The window-based approach makes use of the recent history of the number of SLAVs before making the appropriate decision concerning the estimation of the parameter $\alpha$. By and large, the two proposed approaches managed to adhere to the SLAVs threshold. However, the experimental evaluation confirms that the window-based approach outperforms the instantaneous one by meeting the SLA with better PM utilization and energy savings. The performance gain of the window-based approach may be attributed to: (1) the more sophisticated strategy used to select the rate with which to increase or decrease the value of the parameter $\alpha$, (2) and perhaps due to the mix between the conservative and non-conservative choices in the window-based algorithm.

# Chapter 6

# Parameter tuning using a Hybrid Approach

## 6.1 Introduction

In Chapter 5, we have introduced reactive instantaneous and window-based approaches for the dynamic estimation of the parameter $\alpha$. These approaches are reactive as they adjust the value of the parameter $\alpha$ based on the SLAVs threshold compared to the current and previous values of SLAVs, respectively. The value of the estimated parameter $\alpha$ might change for each scheduling interval; however, it is the same for all VMs in the cloud data centre (CDC). Determining the parameter $\alpha$ for each VM (a workload-aware parameter) as opposed to establishing a CDC-wide parameter could result in more favourable results in terms of more efficient utilization of the PMs while adhering to the SLA requirements. Therefore, this chapter focuses on the online estimation of the parameter $\alpha$ for each VM on each scheduling interval using a hybrid approach.

This chapter presents a hybrid approach (both reactive and proactive) for parameter estimation. The proposed hybrid approach is reactive as it reacts to the current state of the VMs-to-PMs mapping based on the present SLAVs and the SLAVs threshold. Moreover, it is proactive as it adjusts the value of the parameter based on the predicted CPU utilization of the individual VMs. To adopt this hybrid approach, the parameter-based VM placement solution should be able to forecast the future workload for each VM and act accordingly. Therefore, this chapter demonstrates the application of two techniques for time series prediction to predict the VMs' CPU utilization for

the proposed hybrid approach of parameter estimation. These two techniques are random forests (RF) and a recurrent neural network model using long short-term memory (LSTM).

## 6.2 A Hybrid Approach for Parameter Estimation

If a cloud provider can estimate the utilization of a VM for the next scheduling interval(s), then he can make use of this knowledge to reduce the value of the parameter whenever he is reasonably sure that future utilization is going to be less than the current one. Reducing the value of the parameter $\alpha$ will minimize the amount of allocated resources for a VM, which can ensure more efficient utilization in order to save more energy. In such a scenario, the parameter-based VM placement solution will need to adopt two parameters as follows:

1. The first parameter is a SLAVs threshold at the CDC level, whose purpose is to ensure that acquired SLA requirements are observed as much as possible. We use $\alpha_{cdc}$ to refer to this parameter which is the same parameter that has been previously introduced in Chapter 5.

2. The second parameter is for the VM level, where each VM could have a different value of that parameter to ensure more efficient utilization. We use $\alpha_{vm}$ to refer to the parameter for the VM level. The maximum value of the $\alpha_{vm}$ is bounded by the maximum value of the $\alpha_{cdc}$. Estimating $\alpha_{vm}$ is the focus of this chapter with the help of time-series prediction techniques.

As an illustration, Figure 6.1 shows an example of adjusting the value of a VM's parameter $\alpha_{vm}$ based on the predicted CPU utilization in the following interval(s). In Figure 6.1, the current CPU utilization of a VM at time $t_i$ ($current_{ti}^{cpu}$) is higher than the expected CPU utilization of that VM at time $t_{i+1}$ ($predicted_{ti+1}^{cpu}$), so increasing the value of parameter is not useful and just wastes resources. Therefore, we can set the value of $\alpha_{vm}$ to a minimum value (zero as an example). On the other hand, if the current CPU utilization of a VM at time $t_j$ ($current_{tj}^{cpu}$) is expected to be lower than the predicted CPU utilization of that VM at time $t_{j+1}$ ($predicted_{tj+1}^{cpu}$), then it may be better to set the value of $\alpha_{vm}$ to the same value of $\alpha_{cdc}$ to make sure that it sticks to the SLAVs threshold. These are just examples, but the principal idea is that the cloud provider can adjust the value of the $\alpha_{vm}$ with the help of the predicted utilization to ensure higher

utilization of the CDC's VMs. The following section describes the algorithm used for estimating the parameter per each VM ($\alpha_{vm}$).

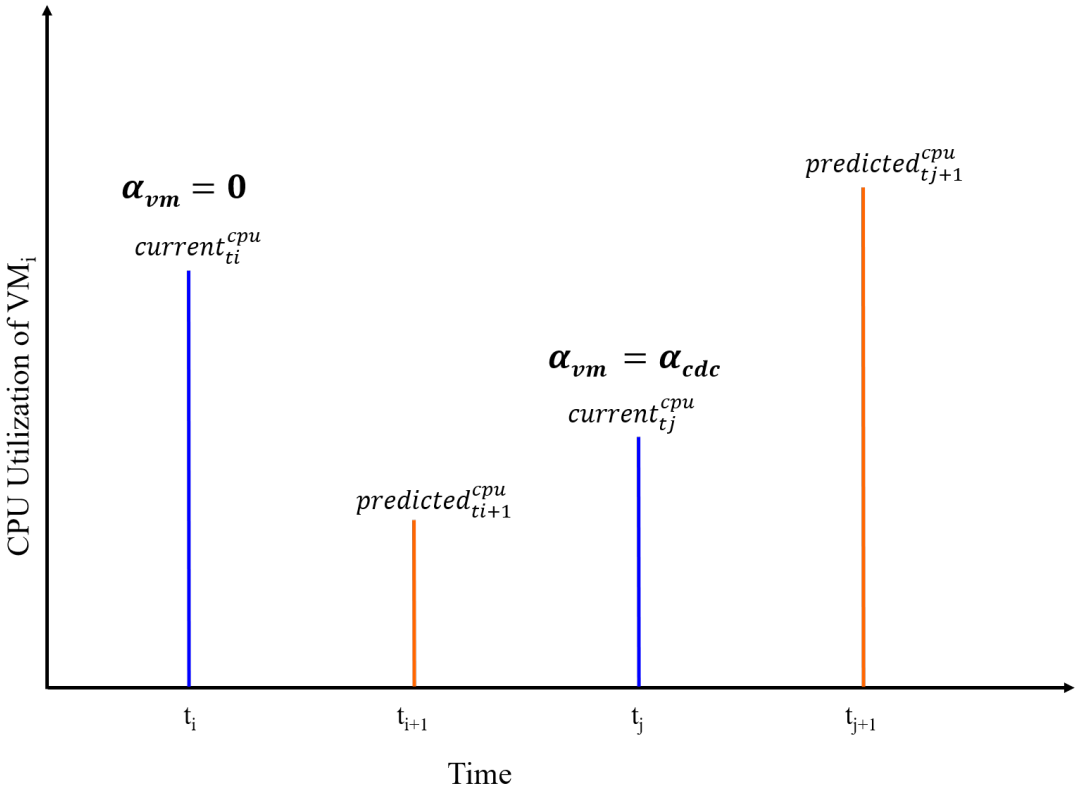



Figure 6.1: Parameter per VM based on current and predicted CPU utilization

## 6.3 Parameter Estimation for VM Level

Algorithm 4 estimates and sets the value of parameter $\alpha_{vm}$ of each VM by considering the predicted CPU utilization and the current value of the CDC's parameter $\alpha_{cdc}$. Line 2 of Algorithm 4 estimates the value of $\alpha_{cdc}$ using the window-based approach which was previously introduced in Algorithm 3. After that, the algorithm creates an array to store the $n$ predicted CPU utilizations of each single VM. Then, the algorithm loops over each VM in the list of VMs to set the value of each VM's parameter $\alpha_{vm}$. Algorithm 4 stores the VM's current CPU utilization, and then predicts the next $n$ CPU utilizations of the current VM. We have decided to compare the current CPU utilizations to the future $n$ predicted values (and not just the next predicted value) to try to mitigate prediction errors and to make the algorithm more robust by not responding to

single sudden spikes. Checking these $n$ predicted utilizations could make the changes of the $\alpha_{vm}$ more robust and reduce the possibility of more SLAVs situations. Then, Algorithm 4 compares the current value of CPU utilization to the array of the predicted values of the next $n$ scheduling intervals as shown in Line 8. If the maximum value of the array of the $n$ predicted CPU utilizations is expected to be less than the current CPU utilization, then it may be a good opportunity to reduce the value of the parameter $\alpha_{vm}$ to ensure better resource utilization (the algorithm sets to zero as shown in Line 9). Setting $\alpha_{vm} = 0$ when the algorithm is confident that the future CPU utilizations are less than the current minimizes the underutilization per VMs. However, if the current CPU utilization is lower than or equal to the maximum of the array of the $n$ predicted utilizations, then the algorithm will set the value of the parameter $\alpha_{vm}$ to the same value of $\alpha_{cdc}$ to ensure that the resulting SLAVs do not exceed the predefined SLAVs threshold as shown in Line 11. Algorithm 4 only reduces the value of the $\alpha_{vm}$ whenever it is expected that the predicted utilization over the $n$ future intervals is lower than the current interval. By this, we can ensure the same level of SLA requirements (when $\alpha_{vm}$ is set to the current value of $\alpha_{cdc}$), while ensuring better utilization when $\alpha_{vm}$ is reduced to a minimum value. The computational complexity of Algorithm 4 is O($mn$) where $m$ is the number of VMs, and $n$ is the number of predicted CPU utilizations that the algorithm needs to check.

**Algorithm 4** Estimating parameter ($\alpha$) per VM based on predicted utilization

```
1:  vmList ← list of VMs;
2:  α_cdc ← get current value of α_cdc;
3:  α_vm ← set the initial value of α for each VM;
4:  nextNPredictedUtilization[] ← array storing the n predicted future utilization;
5:  for all (vm in vmList) do
6:      currentUtilization = vm.getCurrentUtilization();
7:      nextNPredictedUtilization = vm.getNextNUtilization();
8:      if (max(nextNPredictedUtilization) ≤ currentUtilization) then
9:          α_vm ← 0;
10:     else
11:         α_vm ← α_cdc;
12: Return α_vm;
```

Sections 6.4, 6.5 and describes in detail how the proposed hybrid approach predicts the future CPU utilizations of the VMs with the help of supervised learning prediction techniques starting from data preparation to model evaluation.

## 6.4 Time Series Prediction

We aim to predict the values of CPU utilization of each VM for one or more future scheduling intervals so that we can adjust the value of the parameter of our parameter-based VM placement solution wisely. By making use of the history of CPU utilization, we can predict the future ones, which helps in making better placement decisions. The CPU utilizations of VMs is an example of a time series where CPU usage of each machine is collected at regular intervals in a timely order. There are two types of classes of time-series forecasting methodologies which are summarized as follows:

1. *Classical statistical techniques:* Examples of the classical statistical forecasting approaches include the autoregressive integrated moving average (ARIMA) and exponential smoothing.

2. *Machine learning (ML) and deep learning (DL) techniques:* Example of supervised ML forecasting approaches include random forests (RF) and support vector regression (SVR). An example of a DL approach is the recurrent neural networks (RNN).

The hybrid approach for parameter estimation adopts two supervised learning techniques for predicting CPU utilization namely, RF and LSTM to adjust the parameter $\alpha_{vm}$ accordingly. Predicting CPU utilization is an example of a *supervised* learning process as CPU traces have *features* such as server name, time, and the actual CPU utilization which represents the *target* variable (the one that needs to be predicted). Finally, predicting CPU utilization is a *regression* and not a classification task as the values of the target variable (CPU utilization) are continuous (for classification tasks, target variables are categorical discreet classes).

Recall that the goal of our prediction task is to predict the CPU utilization of the specified VMs over the future scheduling intervals. Regardless of the adopted time-series forecasting technique, the following five steps summarize the main tasks that have been followed to predict the CPU utilization of the VMs:

1. **Acquisition and preparation of CPU traces:**
   For this task, we need to define the source of data and clean that data. After this, we should transform the data in the proper structure. Finally, we split the data into training and testing sets.

2. **Establishing a baseline model:** It is always better to set a baseline model that the adopted prediction model should beat. If the prediction model is worse than

the baseline model, then this is a clear indication that this model is useless and we should find another one. Here, we have chosen to set the baseline model to the baseline error which is the absolute difference between the average utilization and the actual ones. Therefore, the adopted prediction model should at least beat the baseline error.

3. **Model training:**
   In this task, we will train the two models (RF and LSTM) using the training set and setting the required parameters.

4. **Making predictions:**
   Once the model has been trained to learn the relationship between features and targets, we can start to make predictions on the test set.

5. **Performance metrics of predictions:**
   Finally, we evaluate the performance of the prediction models using performance metrics and perform hyperparameter optimization if required to improve the results of the prediction model. Hyperparameters are parameters that configured externally by the users of the model and can not be estimated from the data. They are called hyperparameters to differentiate it from the model parameters that are internal to the model and whose values can be estimated from the data. Examples of hyperparameters include the *number of layers* and *batch size* in an artificial neural network (ANN) or the $k$ in K-nearest neighbour (k-NN).

Tasks (1) and (2) are the same for both RF and LSTM, so we are going to discuss them in the following subsections. Then, we are going to demonstrate tasks (3, 4, 5) under Subsections 6.5.1 and 6.5.2.

### 6.4.1 Acquisition and Preparation of CPU Traces

The majority of machine learning models are profoundly affected by the amount of data available. As previously introduced in Chapter 3, there is a limited number of publicly available cloud data sets. The available PlanetLab traces contain server utilization throughout ten days in the period between the $3^{rd}$ of Mach 2011, and the $20^{th}$ of April 2011. The data are in the form of CPU utilization percentage and recorded every five minutes. K-fold cross-validation is a simple and powerful approach for splitting the data into training and test sets. It divides the data into $k$ groups, and iteratively selects one of the groups to be the test set while the remaining $k-1$ to be the training

set [JWHT13]. Empirical evaluations show that choosing a value of five or ten for $k$ can enhance the learning of the model [KJ13]. This means that the test set is usually between 80% and 90%. The available PlanetLab CPU traces are for ten days only, and the aim is to predict the CPU utilization of the VMs on the last day ($20^{th}$ April 2011). Therefore, the first nine days of the PlanetLab traces represent the *training set*, while the last day ($20^{th}$ April 2011) represent the *test set*.

The current CPU traces are organized in files (each server usage data are in one file for each day), and each file has 288 rows where each row is the CPU usage recorded every five minutes of each day. Because this thesis employs supervised ML and DL approaches for the CPU prediction; so, the available CPU usage traces need to be transformed into the proper structure that the prediction model requires. Therefore, the current data structure needs to be converted into a one that consists of input (X) and output (y) components. Based on the provided training set, the prediction model can learn how to map inputs (X) to outputs (y).

$$y = f(X).$$

For example, if we have six points of CPU utilization values as follows:

$$\text{CPU utilization} = [12, 6, 17, 82, 67, 77].$$

These data points will be transformed into the proper structure for the supervised ML/DL technique as follows:

| **Input(X)** | **Output(y)** |
|---|---|
| [12, 6, 17, 82] | 67 |
| [6, 17, 82, 67] | 77 |

On this new structure, each output (y) depends on several previous steps, which are known as the *lag* observations. In the previous example, we suppose that the lag is four, so each output (y) is estimated based on the four previous CPU utilization traces. Thus, the question is how we can convert the existing CPU traces into samples of input and output components? Instead of preparing it manually, the deep learning library *Keras*[1] provides a *TimeseriesGenerator* class that can automatically convert the univariate CPU usage traces into batches of temporal data that are suitable to train the supervised ML/DL model.

[1] https://keras.io/

One of the useful exploratory data analysis tools before creating a prediction model is to figure out the correlation between the current CPU traces. Therefore, we employ the autocorrelation function (ACF) and the partial autocorrelation function (PACF) to analyse the correlation among the CPU traces. The following subsection plots and presents the ACF and PACF of the PlanetLab traces.

**Autocorrelation and Partial Autocorrelation Functions**

Both autocorrelation function (ACF) and partial autocorrelation function (PACF) are measures of association between current and past values of time-series data. ACF and PACF are handy measures for prediction as they indicate which past series values are most valuable in predicting future values. On the one hand, an ACF can measure the association between time series observations by estimating a correlation coefficient at certain lags of the series. As an illustration, *lag-1* autocorrelation measures the correlation between $y_t$ and $y_{t-1}$; *lag-2* autocorrelation measures the correlation between $y_t$ and $y_{t-2}$ (two periods apart) and so forth. On the other hand, a PACF at lag $x$ measures the correlation between series values and its lagged values after removing the effect of in-between $x$ lags.

Figure 6.2 exhibits the ACF and PACF of the CPU utilization of PlanetLab traces. In both plots, the x-axis represents the lag, and the y-axis represents the autocorrelation (usually between -1 and 1). The autocorrelation at lag zero is always one, as each value perfectly correlates with itself. In the autocorrelation plot, a spike at lag 1 indicates a correlation of around 0.7 between each series value and the preceding value; also, the peak at lag 2 shows a strong correlation between each value and the value of the two previous points, and so on. The ACF plot indicates that there is a strong autocorrelation component, whereas the PACF plot suggests that this component is distinct for the first 10 to 20 lag observations, approximately. Figure 6.2 suggests that a good starting size of a window would be between 10 and 20.

### 6.4.2 Establishing a Baseline Model

The mean absolute error (MAE) is used as the baseline error that the prediction model should outperform to be considered valuable. The baseline error is the absolute difference between the average CPU utilization and the actual values of CPU utilization, as shown in Equation 6.1.

$$MAE(y,\bar{y}) = \frac{1}{n}\sum_{l=1}^{n} |\bar{y} - y| \tag{6.1}$$

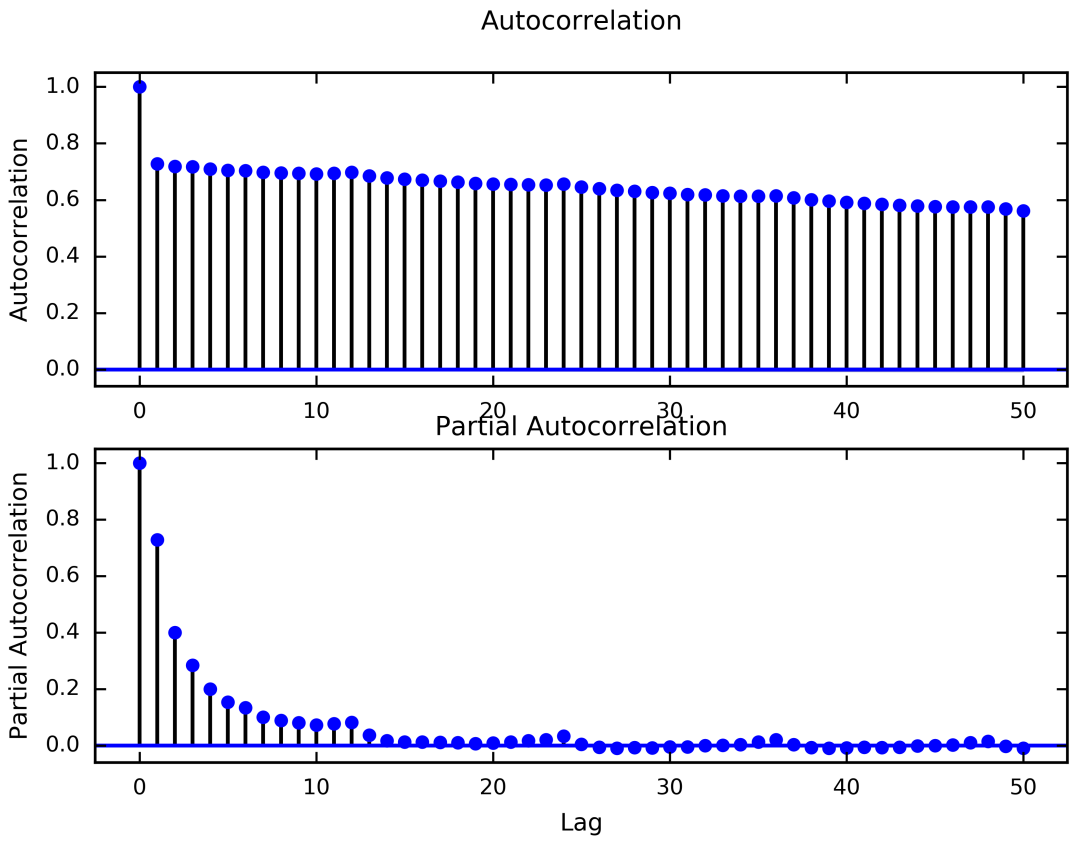



Figure 6.2: ACF and PACF

Here $n$ is the number of CPU traces, $y$ is the actual CPU utilization, and $\bar{y}$ is the average CPU utilization. Based on the test dataset of PlanetLab traces, we have estimated the baseline error using the MAE.

$$BaselineError = 10.31.$$

This means that the MAE resulting from the adopted prediction model should be less than 10.31; otherwise, it is useless. If the MAE of the prediction model is greater than the baseline error, then that model is considered worthless. Once the baseline error has been estimated, the next step is to start the model training process.

## 6.5 Model Training and Making Predictions

This chapter adopted two supervised learning algorithms, namely RF and LSTM. The following subsections demonstrate these two algorithms in detail, starting from model

training and making predictions to the performance evaluation of the resulting model.

### 6.5.1 Random Forest

Random forest (RF) is one of the *ensemble learning* methods that generate and combine multiple classifiers and aggregate their results, which improves the performance of the model [LW+02]. RF is an example of a robust supervised machine learning algorithm that is introduced by Leo Breiman [Bre01], depending on decision trees (DTs). Put simply, a RF is a collection of many randomized independent DTs, and it combines the predictions from these multiple DTs (which will have a broader scope of information) to make a better prediction [LW+02]. Each DT in the random forest grows using a *bootstrap* version of the sample instead of using the entire sample. A bootstrap version of the sample means that some data points will be missing, and some might be selected more than once (which is known as sampling with replacement). Moreover, random forest adds randomness to the split of each node of the tree, which yields better performance compared to support vector machines (SVM) and neural networks [Bre01]. Random forest is an averaging method of the various independent estimators (DTs). If the RF is solving a *regression* task, then it estimates the mean of the various DTs. On the other hand, RF estimates the highest vote among the DTs if a *classification* task is being solved. Figure 6.3, from [RGSCD+16], shows how a RF can be used for solving a regression problem. Any RF has two important parameters: (1) the number of trees in the forest, (2) number of variables at each node of any tree [LW+02]. Let us assume that $n$ represent the number of samples in the training data set, $f$ is the number of features, and $n_{trees}$ is the number of trees in the random forest. Then, the computational complexity of the RF *training* algorithm for the regression task is $O(n^2 f n_{trees})$. Additionally, the computational complexity of RF *prediction* is $O(f n_{trees})$.

Motivations for using RFs include *easy to use*, *computational efficiency*, *fewer parameters* to tune, and *good performance*. RFs are easy to use and available through many powerful, open-source libraries such as scikit-learn[2]. Additionally, a RF can be grown in parallel as it consists of many disjoint DTs. Furthermore, a RF only requires few parameters to tune, can be applied to a wide range of prediction problems, and have an outstanding practical performance [SBV+15]. Finally, another important feature of RFs is that they can approximate both linear and nonlinear functions.

[2]https://scikit-learn.org/stable/

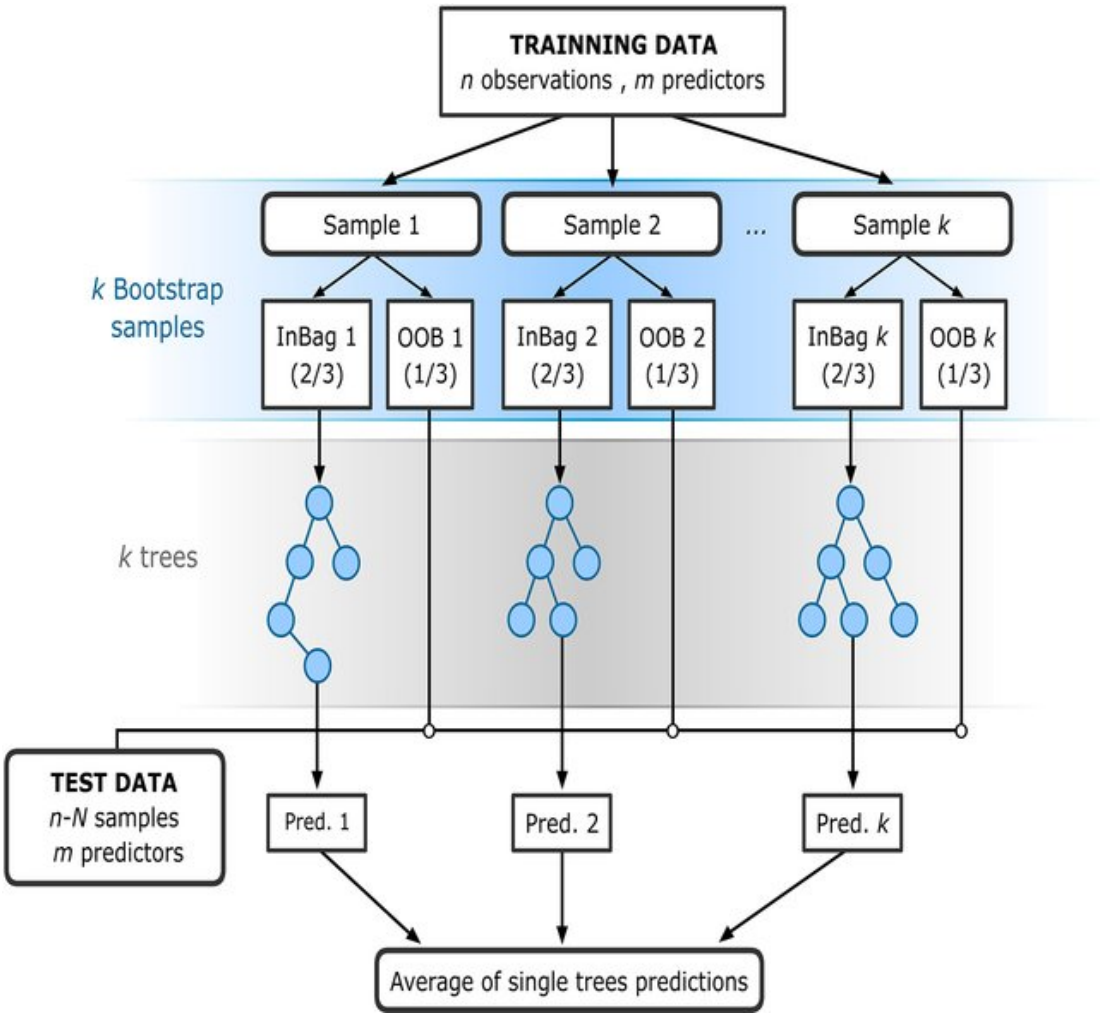



Figure 6.3: Random forest for regression from [RGSCD+16]

### RF: Model Training and Making Predictions

The following set of parameters have been used to train the random forest regression model using scikit-learn library:

- *Number of estimators*: The number of estimators represents the number of decision trees in the RF; 200 estimators have been used for training the RF model.
- *Maximum depth*: This parameter defines the maximum depth of each decision tree; the RF model has been trained using a depth of 20.
- *Criterion*: The quality of each split is measured using a criterion function such as mean squared error (MSE) or mean absolute error (MAE). MSE has been used as the criterion to measure the quality of the split.

- *Random state*: Random number generators require a seed which is represented by the *random state* parameter. The random state has been set to 100.

- *Number of jobs*: This parameter defines the number of jobs to run in parallel; a value of $-1$ has been used, which means to use all available processors.

Following the model training using the training dataset, we have used the model to make predictions of the test data set (which is represented by the last day of PlanetLab traces). Figure 6.4 shows the predicted versus actual CPU utilization from four randomly selected VMs using the random forest. The X-axis represents the time in minutes, and the Y-axis represents the percentage of CPU utilization. Figure 6.4 illustrates that the predicted CPU utilization (using RF) is quite good without too much optimization of the hyperparameters, which means that the model is robust.

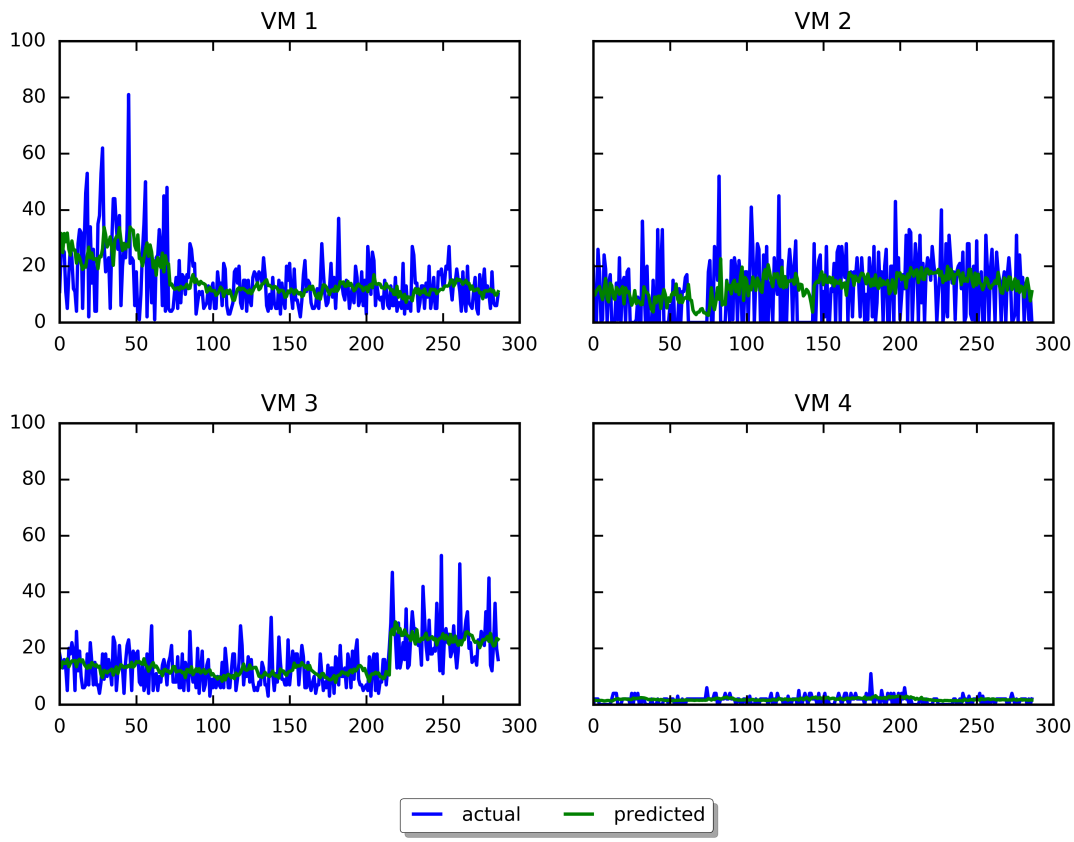



Figure 6.4: RF: Predicted versus actual CPU utilization of four random VMs

### 6.5.2 Long Short-Term Memory

An artificial neural network (ANN) is a computing system inspired by (but not identical to) the biological neural networks (NN) of human brains [Kri07]. An ANN tries to mimic human brains by considering examples to learn how to perform tasks without being explicitly programmed to do so. An ANN consists of three components: (1) set of neurons, (2) directed connections among the neurons, and (3) weights of the connections. A neuron (also known as a *unit* or a *node*) is the central unit of an ANN. Each neuron in the ANN represents a simple processing unit and loosely models the neurons in a biological brain. The connections among neurons resemble the synapses in a biological brain by transferring signals among neurons. The weight of a connection represents the strength of connections between neurons and decides how much influence the input will have on the output. Each neuron receives a number of *inputs*, the *weights* of the connections and a *bias* value. During the model training phase, the weights are updated in a way that decreases the prediction error (loss). The bias adds an extra input to make sure that the neuron can be activated if all inputs are zeros. Besides the inputs, weights, and bias, the *activation* function defines the output of a neuron by squashing the output values in a smaller range. Common examples of activation functions include linear, sigmoid, hyperbolic tangent (tanh), softmax, exponential linear unit (elu) [CUH15], scaled exponential linear units (SELUs) [KUMH17], rectified linear unit(ReLU), and parametric rectified linear unit(PReLU). Figure 6.5 shows the operations at a single neuron of a single-layered ANN and explains how the output of a neuron is calculated based on the inputs, weights, bias and the application of the sigmoid activation function (g).

An ANN consists of an *input layer*, *output layer* and one or more *hidden layers*. If an ANN has two or more hidden layers, then it is called a deep neural network. Figure 6.6 shows the architecture of an ANN. The input layer accepts the input values and passes them to the next layer. The hidden layers consist of one or more hidden layers, and each hidden layer may have a different number of neurons. The connection between each two connected neurons should have a weight representing the strength of that connection. The output layer receives input from the last hidden layer; the number of outputs depends on the number of classes in case of classification problems or a single value in case of regression problems.

The traditional feedforward neural networks, also known as multilayer perceptrons

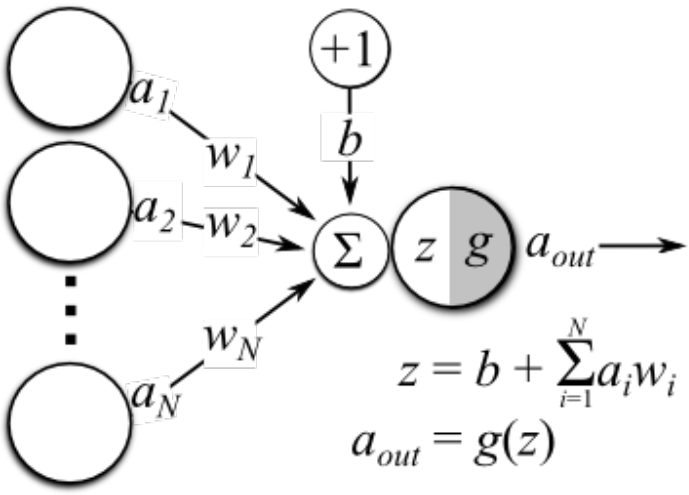



Figure 6.5: Operations at a single neuron of a single-layered ANN, from [Sta14]

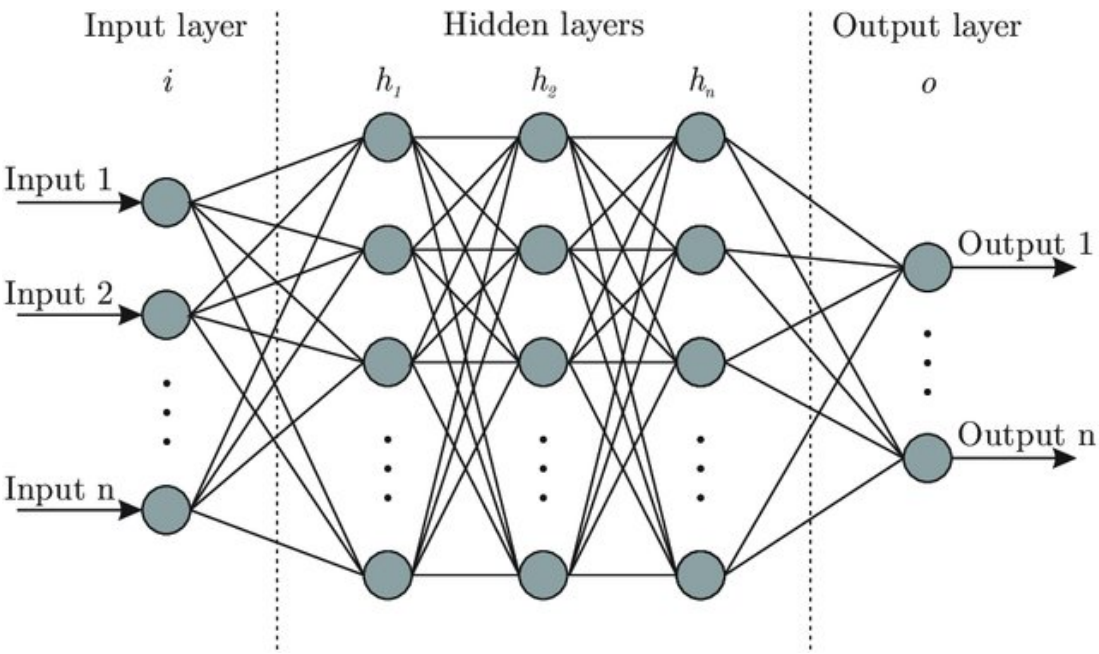



Figure 6.6: Architecture of an ANN, from [BGF18]

(MLPs), have some limitations, including being stateless, and having no explicit handling or understanding of the temporal structure between observations [Bro17]. Therefore, Recurrent Neural Networks (RNNs) has been introduced to allow information persistence. RNNs are a special type of neural network with *loops* that enable it to retain information, by allowing information to be passed from one step of the network to the next. Figure 6.7, from [Bro17], shows an example of an unrolled RNN that reveals how the RNN can take input from previous time steps. In Figure 6.7

$(X(t), X(t+1), \dots)$ represent multiple time steps of the input, $(y(t), y(t+1), \dots)$ represent multiple time steps of outputs, and $(u(t), u(t+1), \dots)$ represent multiple time steps of internal state. Figure 6.7 demonstrates that the inputs at the current time step $X(t+1)$ takes the output $y(t)$ and the state $u(t)$ of the previous time step (t). RNNs suits *sequence* problems as they can pass information from previous time steps to following time steps. A Sequence imposes an order on the data sample, and that order must be maintained when training models and making predictions [Bro17]. As an illustration, the CPU prediction problem is an example of sequence problems where the learning model should preserve the order of the CPU utilization across the time during model training and prediction phases.

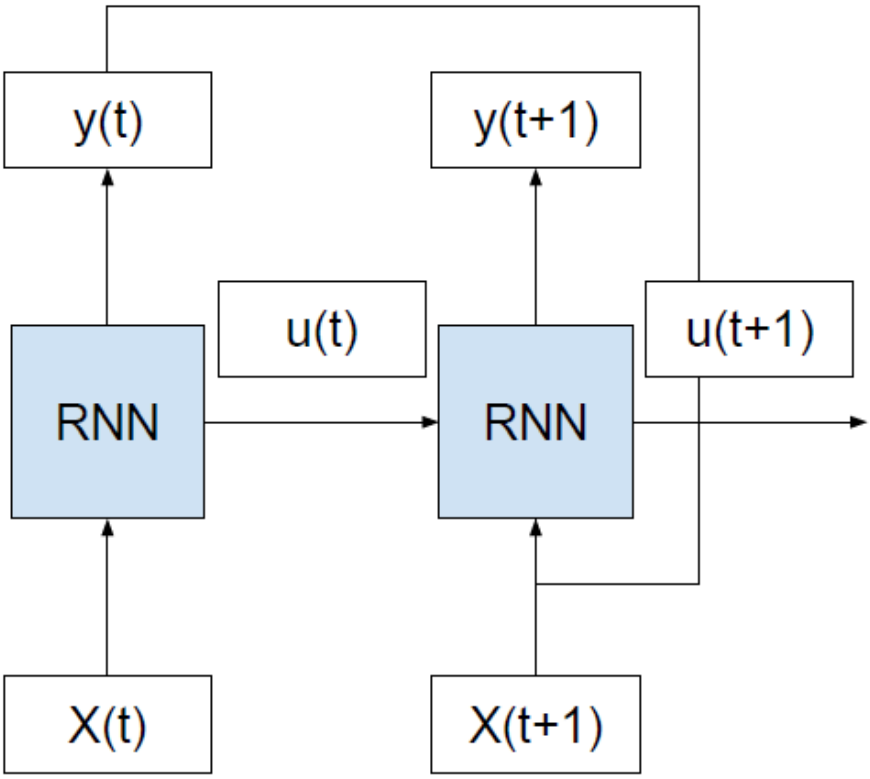



Figure 6.7: Example of an unfolded recurrent neural network, from [Bro17]

Theoretically, RNNs are expected to handle *long-term dependencies* where the gap between the current and previous time steps becomes very large. However, in practice, it is found that RNNs are not capable of handling long-term dependencies [BSF94]. Therefore, Long Short-Term Memory (LSTM) has been introduced to solve the long-term dependencies of the standard RNNs. LSTM is a learning technique based on RNNs [GES02] that is widely used for sequence problems such as speech recognition,

sentiment analysis, and text prediction. What is special about LSTM network is that it has internal contextual state cells that act as long-term or short-term memory cells, and therefore, it is capable of learning long-term dependencies [HS97]. LSTM networks have memory blocks that are connected into layers, and each block contains gates that manage the state of the block and the output. There are three types of gates within a memory block namely, *input*, *output* and *forget*. The forget gate decides what information to discard or throw away from the previous cell state.The input gate decides which values from the input to update the memory state (updating old cell state into the new cell state). Finally, the output gate decides what to output based on the input and memory. It is found that LSTM outperforms traditional RNNs on temporal processing tasks [GES02]. There are many variants of LSTM that introduce minor or major changes; for example, combining the input and forget gates into a single new gate called *update gate* [CVMG+14]. So far, the performance of various LSTM variants does not show significant differences [GSK+16].

### LSTM: Model Training and Making Predictions

Keras[3], a Python-based open-source neural network library, is used for the implementation and evaluation of the LSTM model. The following are the set of hyperparameters and their values that have been used during the training of the LSTM model.

- *Batch size*: The batch size is an integer that defines the number of training samples to be processed before updating the internal parameters of the LSTM model. The entire training dataset can be divided into many batches. A *mini-batch gradient descent* have been adopted where the batch size is greater than one and less than the size of the training dataset. There is an inverse relationship between the batch size and the required memory space. Popular batch sizes for the mini-batch gradient descent include 32, 64, and 128 samples. Therefore, we have used a batch size consisting of 64 samples for LSTM training.

- *Number of epochs*: The number of epochs is an integer that defines the number of times that the learning algorithm will work through the entire training dataset. The number of epochs should be sufficient so that the learning algorithm can run until the prediction error is adequately minimized. The learning curve can be plotted using the number of epochs against the error to see the fit (*e.g.* overfitting, underfitting, or good fitting) of the model. The model training has been

[3]https://keras.io/

conducted using 10 epochs, which were reasonable to improve LSTM learning by minimizing the prediction error.

- *Number of units in the LSTM layer*: Number of units in an LSTM layer defines the dimension of LSTM's inner memory cells (the number of neurons per LSTM layer). An LSTM layer consisting of 50 units has been used during LSTM model training.

- *Activation function*: The activation function determines the output of a neural in each layer of the neural network. Common activation functions have been mentioned in Section 6.5.2. The tanh activation function has been used for the LSTM layer, while a linear activation layer has been used for the output layer as it fits well for regression tasks.

- *Optimizer*: To compile a Keras model, we need to specify the optimizer, which is a search technique that iteratively updates the model's weights. Common optimizers include stochastic gradient descent (SGD) [Bot10], Adagrad [DHS11], Adadelta [Zei12] and Adam [Ker18b]. Adam optimizer [KB14] has been adopted as it is computationally efficient (does not require too much memory). Additionally, Adam optimizer suits prediction problems that have significant data very well [KB14].

- *Loss function*: The loss (*a.k.a* error) function estimates the error for current the state of the LSTM model so that the weights can be updated to decrease the prediction error. There are various loss functions for regression and classification problems. Examples of loss functions for regression include mean squared error (MSE), mean absolute error (MAE), and mean squared logarithmic error (MSLE). On the other hand, examples of loss functions for classification tasks include binary cross-entropy, sparse multiclass cross-entropy, and squared hinge [Ker18a]. MSE has been used as the loss function for the LSTM model.

- *Dropout ratio*: The dropout layer helps prevent overfitting and hence reduces the sensitivity to the weights. The dropout ratio defines the fraction of the units to drop for the linear transformation of the inputs; the dropout ratio is always between 0 and 1. As a best practice, it is recommended to use a dropout rate between 0.2 and 0.5 as too-small ratio will have minimal effect, and a too-high ratio results in under-learning by the neural network. The adopted LSTM model uses a value of 0.2 for the dropout ratio.

Once the LSTM model has been constructed based on the training dataset of PlanetLab CPU traces, the resulting model has been used to predict the last day of PlanetLab traces (which represent the test dataset). Figure 6.8 shows the predicted versus actual CPU utilization from four randomly selected VMs using the resulting LSTM model. The visual analysis of LSTM predictions in Figure 6.8 are comparable to the ones from random forest that has been previously shown in Figure 6.4.

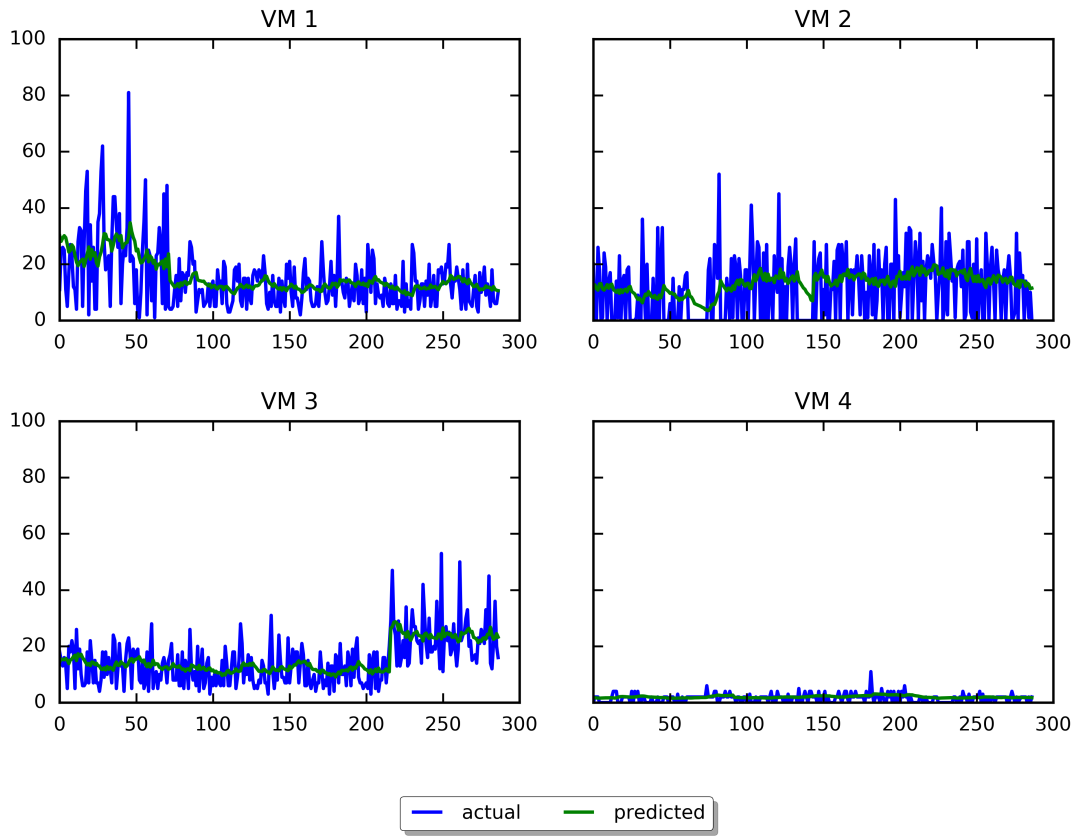



Figure 6.8: LSTM: Predicted versus actual CPU utilization of four random VMs

### 6.5.3 Model Evaluation

This section aims to evaluate the performance of both RF and LSTM for the prediction of CPU utilization. Scikit-learn offers several methods to evaluate the quality of the model's predictions. To measure the regression performance, there are various regression metrics such as *mean absolute error* (MAE), *mean squared error* (MSE), *Root mean square error* (RMSE) and *R-squared* ($R^2$). Table 6.1 summarises the results of the performance metrics for both RF and LSTM using both training and test datasets.

Table 6.1: Performance evaluation of RF and LSTM

| Algorithm | MAE | MSE | RMSE | $R^2$ |
|---|---|---|---|---|
| RF Training set | 3.63586 | 33.60172 | 5.7967 | 0.85828 |
| LSTM Training set | 4.57961 | 68.24607 | 8.26112 | 0.71102 |
| RF Test set | 5.56713 | 102.18039 | 10.10843 | 0.56002 |
| LSTM Test set | 5.43764 | 103.02204 | 10.14998 | 0.5565 |

In Table 6.1, the first three metrics (MAE, MSE, and RMSE) are different measures of prediction errors. On the other hand, the $R^2$ is the regression score function, which measures the goodness of the model fit. Considering the results of the training dataset in Table 6.1, it can be seen that RF outperforms LSTM. However, looking at the results from the test dataset, there is no significant difference between the two models ( they are comparable).

As the evaluation of the prediction phase has been completed, the following section describes the experimental evaluation of the proposed hybrid approach for parameter estimation per VM ($\alpha_{vm}$) using RF.

## 6.6 Experimental evaluation

### 6.6.1 Experiments Setup

The details of the experimental setup are as follows:
The initial and dynamic VM placement algorithm, the overload and the underload detection algorithms, the SLAVs thresholds and the choice of the initial values of the parameter $\alpha_{cdc}$ are exactly the same as those in Section 5.6. The main difference is that the following experiment only uses the normal PlanetLab traces. Finally, the details of the hyperparameters used for both RF and LSTM have been reported in Section 6.5.1 and Section 6.5.2 respectively.

### 6.6.2 Reactive versus Hybrid Parameter Estimation

This experiment aims to evaluate the reactive and hybrid (using RF) approaches for the parameter estimation. On the one hand, the window-based approach (*c.f.* Algorithm3) represents the reactive estimation of the CDC's parameter $\alpha_{cdc}$. On the other hand, Algorithm 4 represents the hybrid approach for the VM-level parameter $\alpha_{vm}$. The experiment uses RF to predict the future CPU usage of the VMs in the hybrid approach

(Algorithm 4). The reason for adopting RF is that it is computationally efficient and shows almost the same performance (or even slightly better) compared to LSTM as previously demonstrated in Section 6.5.3.

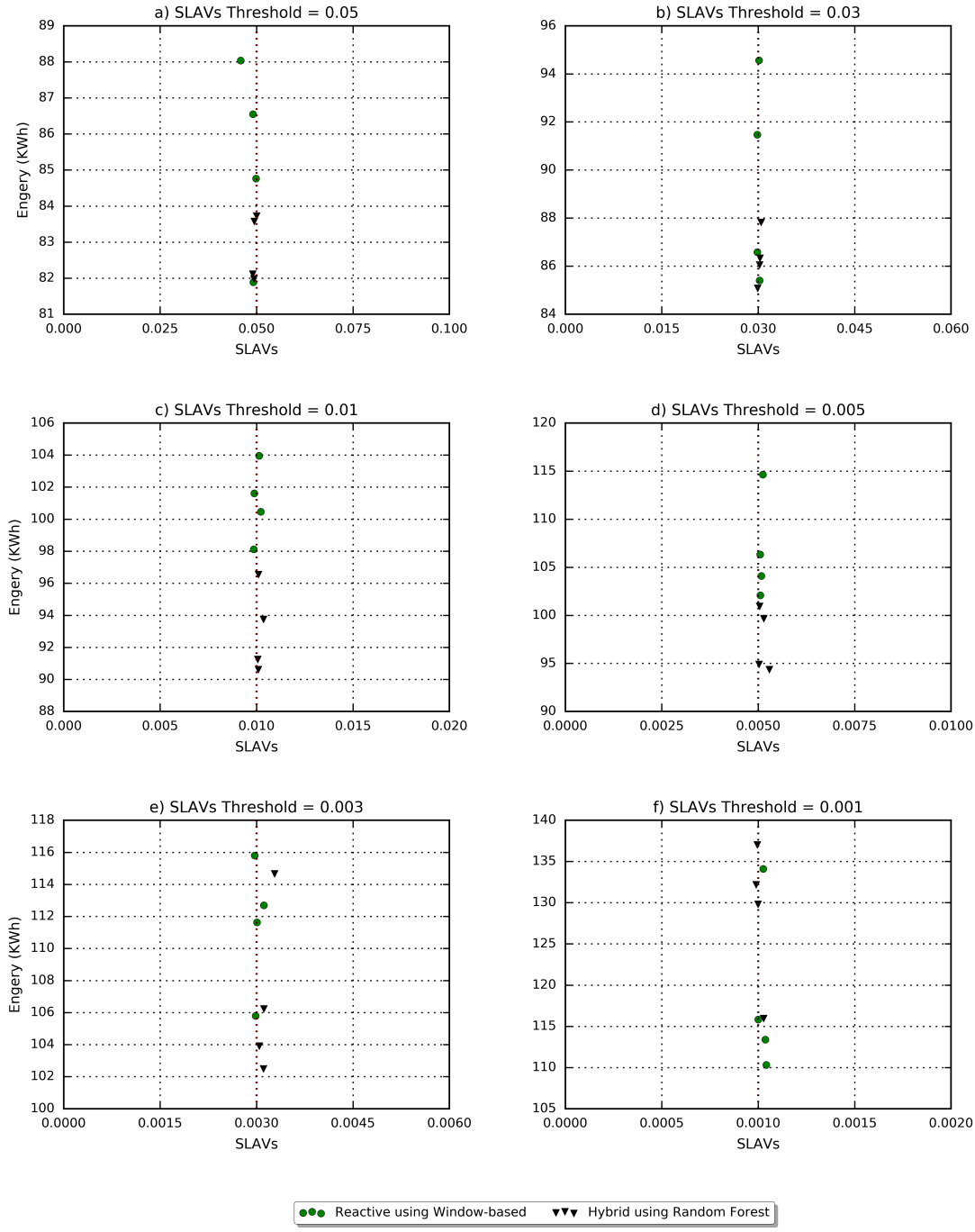



Figure 6.9: Hybrid versus reactive approaches for parameter estimation

Figure 6.9 shows the results from the reactive window-based approach (*c.f.* Algorithm 3) as opposed to the ones from the hybrid approach that predicts CPU utilization using random forest based on Algorithm 4. Each one of the six sub-plots measures the energy consumed and the resulting SLAVs for the specified SLAV threshold. On each sub-plots, there are eight points; four for each approach (reactive and hybrid). Each point represents a different value for the initial parameter (initial values are 0.2, 0.4, 0.6, and 0.8 as previously described in Chapter 5). By and large, Figure 6.9 shows that predicting the CPU utilization results in more efficient utilization of the PMs (shown in the lower values of energy consumption while meeting the SLA requirements).

Moreover, Figures 6.9(a), (b), (c) and (d) show that the hybrid approach is slightly less sensitive to the initial value of the parameter $\alpha_{cdc}$ which can be seen in the lower dispersion of the plotted points of the hybrid versus the reactive approach. Additionally, they confirm the energy savings, which resulted from predicting the utilization in the hybrid approach as opposed to the reactive window-based approach. When comparing the best values of the reactive (windows-based) and the hybrid (using RF), we can see that the hybrid approach can save an additional 7% and 8% of energy consumption. As an illustration, in Figure 6.9(c), the lowest energy consumption using the hybrid approach is 90 KWh as opposed to 98 KWh in case of the reactive approach. The only case when the hybrid approach did not manage to show significant improvement above the reactive approach is when the SLAVs threshold is very low. With better prediction accuracy, which can be obtained by training the models using more data that may even have more feature, the hybrid approach can show more significant improvements in energy savings.

Figure 6.10 summarizes the *average* energy consumption for the hybrid and reactive approaches using different values of the SLAVs thresholds. Figure 6.10 has ignored the resulting SLAVs, as the average SLAVs from both approaches were almost the same and managed to comply with the specified SLAVs threshold. It can be seen from Figure 6.10 that the hybrid approach can improve energy-savings for all the specified SLAVs thresholds except when the SLAVs threshold is minimal (*e.g.*0.001). What is striking about Figure 6.10 is that the reactive approach is found to be slightly better than the hybrid approach for the smallest value of SLAVs threshold. Looking at Figure 6.9(f), it can bee seen that one of the points from the hybrid approach is located away from others, which justifies the increase in the average energy consumption of the hybrid approach in Figure 6.10.

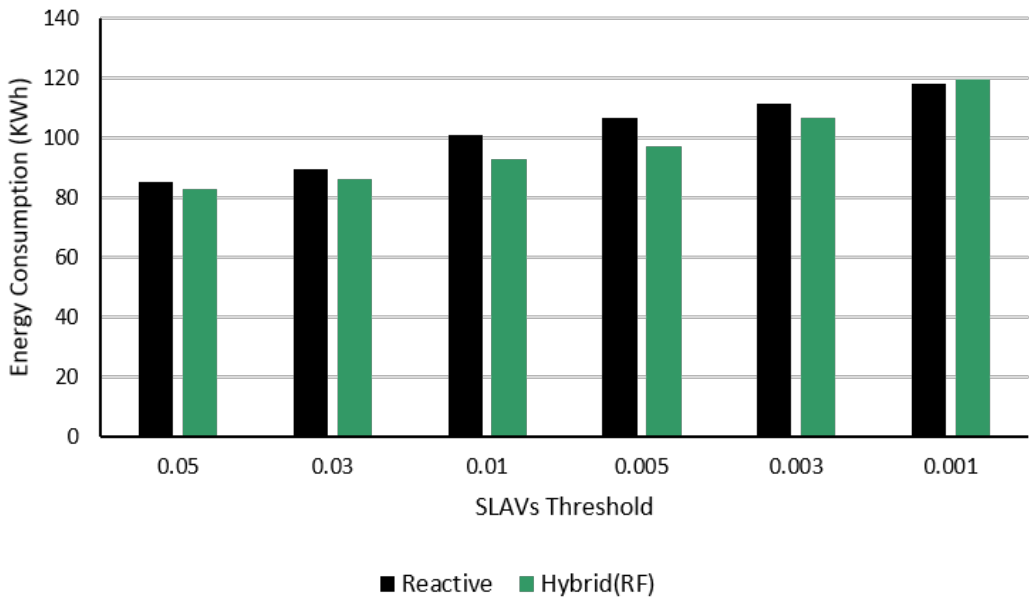



Figure 6.10: Average energy consumption using hybrid and reactive approaches for the specified SLAVs' thresholds.

## 6.7 Discussion

On the one hand, the evaluation of RF and LSTM for predicting VMs' CPU utilization using the test dataset indicate that there is no significant difference between the two supervised learning approaches. In general, additional CPU traces (*e.g.* traces for longer periods such as months) besides considering other features such as the name of each server, the day of the week, and server location may result in more efficient predictions for the LSTM model. On the other hand, the proposed hybrid approach for parameter estimation per VM shows its effectiveness even with the current limited amount of the PlanetLab traces. However, a more efficient prediction could result in further energy savings while adhering to the SLA requirements.

Some machine learning models can support offline learning, while others support online learning. In offline learning, the machine learning models are typically trained only once and then used for prediction whenever the model is satisfactory. The performance of the offline model might be degraded over time, especially when the environment frequently changes with new observations. On the other hand, in online learning, the model is trained and updated continuously using the new observations and adapts accordingly. Offline learning can be considered as a good option if there is enough training data set, and the data are not continuous streams. Over time, the model can be

updated with new observations to avoid performance degradation, and that new model can replace the old one for making predictions. However, the CPU utilization traces are continuous streams of data, and hence online learning algorithm may be better than the offline ones. LSTM originally supports batch learning; however, it can support online learning by setting the batch size to one (while in production). Setting the batch size to one will make LSTM adapt quickly, but it can affect the prediction accuracy. Therefore, experimenting with other models that support online learning (*e.g.* hierarchical temporal memory (HTM) [CSAH16]) of streaming data worth further investigation.

Even though the hybrid approach using prediction models outperforms the reactive one for parameter estimation, the computational complexity of these predictive models should be put into consideration. The computational cost of predictive models, especially the neural network-based ones (such as LSTM), is high during the training phase. However, the advances in hardware and the cloud-based computational services make the training of these model more feasible. Additionally, neural network-based models are generally slow to train, but they are usually fast to evaluate [Bro17]. Therefore, with powerful hardware, the cost of prediction models (especially during training) may become less significant. This makes the adoption of these models practical even if the current added value is not of great significance in the conducted experiment. However, with more data, features, and further hyperparameter optimization, such models can improve the performance of the hybrid approach.

## 6.8 Summary

This chapter sets out to investigate the impact of adopting a hybrid (reactive and proactive) approach on improving the PMs' utilization while satisfying SLA requirements. The hybrid approach both reacts to the current SLAVs and forecasts the CPU utilization of the VMs. Predicting the CPU utilization of the individual VMs enables the parameter-based VM placement solution to update the value of the parameter accordingly. Therefore, there was a need to have two parameters: (1) one parameter for the CDC level $\alpha_{cdc}$, and (2) another parameter for the VM level $\alpha_{vm}$. The current value of $\alpha_{cdc}$ bounds the maximum value of each $\alpha_{vm}$. Then, the chapter demonstrated the algorithm for the hybrid approach to adjust $\alpha_{vm}$ taking into consideration the predicted value for the CPU utilization.

Before creating a prediction model for the CPU utilization, the PlanetLab traces has been transformed into an inputs and outputs structure that suits the supervised

learning algorithms. Then, we have conducted an exploratory data analysis of the PlanetLab traces with the help of ACF and PACF function. The graphs of ACF and PACF showed that there is a correlation between the current and previous CPU utilization which means they can be predicted. Next, the chapter described the details of creating the two regression methods that we have adopted for prediction, namely, RF and LSTM. The results from RF and LSTM are comparable, and there is no significant difference between the results from the two approaches. Additional datasets can reveal the difference between the two approaches as the neural network-based learning models improve with extra data. There is also a need to investigate online learning models such as hierarchical temporal memory (HTM) as CPU traces are examples of online streaming data. Eventually, the chapter evaluated the new hybrid approach, which predicts CPU Utilization using RF, against the reactive window-based approach. The empirical evaluation confirms that predicting VMs' utilization results in more efficient utilization ratios of the hosting PMs, which improves energy savings and reduces other related costs.

# Chapter 7

# Conclusion and Future Directions

## 7.1 Conclusion

In modern cloud data centres, resource overprovisioning can lead to inefficient utilization (underutilization) of the CDC's PMs, which results in more energy consumption and operational costs. On the other hand, underprovisioning may result in overutilization situations, which may lead to more SLAVs. The SLAVs incurs additional costs related to either the penalties that must be paid to the cloud customers or losing unsatisfied customers. Therefore, there is a cloudless trade-off between energy consumption and SLAVs in existing reservation-based and demand-based VM placement strategies. All current literature on VM placement pays the entire attention to the two extremes of the VM placement strategies and ignore the middle way in-between these two strategies. As a result, this thesis aimed to find a more desirable and flexible strategy (representing the middle way) that might help to overcome the problems associated with the two extremes. This strategy exploits the area between demand-based and reservation-based placement, striving for a workable balance between resource utilization and SLAVs.

Chapter 2 conducted a review and a critical analysis of existing literature on VM placement, which resulted in a novel classification of the existing VM placement research. The proposed classification groups current VM placement research in four areas, namely VM placement problems, strategies, policies, and objectives. Through our novel classification and analysis, we aim to direct ourselves and other researchers to concentrate on areas of higher impact. Additionally, our novel classification offers a roadmap that can be followed when trying to propose a new VM placement solution. This roadmap is summarized in the following four steps:

1. Specifying the type of the problem being solved (initial vs. dynamic placement) and the architecture of the CDC.
2. Choosing the strategy to allocate resources to VMs (reservation-based vs. demand-based strategies).
3. Deciding the autonomic policy (*e.g.* action or utility functions) that should be applied to develop these self-managing VM placement systems.
4. Specifying the placement objectives and constraints.

Moreover, the observations concluded from the critical review of the literature were the key to define three possible areas that require further investigations. These research areas are: (1) exploring new VM placement strategies, (2) real-world performance evaluation of existing solutions, and (3) developing an evaluation and verification framework. Using these observations and the roadmap, we managed to delineate our pathway to introduce novel VM placement solution by investigating the area between demand- and reservation-based VM placement strategies. Furthermore, during the analytical review of the literature, an empirical evaluation of a genetic algorithm and simulated annealing for the dynamic reallocation of VMs been conducted (in Appendix A). This evaluation concludes that a genetic algorithm outperforms simulated annealing for the dynamic reallocation of VMs in CDCs. Additionally, another empirical evaluation of a VM placement solution that considers CPU and memory resources has been conducted (in Appendix B), which points out the importance of considering multiple types of resources. Eventually, it is found that the prohibitive SLAVs and underutilization costs are roadblocks to consolidation ratios from demand-based and reservation-based VM placement strategies, respectively.

Chapter 3 demonstrated the main components of the proposed parameter-based VM placement solution, which includes the VM placement requests, the initial VM placement controller, monitoring, dynamic VM placement controller, and the parameter estimation components. The parameter can be either statically set or dynamically estimated on the fly. Then, the chapter justified the reasons for adopting the CloudSim simulator. These reasons include assuring reproducibility at a low cost and the extensive use of CloudSim in the area of VM placement and resource management in clouds. After that, the chapter described the details of the simulated cloud environment and the characteristics of workload traces, besides demonstrating the need for skewing the PlanetLab traces. Finally, the chapter proposed various performance metrics that can measure the performance of the proposed VM placement solution.

Chapter 4 demonstrated the proposed parameter-based VM placement solution that makes use of a single static parameter to explore the middle way between the two extremes (reservation-based and demand-based) of VM placement strategies. Chapter 4 started by demonstrating the mathematical formulation of the VM placement problem. Then, it illustrated the details of the initial and dynamic components of the proposed parameter-based VM placement solution. The conducted experiments on Chapter 4 concludes the inherent benefits of investigating the area between the actual demand and the full reservation using the proposed parameter-based VM placement strategy. As an illustration, with a slight decrease in the allocated resources (compared to the reservation-based VM placement), the parameter-based VM placement can save a considerable amount of the SLAVs with an insignificant increase in the amount of energy consumed. As opposed to the existing demand-based and reservation-based VM placement strategies, the proposed parameter-based VM placement strategy proved to offer wide flexibility and viable options for CDC operators to strike an appropriate balance between energy consumption and SLAVs. The static nature of the parameter introduced in Chapter 4 is a key limitation of the proposed parameter-based VM placement strategy. Employing such a static parameter is not enough to enable the cloud provider to set specific business-related thresholds that should not be surpassed.

Chapter 5 demonstrated the need for the dynamic estimation of the parameter $\alpha$ of the proposed parameter-based VM placement solution. Because a static parameter $\alpha$ does not enable the cloud provider to set SLAVs thresholds that should not be exceeded, Chapter 5 proposed different algorithms for the dynamic estimation of the parameter $\alpha$ on the fly. The dynamic estimation of the parameter $\alpha$ has the objective of not exceeding a certain SLAVs threshold while achieving an efficient resource utilization. Then, the chapter presented two algorithms for the online estimation of the parameter (namely, instantaneous and window-based). The instantaneous approach adjusts the value of the parameter $\alpha$ based on the distance between the current number of SLAVs and the predefined SLAVs threshold. The window-based approach makes use of the recent history of the number of SLAVs. This online estimation of the parameter $\alpha$ offers more flexibility for CDC management in achieving business-related objectives, such as setting an SLAVs threshold that should not be exceeded. Experimental evaluation concludes that the window-based approach is more efficient in estimating a dynamic parameter that adheres to the SLAVs threshold while efficiently utilizes the PMs, thus saving energy and costs. However, the principal limitation of the dynamic approaches introduced in Chapter 5 is that they are coarse-grained as they estimate the

same value of parameter $\alpha$ for all VMs. Therefore, more-fine grained approaches are required to estimate different values for the parameter for each single VM whenever needed.

Finally, Chapter 6 introduced a hybrid approach that can estimate a fine-grained parameter at the VM level. The proposed hybrid approach estimates the parameter at the VM level by considering the current SLAVs threshold (reactive) and the predicted CPU utilization of the VMs (proactive). Chapter 6 investigated two supervised learning models (namely, RF and LSTM) to predict the CPU utilization of the individual VMs. The model evaluation concludes that there is no significant difference in the performance of the two techniques for CPU prediction. The hybrid approach confirms that predicting the utilization of the individual VMs can result in better utilization ratios of the hosting PMs and hence saves energy and other related costs. The computational complexity of the predictive models might be a limiting factor, especially when the model accuracy is not quite high. Notwithstanding the computational complexity of these predictive models, the advances in the hardware with more historical data and features can result in more efficient prediction, which aids in building robust VM placement solutions.

## 7.2 Future directions

The remaining of this chapter introduces some potential research directions that can extend the work conducted in this thesis.

### 7.2.1 Online Prediction of Resource utilization using Real Traces

This direction of research involves two factors, namely *using online prediction algorithms* that support data streams and *exploring additional real traces* of resource utilization. Concerning the online prediction of data streams, monitoring tools in real cloud data centres generate continuous streams of various metrics, including resource utilization. Therefore, online prediction algorithms are required to learn and adapt to changes in the data continuously. One suggestion is to investigate the effectiveness of the hierarchical temporal memory (HTM) as opposed to the neural-network-based model. The reasons for choosing HTM include: (1) its continuous learning nature using data streams, (2) proven robustness against noise in data, (3) requires less hyperparameter optimization. Concerning the exploration of additional real traces, it is

crucial to test the online prediction algorithms on various real workload traces which exhibit distinct characteristics. With more additional datasets, we can measure the effectiveness of the adopted learning algorithms; additionally, more data for neural-network-based prediction techniques (such as LSTM) can improve the learning process and lead to more accurate predictions. Therefore, conducting comparative studies of learning algorithms that support online learning using several real data sources can add real value to the current literature on predicting resource utilization in cloud and distributed environments.

### 7.2.2 Various SLA Requirements per Instance Type

The work on this thesis sought to consolidate VMs while satisfying the cloud provider's objectives by not exceeding a predefined SLAVs threshold for the entire CDC. However, we propose an alternative wherein the cloud provider can offer a number of VM instances with different SLAVs thresholds for each type of VM instance. In this scenario, the SLAVs thresholds are set for the individual types pf VM instances and are not for the whole cloud data centre. For example, if a cloud user requires the full reservation regardless of the cots, then we can set $\alpha = 1$. On the other hand, new parameter estimation approaches can be proposed to estimate the value of the parameter for each type of the running VM instances based on the SLA requirements.

### 7.2.3 Multiple Resource Types Parameter-based VM Placement

In this thesis, we have introduced the parameter-based VM placement strategy based on a single resource type (CPU). Despite the importance of CPU resources, it is essential to consider other resource types such as memory and network requirements. As an illustration, if the cloud provider is offering VM instances that are either memory-intensive or network-intensive, then considering memory and network requirements is going to be of great importance. Furthermore, an additional research direction might be considering the topology type of the underlying network architecture of the CDC and estimating its effectiveness in large-scale cloud infrastructures.

### 7.2.4 Parameter-based VM Placement using Utility Functions

In our proposed parameter-based VM placement solution, we have adopted an action policy to create an autonomic system that adapts to changes in the cloud data centre. A potential research direction might be employing a utility-based policy for the parameter-based VM placement solution. The reason for this suggestion is that the utility-based approaches are compelling for creating such self-managing systems, and therefore may introduce some valuable insights. This utility-based approach involves utility function definition followed by the adoption of meta-heuristics such as genetic algorithms, ant colony system, particle swarm optimization, or others to search for the desired state of the VMs-to-PMs mapping.

### 7.2.5 Considering Multiple Cloud Data Centres

The majority of existing VM placement solutions, including the work conducted in this thesis, only considers a single cloud data centre. However, reallocating VMs among various CDCs that spans multiple geographical locations might be useful for several use-cases. The consideration of multiple CDCs entails paying attention to the constraints of VMs' migrations across various geographical locations of the data centres. There is also a need to conduct a cost-benefit analysis of VMs' migrations among CDCs belonging to the same cloud provider.

### 7.2.6 Real (in-vivo) Evaluation of Existing Solutions

With accessibility to even a small-scale cloud data centre, conducting real-world experiments can add tangible value and can result in useful insights. Even though there many limitations of running real experiments as previously explained in Chapter 3, there are some possible ways to bypass such limitations. For example, using small-scale private cloud data centre that hosts dozens of PMs might be of great value. Another option is the collaboration with institutions that have their own cloud data centres, or even creating community clouds for research institutions in any region. Conducting real experiments would enable us to measure the effectiveness of many of the current solutions in practical scenarios. Furthermore, conducting real experiments can tell us why many of the industrial solutions adopt action policies even though the research shows that utility-based policies result in better solutions.

### 7.2.7 An Evaluation and Verification Framework

A final potential research direction might be the development of a component-based evaluation and verification framework that can systematically verify existing solutions using real workload traces. Currently, researchers develop and validate their solutions in various simulations environments that are usually incomparable. This framework can be offered through a web-based system linked to a simulated or a real CDC so that researchers only upload their algorithms and run them. The proposal of such an evaluation framework can enable researchers to easily plug their new VM placement solutions, which are then evaluated and tested against realistic use-cases.

# Bibliography


[AB12] Salman Abdul Baset. Cloud slas: Present and future. *SIGOPS Oper. Syst. Rev.*, 46(2):57–66, July 2012.

[AE15] Anders SG Andrae and Tomas Edler. On global electricity usage of communication technology: trends to 2030. *Challenges*, 6(1):117–157, 2015.

[Ale06] Bryzan Alexander. Web 2.0. *A New Wave of Innovation for Teachning and learning*, pages 32–44, 2006.

[Ama] Amazon EC2 Instances.

[ARMMA18] Patricia Arroba, José L Risco-Martín, José M Moya, and José L Ayala. Heuristics and metaheuristics for dynamic management of computing and cooling energy in cloud data centers. *Software: Practice and Experience*, 48(10):1775–1804, 2018.

[BAB12] A Beloglazov, J Abawajy, and R Buyya. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future Generation Computer Systems*, 2012.

[BB10] A. Beloglazov and R. Buyya. Energy efficient allocation of virtual machines in cloud data centers. In *2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, pages 577–578, May 2010.

[BB12] Anton Beloglazov and Rajkumar Buyya. Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. *Concurrency and Computation: Practice and Experience*, 24(13):1397–1420, 2012.

[BBLZ11] Anton Beloglazov, Rajkumar Buyya, Young Choon Lee, and Albert Zomaya. A Taxonomy and Survey of Energy- Efficient Data Centers and Cloud Computing Systems. *Engineering*, (December 2010):1–24, 2011.

[BCF+12] Ofer Biran, Antonio Corradi, Mario Fanelli, Luca Foschini, Alexander Nus, Danny Raz, and Ezra Silvera. A stable network-aware VM placement for cloud systems. *Proceedings - 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid 2012*, pages 498–506, 2012.

[BDF+03] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *ACM SIGOPS operating systems review*, volume 37, pages 164–177. ACM, 2003.

[Bel13] Anton Beloglazov. *Energy-efficient management of virtual machines in data centers for cloud computing*. PhD thesis, 2013.

[BGF18] Facundo Bre, Juan M Gimenez, and Víctor D Fachinotti. Prediction of wind pressure coefficients on building surfaces using artificial neural networks. *Energy and Buildings*, 158:1429–1441, 2018.

[BH07] Luiz André Barroso and Urs Hölzle. The case for energy-proportional computing. 2007.

[BKB07] Norman Bobroff, Andrzej Kochut, and Kirk Beaty. Dynamic placement of virtual machines for managing sla violations. In *Integrated Network Management, 2007. IM'07. 10th IFIP/IEEE International Symposium on*, pages 119–128. IEEE, 2007.

[BKNT11] Christian Baun, Marcel Kunze, Jens Nimis, and Stefan Tai. *Cloud computing: Web-based dynamic IT services*. Springer Science & Business Media, 2011.

[Bot10] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In Yves Lechevallier and Gilbert Saporta, editors, *Proceedings of COMPSTAT'2010*, pages 177–186, Heidelberg, 2010. Physica-Verlag HD.

[Bre01] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[Bro17] Jason Brownlee. *Long Short-term Memory Networks with Python: Develop Sequence Prediction Models with Deep Learning*. Machine Learning Mastery, 2017.

[BS14] D. Borgetto and P. Stolf. An energy efficient approach to virtual machines management in cloud computing. In *2014 IEEE 3rd International Conference on Cloud Networking (CloudNet)*, pages 229–235, Oct 2014.

[BSF94] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[BVWS14] Adam Barker, Blesson Varghese, Jonathan Stuart Ward, and Ian Sommerville. Academic cloud computing research: Five pitfalls and five opportunities. In *HotCloud*, 2014.

[BZP+10] C. Bezemer, A. Zaidman, B. Platzbeecker, T. Hurkmans, and A. '. Hart. Enabling multi-tenancy: An industrial experience report. In *2010 IEEE International Conference on Software Maintenance*, pages 1–8, Sep. 2010.

[CBHM12] Nicolò Maria Calcavecchia, Ofer Biran, Erez Hadad, and Yosef Moatti. VM placement strategies for cloud scenarios. *Proceedings - 2012 IEEE 5th International Conference on Cloud Computing, CLOUD 2012*, pages 852–859, 2012.

[CD13] Z. Cao and S. Dong. Energy-aware framework for virtual machine consolidation in cloud computing. In *2013 IEEE 10th International Conference on High Performance Computing and Communications and 2013 IEEE International Conference on Embedded and Ubiquitous Computing*, pages 1890–1895, Nov 2013.

[CFF14] Antonio Corradi, Mario Fanelli, and Luca Foschini. VM consolidation: A real case based on OpenStack Cloud. *Future Generation Computer Systems*, 32(1):118–127, 2014.

[CFH+05] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live migration of virtual machines. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*, pages 273–286. USENIX Association, 2005.

[CHT03] Kishore Channabasavaiah, Kerrie Holley, and Edward Tuggle. Migrating to a service-oriented architecture. *IBM DeveloperWorks*, 16:727–728, 2003.

[CMR15] Mohammed Rashid Chowdhury, Mohammad Raihan Mahmud, and Rashedur M Rahman. Implementation and performance analysis of various vm placement strategies in cloudsim. *Journal of Cloud Computing*, 4(1):1, 2015.

[CNDRB13] Rodrigo N Calheiros, Marco AS Netto, César AF De Rose, and Rajkumar Buyya. Emusim: an integrated emulation and simulation environment for modeling, evaluation, and validation of performance of cloud computing applications. *Software: Practice and Experience*, 43(5):595–612, 2013.

[CRB+11a] Rodrigo N Calheiros, Rajiv Ranjan, Anton Beloglazov, César AF De Rose, and Rajkumar Buyya. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and experience*, 41(1):23–50, 2011.

[CRB11b] Rodrigo N Calheiros, Rajiv Ranjan, and Rajkumar Buyya. Virtual machine provisioning based on analytical performance and QoS in cloud computing environments. In *Parallel Processing (ICPP), 2011 International Conference on*, pages 295–304. IEEE, 2011.

[CRDB09] Rodrigo N Calheiros, Rajiv Ranjan, César A F De Rose, and Rajkumar Buyya. Cloudsim: A novel framework for modeling and simulation of cloud computing infrastructures and services. *arXiv preprint arXiv:0903.2525*, 2009.

[CSAH16] Yuwei Cui, Chetan Surpur, Subutai Ahmad, and Jeff Hawkins. A comparative study of htm and other neural network models for online sequence learning with streaming data. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 1530–1538. IEEE, 2016.

[CSP11] Chetan Chudasama, S. M Shah, and Mahesh Panchal. Comparison of Parents Selection Methods of Genetic Algorithm for TSP. *International Conference on Computer Communication and Networks*, pages 85–87, 2011.

[CUH15] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

[CVMG+14] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[DAGLO12] Marcos Dias De Assuncao, Jean-Patrick Gelas, Laurent Lefevre, and Anne-Cécile Orgerie. The green grid5000: Instrumenting and using a grid with energy sensors. In *Remote Instrumentation for eScience and Related Aspects*, pages 25–42. Springer, 2012.

[Dan09] Jeff Daniels. Server virtualization architecture and implementation. *Crossroads*, 16(1):8–12, 2009.

[Dar14] W. Dargie. Estimation of the cost of vm migration. In *2014 23rd International Conference on Computer Communication and Networks (ICCCN)*, pages 1–8, Aug 2014.

[DHC16] D. Deng, K. He, and Y. Chen. Dynamic virtual machine consolidation for improving energy efficiency in cloud data centers. In *2016 4th International Conference on Cloud Computing and Intelligence Systems (CCIS)*, pages 366–370, Aug 2016.

[DHS11] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[DMKP16] Vincenzo De Maio, Gabor Kecskemeti, and Radu Prodan. An improved model for live migration in data centre simulators. In *Proceedings of the 9th International Conference on Utility and Cloud Computing*, UCC '16, pages 108–117, New York, NY, USA, 2016. ACM.

[FAP+15] Fahimeh Farahnakian, Adnan Ashraf, Tapio Pahikkala, Pasi Liljeberg, Juha Plosila, Ivan Porres, and Hannu Tenhunen. Using ant colony system to consolidate vms for green cloud computing. *IEEE Transactions on Services Computing*, 8(2):187–198, 2015.

[FFRR15] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio. An updated performance comparison of virtual machines and linux containers. In *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 171–172, March 2015.

[FHG+08] David Filani, Jackson He, Sam Gao, Murali Rajappa, Anil Kumar, Pinkesh Shah, and Ram Nagappan. Dynamic data center power management: Trends, issues, and solutions. *Intel Technology Journal*, 12(1), 2008.

[Fil] Filter scheduler. `https://docs.openstack.org/newton/config-reference/compute/schedulers.html`. Accessed: 2018-05-18.

[FMCB14] Md Hasanul Ferdaus, Manzur Murshed, Rodrigo N. Calheiros, and Rajkumar Buyya. Virtual machine consolidation in cloud data centers using aco metaheuristic. In Fernando Silva, Inês Dutra, and Vítor Santos Costa, editors, *Euro-Par 2014 Parallel Processing*, pages 306–317, Cham, 2014. Springer International Publishing.

[FPLP13] F. Farahnakian, T. Pahikkala, P. Liljeberg, and J. Plosila. Energy aware consolidation algorithm based on k-nearest neighbor regression for cloud data centers. In *2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*, pages 256–259, Dec 2013.

[FWB07] Xiaobo Fan, Wolf-Dietrich Weber, and Luiz Andre Barroso. Power provisioning for a warehouse-sized computer. In *Proceedings of the 34th Annual International Symposium on Computer Architecture*, ISCA '07, pages 13–23, New York, NY, USA, 2007. ACM.

[GCG+12] Anshul Gandhi, Yuan Chen, Daniel Gmach, Martin Arlitt, and Manish Marwah. Hybrid resource provisioning for minimizing data center SLA violations and power consumption. *Sustainable Computing: Informatics and Systems*, 2(2):91–104, 2012.

[GES02] Felix A Gers, Douglas Eck, and Jürgen Schmidhuber. Applying lstm to time series predictable through time-window approaches. In *Neural Nets WIRN Vietri-01*, pages 193–200. Springer, 2002.

[GGQ+13] Yongqiang Gao, Haibing Guan, Zhengwei Qi, Yang Hou, and Liang Liu. A multi-objective ant colony system algorithm for virtual machine placement in cloud computing. *Journal of Computer and System Sciences*, 79(8):1230–1242, 2013.

[GHBRK12] Anshul Gandhi, Mor Harchol-Balter, Ram Raghunathan, and Michael A Kozuch. Autoscale: Dynamic, robust capacity management for multi-tier data centers. *ACM Transactions on Computer Systems (TOCS)*, 30(4):14, 2012.

[Gol] Golden mean (philosophy) - wikipedia. `https://en.wikipedia.org/wiki/Golden_mean_(philosophy)`. Accessed: 2018-05-18.

[GSA12] Tarun Goyal, Ajit Singh, and Aakanksha Agrawal. Cloudsim: simulator for cloud computing infrastructure and modeling. *Procedia Engineering*, 38:3566–3572, 2012.

[GSK+16] Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 28(10):2222–2232, 2016.

[HKR+14] Abdul Hameed, Alireza Khoshkbarforoushha, Rajiv Ranjan, Prem Prakash Jayaraman, Joanna Kolodziej, Pavan Balaji, Sherali Zeadally, Qutaibah Marwan Malluhi, Nikos Tziritas, Abhinav Vishnu, Samee U. Khan, and Albert Zomaya. A survey and taxonomy on energy efficient resource allocation techniques for cloud computing systems. *Computing*, 2014.

[HMGW07] Chris Hyser, Bret McKee, Rob Gardner, and Brian J Watson. Autonomic virtual machine placement in the data center. *Hewlett Packard Laboratories, Tech. Rep. HPL-2007-189*, 189, 2007.

[Hoc97] Dorit S Hochba. Approximation algorithms for np-hard problems. *ACM Sigact News*, 28(2):40–52, 1997.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[HU13] A Hios and T Ulichnie. Top 10 Data Center Business Management Priorities for 2013 about the Uptime Institute Network. Technical report, Technical Report, Uptime Institute, 2013.

[IKU15] Amir Rahimzadeh Ilkhechi, Ibrahim Korpeoglu, and Özgür Ulusoy. Network-aware virtual machine placement in cloud data centers with multiple traffic-intensive components. *Computer Networks*, 91:508–527, 2015.

[JCC15] Christina Terese Joseph, K. Chandrasekaran, and Robin Cyriac. A novel family genetic approach for virtual machine allocation. *Procedia Computer Science*, 46:558 – 565, 2015. Proceedings of the International Conference on Information and Communication Technologies, ICICT 2014.

[JS14] Brendan Jennings and Rolf Stadler. Resource Management in Clouds: Survey and Research Challenges. *Journal of Network and Systems Management*, pages 1–53, 2014.

[JWHT13] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to statistical learning*, volume 112. Springer, 2013.

[KA19] Ashwin Kumar Kulkarni and B Annappa. Context aware vm placement optimization technique for heterogeneous iaas cloud. *IEEE Access*, 7:89702–89713, 2019.

[KB14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[KC03] Jeffrey O Kephart and David M Chess. The vision of autonomic computing. *Computer*, (1):41–50, 2003.

[KD07] Jeffrey O. Kephart and Rajarshi Das. Achieving self-management via utility functions. *IEEE Internet Computing*, 11(1):40–48, 2007.

[KDA+17] Mohammad Ali Khoshkholghi, Mohd Noor Derahman, Azizol Abdullah, Shamala Subramaniam, and Mohamed Othman. Energy-efficient algorithms for dynamic virtual machine consolidation in cloud data centers. *IEEE Access*, 5:10709–10722, 2017.

[Ker18a] Keras. Loss functions - keras. `https://keras.io/losses/`, 2018.

[Ker18b] Keras. Optimizers - keras. `https://keras.io/optimizers/`, 2018.

[KFK08] James M Kaplan, William Forrest, and Noah Kindler. Revolutionizing data center efficiency mckinsey and company, 2008.

[KGV83] Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

[Kir84] Scott Kirkpatrick. Optimization by simulated annealing: Quantitative studies. *Journal of statistical physics*, 34(5-6):975–986, 1984.

[KJ13] Max Kuhn and Kjell Johnson. *Applied predictive modeling*, volume 26. Springer, 2013.

[Kre00] Frank Kreith. *The CRC handbook of thermal engineering*. Springer Science & Business Media, 2000.

[Kri07] David Kriesel. *A Brief Introduction to Neural Networks*. 2007.

[KT15] Jonathan Koomey and Jon Taylor. New data supports finding that 30 percent of servers are comatose, indicating that nearly a third of capital in enterprise data centers is wasted. *June*, 3:2015, 2015.

[KUMH17] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. In *Advances in neural information processing systems*, pages 971–980, 2017.

[LB13] D. Lucanin and I. Brandic. Take a break: Cloud scheduling optimized for real-time electricity pricing. In *2013 International Conference on Cloud and Green Computing*, pages 113–120, Sept 2013.

[LKN+09] Alexander Lenk, Markus Klems, Jens Nimis, Stefan Tai, and Thomas Sandholm. What's inside the Cloud? An architectural map of the Cloud

landscape. In *Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*, pages 23–31. IEEE Computer Society, 2009.

[LL+12] Feng LIU, Zhen LIU, et al. Multi-objective optimization for initial virtual machine placement in cloud data center. *Journal of Information &Computational Science*, 9(16):5029–5038, 2012.

[LLW11] C. Lin, P. Liu, and J. Wu. Energy-efficient virtual machine provision algorithms for cloud systems. In *2011 Fourth IEEE International Conference on Utility and Cloud Computing*, pages 81–88, Dec 2011.

[LLYL15] Jiaxin Li, Dongsheng Li, Yuming Ye, and Xicheng Lu. Efficient multi-tenant virtual machine allocation in cloud data centers. *Tsinghua Science and Technology*, 20(1):81–89, 2015.

[LPB15] Fabio Lopez-Pires and Benjamin Baran. Virtual machine placement literature review. *arXiv preprint arXiv:1506.01509*, 2015.

[LS14] Craig A. Lee and Alan F. Sill. A design space for dynamic service level agreements in openstack. *Journal of Cloud Computing*, 3(1):17, 2014.

[LSN+09] S. Lim, B. Sharma, G. Nam, E. K. Kim, and C. R. Das. Mdcsim: A multi-tier data center simulation, platform. In *2009 IEEE International Conference on Cluster Computing and Workshops*, pages 1–9, Aug 2009.

[LSWJ18] Y. Liu, X. Sun, W. Wei, and W. Jing. Enhancing energy-efficient and qos dynamic virtual machine consolidation method in cloud environment. *IEEE Access*, 6:31224–31235, 2018.

[Luk15] Sean Luke. Essentials of metaheuristics. *Online Version*, 2015.

[LW+02] Andy Liaw, Matthew Wiener, et al. Classification and regression by randomforest. *R news*, 2(3):18–22, 2002.

[LXJ+11] Haikun Liu, Cheng-Zhong Xu, Hai Jin, Jiayu Gong, and Xiaofei Liao. Performance and energy modeling for live migration of virtual machines. In *Proceedings of the 20th International Symposium on High Performance Distributed Computing*, HPDC '11, pages 171–182, New York, NY, USA, 2011. ACM.

[LZLY17] Rui Li, Qinghua Zheng, Xiuqi Li, and Zheng Yan. Multi-objective optimization for rebalancing virtual machine placement. *Future Generation Computer Systems*, 2017.

[Man15] Zoltán Ádám Mann. Allocation of virtual machines in cloud data centers—a survey of problem models and optimization algorithms. *ACM Comput. Surv.*, 48(1):11:1–11:34, August 2015.

[MC96] Theodore W Manikas and James T Cain. Genetic algorithms vs. simulated annealing: A comparison of approaches for solving the circuit partitioning problem. 1996.

[MFD11] K. Mills, J. Filliben, and C. Dabrowski. Comparing vm-placement algorithms for on-demand clouds. In *2011 IEEE Third International Conference on Cloud Computing Technology and Science*, pages 91–98, Nov 2011.

[MG10] Peter Mell and Tim Grance. The nist definition of cloud computing. *Communications of the ACM*, 53(6):50, 2010.

[MM17] M. A. H. Monil and A. D. Malony. Qos-aware virtual machine consolidation in cloud datacenter. In *2017 IEEE International Conference on Cloud Engineering (IC2E)*, pages 81–87, April 2017.

[MMP+14] D. More, S. Mehta, P. Pathak, L. Walase, and J. Abraham. Achieving energy efficiency by optimal resource utilisation in cloud environment. In *2014 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)*, pages 1–8, Oct 2014.

[MP16] Abdelkhalik Mosa and Norman W Paton. Optimizing virtual machine placement for energy and sla in clouds using utility functions. *Journal of Cloud Computing*, 5(1):17, 2016.

[MR16] Mohammad Alaul Haque Monil and Rashedur M Rahman. Vm consolidation approach based on heuristics fuzzy logic, and migration control. *Journal of Cloud Computing*, 5(1):1–18, 2016.

[MS17a] Zoltán Adám Mann and Máté Szabó. Which is the best algorithm for virtual machine placement optimization? *Concurrency and Computation: Practice and Experience*, 29(10):e4083, 2017.

[MS17b] Abdelkhalik Mosa and Rizos Sakellariou. Virtual machine consolidation for cloud data centers using parameter-based adaptive allocation. In *Proceedings of the Fifth European Conference on the Engineering of Computer-Based Systems*, ECBS '17, pages 16:1–16:10, New York, NY, USA, 2017. ACM.

[MS18] A. Mosa and R. Sakellariou. Dynamic tuning for parameter-based virtual machine placement. In *2018 17th International Symposium on Parallel and Distributed Computing (ISPDC)*, pages 38–45. IEEE, June 2018.

[MS19] Abdelkhalik Mosa and Rizos Sakellariou. Dynamic virtual machine placement considering cpu and memory resource requirements. In *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, pages 196–198. IEEE, 2019.

[MSY12] S. T. Maguluri, R. Srikant, and L. Ying. Stochastic models of load balancing and scheduling in cloud computing clusters. In *2012 Proceedings IEEE INFOCOM*, pages 702–710, March 2012.

[NFYJ18] T. H. Nguyen, M. Di Francesco, and A. Yla-Jaaski. Virtual machine consolidation with multiple usage prediction for energy-efficient cloud data centers. *IEEE Transactions on Services Computing*, pages 1–1, 2018.

[NnVPC+12] Alberto Núñez, Jose L. Vázquez-Poletti, Agustin C. Caminero, Gabriel G. Castañé, Jesus Carretero, and Ignacio M. Llorente. ICanCloud: A Flexible and Scalable Cloud Infrastructure Simulator. *Journal of Grid Computing*, 10(1):185–209, 2012.

[Nrd15] Nrdc.org. America's Data Centers Consuming and Wasting Growing Amounts of Energy, 2015.

[NS10] TR Nair and Kavitha Sooda. Comparison of genetic algorithm and simulated annealing technique for optimal path selection in network routing. *arXiv preprint arXiv:1001.3920*, 2010.

[PS16] Ilia Pietri and Rizos Sakellariou. Mapping virtual machines onto physical machines in cloud computing: A survey. *ACM Computing Surveys (CSUR)*, 49(3):49, 2016.

[QHNN+13] Nguyen Quang-Hung, Pham Dac Nien, Nguyen Hoai Nam, Nguyen Huynh Tuong, and Nam Thoai. A genetic algorithm for power-aware virtual machine allocation in private cloud. In *Information and Communication Technology-EurAsia Conference*, pages 183–191. Springer, 2013.

[Rap04] Michael A Rappa. The utility business model and the future of computing services. *IBM systems journal*, 43(1):32–42, 2004.

[RCTY19] Xiaojun Ruan, Haiquan Chen, Yun Tian, and Shu Yin. Virtual machine allocation and migration based on performance-to-power ratio in energy-efficient clouds. *Future Generation Computer Systems*, 100:380–394, 2019.

[REP+13] M. G. Rabbani, R. P. Esteves, M. Podlesny, G. Simon, L. Z. Granville, and R. Boutaba. On tackling virtual data center embedding problem. In *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*, pages 177–184, May 2013.

[RGSCD+16] Víctor Francisco Rodríguez Galiano, Manuel Sánchez Castillo, Jadunandan Dash, Peter Atkinson, and José Ojeda Zújar. Modelling interannual variation in the spring and autumn land surface phenology of the european forest. *Biogeosciences, 13, 3305-3317.*, 2016.

[RI94] R. L. Rao and S. S. Iyengar. Bin-packing by simulated annealing. *Computers and Mathematics with Applications*, 27(5):71–82, 1994.

[RKT19] Somayeh Rahmani, Vahid Khajehvand, and Mohsen Torabian. Burstiness-aware virtual machine placement in cloud computing systems. *The Journal of Supercomputing*, pages 1–26, 2019.

[SB03] R. Sterritt and D. Bustard. Towards an autonomic computing environment. *14th International Workshop on Database and Expert Systems Applications, 2003. Proceedings.*, 2003.

[SBBJ13] L. Shi, B. Butler, D. Botvich, and B. Jennings. Provisioning of requests for virtual machine sets with placement constraints in iaas clouds. In *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*, pages 499–505, May 2013.

[SBV+15] Erwan Scornet, Gérard Biau, Jean-Philippe Vert, et al. Consistency of random forests. *The Annals of Statistics*, 43(4):1716–1741, 2015.

[SDJ99] Randall S Sexton, Robert E Dorsey, and John D Johnson. Optimization of neural networks: A comparative analysis of the genetic algorithm and simulated annealing. *European Journal of Operational Research*, 114(3):589–601, 1999.

[SH11] W. Shi and B. Hong. Towards profitable virtual machine placement in the data center. In *2011 Fourth IEEE International Conference on Utility and Cloud Computing*, pages 138–145, Dec 2011.

[Sta14] Dustin Stansbury. A gentle introduction to artificial neural networks. `https://theclevermachine.wordpress.com/2014/09/11/a-gentle-introduction-to-artificial-neural-networks/`, 2014.

[Sta15] M Stansberry. Uptime institute data center industry survey 2015 - focus on budgets, planning, metrics, and life safety. `https://www.scribd.com/document/293839439/uptime-report-pdf`, 2015.

[SW12] Thomas Setzer and Andreas Wolke. Virtual machine re-assignment considering migration overhead. *Proceedings of the 2012 IEEE Network Operations and Management Symposium, NOMS 2012*, pages 631–634, 2012.

[TCTB11] Adel Nadjaran Toosi, Rodrigo N Calheiros, Ruppa K Thulasiram, and Rajkumar Buyya. Resource Provisioning Policies to Increase IaaS Provider's Profit in a Federated Cloud Environment. In *High Performance Computing and Communications (HPCC), 2011 IEEE 13th International Conference on*, pages 279–287. IEEE, 2011.

[TKBL12] M. Tighe, G. Keller, M. Bauer, and H. Lutfiyya. Dcsim: A data centre simulation tool for evaluating dynamic virtualized resource management. In *2012 8th international conference on network and service management (cnsm) and 2012 workshop on systems virtualiztion management (svm)*, pages 385–392, Oct 2012.

[TLK+18] Nikos Tziritas, Thanasis Loukopoulos, Samee Khan, Cheng-Zhong Xu, and Albert Zomaya. A communication-aware energy-efficient graph-coloring algorithm for vm placement in clouds. In *2018 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*, pages 1684–1691. IEEE, 2018.

[TXC+15] Wenhong Tian, Minxian Xu, Aiguo Chen, Guozhong Li, Xinyang Wang, and Yu Chen. Open-source simulators for cloud computing: Comparative study and challenging issues. *Simulation Modelling Practice and Theory*, 58:239–254, 2015.

[TZX+15] W. Tian, Y. Zhao, M. Xu, Y. Zhong, and X. Sun. A toolkit for modeling and simulation of real-time virtual machine allocation in a cloud data center. *IEEE Transactions on Automation Science and Engineering*, 12(1):153–161, Jan 2015.

[Uni10] University of Luxembourg. GreenCloud - The green cloud simulator, 2010.

[VRMB11] Luis M Vaquero, Luis Rodero-Merino, and Rajkumar Buyya. Dynamically scaling applications in the cloud. *ACM SIGCOMM Computer Communication Review*, 41(1):45–52, 2011.

[WBS16] Andreas Wolke, Martin Bichler, and Thomas Setzer. Planning vs. dynamic control: Resource allocation in corporate clouds. *IEEE Transactions on Cloud Computing*, 4(3):322–335, 2016.

[WD14] Josh Whitney and Pierre Delforge. Nrdc: Data center efficiency assessment - scaling up energy efficiency across the data center industry: Evaluating key drivers and barriers, 2014.

[WGB11] Linlin Wu, Saurabh Kumar Garg, and Rajkumar Buyya. SLA-based resource allocation for software as a service provider (SaaS) in cloud computing environments. In *Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on*, pages 195–204. IEEE, 2011.

[Whi94] Darrell Whitley. A genetic algorithm tutorial. *Statistics and Computing*, 4(2):65–85, 1994.

[WLFJ13] Xiaoying Wang, Xiaojing Liu, Lihua Fan, and Xuhan Jia. A decentralized virtual machine migration approach of data centers for cloud computing. *Mathematical Problems in Engineering*, 2013, 2013.

[WLZ+13] S. Wang, Z. Liu, Z. Zheng, Q. Sun, and F. Yang. Particle swarm optimization for energy-aware virtual machine placement optimization in virtualized data centers. In *2013 International Conference on Parallel and Distributed Systems*, pages 102–109, Dec 2013.

[WMZ11] Meng Wang, Xiaoqiao Meng, and Li Zhang. Consolidating virtual machines with dynamic bandwidth demand in data centers. In *INFOCOM, 2011 Proceedings IEEE*, pages 71–75. IEEE, 2011.

[WT18] Hui Wang and Huaglory Tianfield. Energy-aware dynamic virtual machine consolidation for cloud datacenters. *IEEE Access*, 6:15259–15273, 2018.

[WTAPB15] Andreas Wolke, Boldbaatar Tsend-Ayush, Carl Pfeiffer, and Martin Bichler. More than bin packing: Dynamic resource allocation strategies in cloud data centers. *Information Systems*, 52:83–95, 2015.

[WYS13] Fetahi Wuhib, Rerngvit Yanggratoke, and Rolf Stadler. Allocating Compute and Network Resources Under Management Objectives in Large-Scale Clouds. *Journal of Network and Systems Management*, pages 1–26, 2013.

[XF10a] Jing Xu and José a B Fortes. Multi-objective virtual machine placement in virtualized data center environments. *Proceedings - 2010 IEEE/ACM International Conference on Green Computing and Communications, GreenCom 2010, 2010 IEEE/ACM International Conference on Cyber, Physical and Social Computing, CPSCom 2010*, pages 179–188, 2010.

[XF10b] Jing Xu and Jose AB Fortes. Multi-objective virtual machine placement in virtualized data center environments. In *2010 IEEE/ACM Int'l Conference on Green Computing and Communications & Int'l Conference on Cyber, Physical and Social Computing*, pages 179–188. IEEE, 2010.

[XF11] Jing Xu and José Fortes. A multi-objective approach to virtual machine management in datacenters. *Proceedings of the 8th ACM international conference on Autonomic computing - ICAC '11*, page 225, 2011.

[XHL19] Hui Xiao, Zhigang Hu, and Keqin Li. Multi-objective vm consolidation based on thresholds and ant colony system in cloud computing. *IEEE Access*, 7:53441–53453, 2019.

[XSC13] Zhen Xiao, Weijia Song, and Qi Chen. Dynamic resource allocation using virtual machines for cloud computing environment. *IEEE transactions on parallel and distributed systems*, 24(6):1107–1117, 2013.

[XWW09] Tianze Xu, Heng Wei, and Zhuan-De Wang. Study on continuous network design problem using simulated annealing and genetic algorithm. *Expert Systems with Applications*, 36(2):2735–2741, 2009.

[YRJ] Amin Yousefipour, Amir Masoud Rahmani, and Mohsen Jahanshahi. Energy and cost-aware virtual machine consolidation in cloud computing. *Software: Practice and Experience*.

[YYL17] Xin Ye, Yanli Yin, and Lan Lan. Energy-efficient many-objective virtual machine placement optimization in a cloud computing environment. *IEEE Access*, 5:16006–16020, 2017.

[ZCB10] Qi Zhang, Lu Cheng, and Raouf Boutaba. Cloud computing: state-of-the-art and research challenges. *Journal of internet services and applications*, 1(1):7–18, 2010.

[Zei12] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[ZHL16] Zhou Zhou, Zhigang Hu, and Keqin Li. Virtual machine placement algorithm for both energy-awareness and sla violation reduction in cloud data centers. *Sci. Program.*, 2016:15–, March 2016.

[ZLD+15] Q. Zheng, J. Li, B. Dong, R. Li, N. Shah, and F. Tian. Multi-objective optimization algorithm based on bbo for virtual machine consolidation problem. In *2015 IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS)*, pages 414–421, Dec 2015.

[ZPXD12] Wei Zhao, Yong Peng, Feng Xie, and Zhonghua Dai. Modeling and simulation of cloud computing: A review. *Proceedings - 2012 IEEE Asia Pacific Cloud Computing Congress, APCloudCC 2012*, pages 20–24, 2012.

[ZWL+18] Hui Zhao, Jing Wang, Feng Liu, Quan Wang, Weizhan Zhang, and Qinghua Zheng. Power-aware and performance-guaranteed virtual machine placement in the cloud. *IEEE Transactions on Parallel and Distributed Systems*, 29(6):1385–1400, 2018.

# Appendix A

# Simulated Annealing vs Genetic Algorithm for Dynamic Reallocation of VMs in Cloud Data Centres

## A.1 Introduction

This appendix aims to conduct an empirical evaluation of two well-known metaheuristic search techniques for the dynamic reallocation of VMs among PMs in cloud data centres. These two metaheuristics are a genetic algorithm (GA) and simulated annealing (SA). Based on the current resource demand, the two metaheuristics will seek to reallocate VMs among CDC's PMs dynamically for saving energy and reducing SLAVs. Firstly, this appendix describes the problem of the dynamic demand-based reallocation of VMs, followed by the definition of the cost function. Then, the appendix introduces GA and SA algorithms and describes the dynamic reallocation of VMs using both metaheuristics based on the specified cost function. The main concern is to compare the energy consumption, percentage of SLA violations, and the execution time between the two metaheuristics search techniques. By and large, the conducted experiments conclude that GA outperforms SA for the dynamic reallocation of VMs. The results confirm that, for the same amount of time, the GA implementation is better than the SA regarding the amount of energy consumed and the percentage of SLAVs. The results conclude that GA better suits the dynamic reallocation of the VMs as opposed to SA.

Mosa *et al*. [MP16] proposed a utility-based solution that dynamically reallocates VMs to reduce the VM placement cost by saving energy consumption and reducing

SLA violations (SLAVs). This utility-based solution makes use of a genetic algorithm (GA) to search for a robust solution that achieves the cloud provider's objectives. In this appendix, we have revisited the search algorithm (searching for a better VMs-to-PMs mapping) for the same problem using the simulated annealing (SA). There are common similarities between GA and SA, even though they are using entirely different terminologies. Theoretically, a simulated annealing algorithm could be considered as equal to a genetic algorithm with only one solution, which means there is no crossover operation. The type of the problem and the representation of the solution determine which algorithm could be more beneficial. Bost GA and SA have been empirically evaluated against each other in literature for different application domains such as circuit partitioning [MC96], optimal path selection in network routing [NS10], training neural networks [SDJ99], and continuous network design [XWW09]. From these previous studies on GA versus SA, there is no generally accepted superiority of GA over SA. However, such superiority could exist in specific application domains. The primary motivation for evaluating SA for the dynamic reallocation of VMs is that SA was proven to find efficient solutions for NP-complete problems [Kir84].

## A.2 Dynamic Reallocation of VMs

The dynamic reallocation of VMs involves searching several potential VMs-to-PMs assignments to find a robust assignment that achieves the cloud provider's goals. Once the dynamic reallocation algorithm finds a more efficient VMs-to-PMs assignment, it starts reallocating the VMs to the new PMs based on the best-found assignment. Table A.1, shows an example of a possible mapping of VMs to PMs. The VM number represents the mapping index, while the PM number represents the PM that is going to host the VM.

Table A.1: An example of VMs to PMs assignment

| $VM_1$ | $VM_2$ | $VM_3$ | $VM_4$ | ... | ... | ... | $VM_n$ |
|---|---|---|---|---|---|---|---|
| $PM_1$ | $PM_2$ | $PM_1$ | $PM_2$ | ... | ... | ... | $PM_m$ |

The dynamic reallocation of VMs aims to place the VMs on the least possible number of PMs and switch unused PMs off to save energy consumption and reduce operational costs. Figure A.1 shows the current VMs-to-PMs mapping as an input to the dynamic VM reallocation solution, which uses either GA or SA to search for an

efficient mapping based on the cost function. Both GA and SA explore the search space to find a more suitable assignment that saves energy and reduces SLA violations. The dynamic VMs reallocation algorithm runs periodically at fixed time intervals, which are five minutes of simulated time. Single objective GA and SA can optimise only one single variable. Therefore, the following section defines the cost function, which GA and SA are going to use to measure the fitness of any candidate VMs-to-PMs assignment.

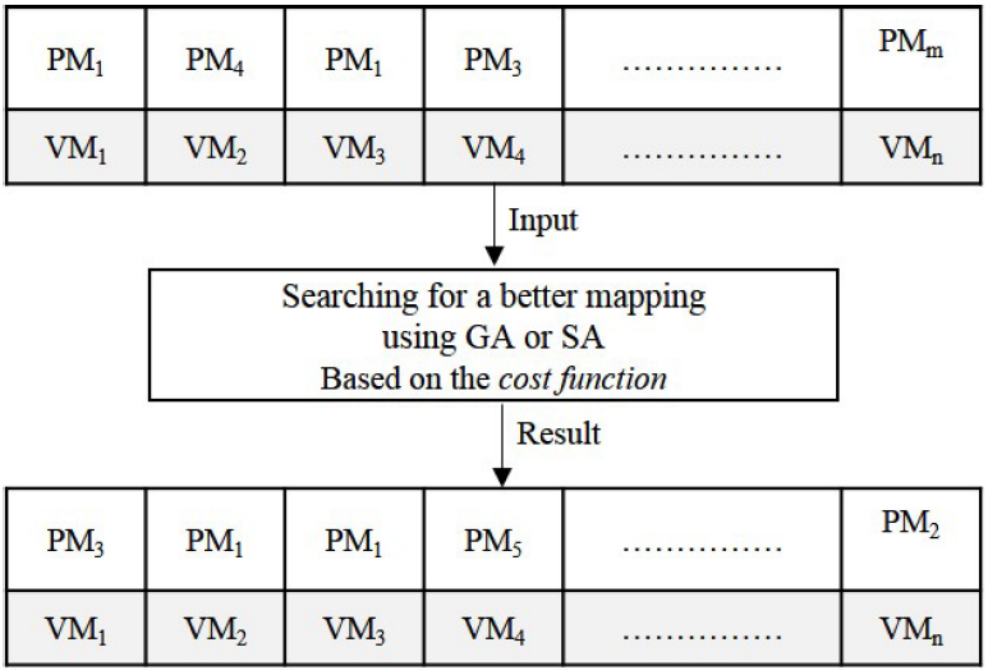



Figure A.1: Input, search and output for the dynamic VM reallocation

### A.2.1 Cost Function

Equation A.1, from [MP16], defines the cost function, which aims to minimize the costs of energy consumption and SLA violations. The ultimate goal is to minimize the total cost of hosting the VMs to maximize the cloud provider's profit. The fitness of any potential VMs-to-PMs assignments using either GA or SA are evaluated against the cost function.

$$Cost(a,t) = EstimatedEnergyCost(a,t) + EstimatedViolationCost(a,t) + PDMCost(a,t)). \tag{A.1}$$

Here $a$ is a map that represents the VMs-to-PMs assignment, and $t$ is the time period of the assignment. $EstimatedEnergyCost(a,t)$ is the expected energy cost due to the assignment. $EstimatedViolationCost(a,t)$ is the expected violation cost resulting from the overutilization of the PM. $PDMCost(a,t)$ is the cost of the *performance degradation due to the migration* (PDM). The calculation details of energy and SLA violation costs can be found in [MP16]. The following section describes the details of GA and SAL for the dynamic reallocation of VMs in cloud data centres.

## A.3 Genetic Algorithm and Simulated Annealing

### A.3.1 Genetic Algorithm

Genetic algorithms are population-based search techniques that mimic Charles Darwins theory of natural evolution [Luk15]. Genetic algorithms were introduced in 1970 by John Holland. A genetic algorithm (GA) starts with a population consisting of several individuals where each individual represents a candidate solution in the problem being solved. The cost function measures the fitness of each of these candidate solutions. For creating a new generation, some solutions are selected for crossover to reproduce new offsprings. The mutation is applied, and the process continues until reaching the specified stopping criteria.

#### Creating Initial Population

The initial population is created by generating a population of $N$ individuals, where $N$ is the *population size*. For the VMs reallocation problem, each individual, *candidate solution*, is represented by VMs-to-PMs assignment/mapping. Generally, the initial population is either randomly generated by allowing the entire range of possible solutions. However, in this appendix, the initial population has been generated by mutating the VMs in the current VMs-to-PMs assignment. The reason for starting with the current VMs-to-PMs assignment is that the algorithm can keep it if there is no better solution.

#### Fitness Evaluation

The cost (fitness) measures the quality of each candidate solution proposed by the metaheuristic algorithm. The evaluation of the cost function involves assigning a single value (fitness score) to each candidate solution. The probability that a candidate

solution (individual) will be selected for reproduction depends on its fitness score. The candidate solution with the lowest cost is considered to be the best. The cost function, introduced in Section A.2.1, is used to measure the quality of each candidate solution.

### Next Generation

The genetic algorithm runs for several times based on the specified number of generations. Three central operations are required for creating the next generation namely *parents selection*, *crossover* and *mutation*. The details of these operations are described below:

- Parents Selection:
  For applying the crossover operation, parents need to be chosen from the current population. Parents selection involves selecting candidate VMs-to-PMs assignments (representing parents). Each candidate solution could be selected as a parent, and individuals with high fitness scores might be favourable for selection.

- Crossover:
  The crossover operation works by selecting two parent solutions and applying a crossover technique. The crossover probability indicates how many parent solutions, individuals, are selected for mating [Whi94]. The implementation of GA in this work adopts a single-point crossover where two offspring are produced by swapping beyond the crossover point. Table A.2, illustrates how the new offspring is created by mating two parents using single-point crossover. The "||" symbol represents the crossover point.

Table A.2: Single point crossover

| | |
|---|---|
| **Parent**$_1$ | $PM_1$ $PM_2$ $PM_1$ \|\| $PM_3$ $PM_2$ $PM_1$ |
| **Parent**$_2$ | $PM_2$ $PM_1$ $PM_3$ \|\| $PM_1$ $PM_3$ $PM_3$ |
| **Offspring**$_1$ | $PM_1$ $PM_2$ $PM_1$ \|\| $PM_1$ $PM_3$ $PM_3$ |
| **Offspring**$_2$ | $PM_2$ $PM_1$ $PM_3$ \|\| $PM_3$ $PM_2$ $PM_1$ |

- Mutation:
  The mutation process alters the values of an existing solution based on the mutation probability. Mutation is applied to new offsprings generated from the crossover operation [CSP11]. A number of candidate solutions (VMs-to-PMs

assignments) of the current population are selected for mutation. New mutated solutions are created as a result of the mutation process. In the resulted mutated solutions, some VMs are migrated to different PMs. The number of the mutated VMs depends on the mutation probability, which defines how many PMs, to which the VMs are assigned, need to be changed to new ones. Table A.3 demonstrates an example of how the mutation works. The PMs in bold in the original offspring are mutated to new ones in the mutated offspring, which means that some VMs are assigned to new PMs.

Table A.3: Mutating the PMs

| | |
|---|---|
| **Original Offspring** | $PM_1$ $PM_2$ $PM_1$ ... $PM_1$ $PM_3$ $PM_3$ |
| **Mutated Offspring** | $PM_1$ $PM_4$ $PM_1$ ... $PM_2$ $PM_2$ $PM_3$ |

**Stopping Criteria**

The generation of new fitter candidate solutions is repeated until reaching the stopping criteria. Common examples of stopping criteria include (1) fixed number of generations, (2) satisfying minimum criteria, (3) reaching the maximum execution time, or (4) the fittest solution reaches a plateau (further generations do not improve the result). The stopping criteria for our implementation of the genetic algorithm is based on a fixed *number of generations*. The algorithm proceeds to search for more competitive solutions until the number of generation becomes zero. After reaching the stopping criteria, the algorithm returns the best solution found so far.

### A.3.2 Simulated Annealing Algorithm

Simulated annealing (SA), also known as statistical cooling, is a general-purpose stochastic optimisation technique which helps finding near-optimal solutions for NP-complete problems [RI94]. In 1983, Kirkpatrick etal [KGV83] introduced SA by mimicking the process of annealing in metallurgy. Simulated annealing, a generalization of the Monte Carlo method, finds global minimum cost solutions by exploiting the analogy between the physical states of the system till reaching thermal equilibrium and the intermediate stages of a solution of a combinatorial problem. SA approximates the global optimum of a given function. SA performs an iterative refinement, and it can accept non-optimal solutions (*i.e.* ones that increase the objective, depending on some probability) to avoid getting stuck into local minima.

Algorithm A.3.2 shows pseudo-code for the simulated annealing algorithm, which is used for reallocating VMs with the goal of energy-saving and reducing SLA violations. As a requirement, the SA algorithm needs to clearly define the initial configuration/solution, initial temperature, final temperature, the temperature decrease factor, the number of iterations per temperature, and the acceptance probability. The details of these operations are defined below:

**Algorithm 5** Simulated annealing algorithm for VM reallocation

```
1:  function VMREALLOCATIONUSINGSA
2:      Save currentVmToPmAllocation
3:      currentAllocation ← currentVmToPmAllocation
4:      bestAllocation ← currentAllocation
5:      C(Allocation_current) ← currentAllocationCost
6:      C(Allocation_neighbor) ← neighborAllocationCost
7:      T_initial ← initialTemperature
8:      T_min ← finalTemperature
9:      for t = T_initial; t > T_min; t = t * decreaseFactor do
10:         for i = 0; i < numberOfIterationsPerTemp; i++ do
11:             neighborAllocation ← selectNeighbor()
12:             Δ(C) ← C(Allocation_current) − C(Allocation_neighbor)
13:             if Δ(C) > 0 then
14:                 currentAllocation ← neighborAllocation
15:             else if Δ(C) < 0 then
16:                 acceptanceProbability ← e^(Δ(C)/t)
17:                 randomValue ← getRandomDouble()
18:                 if randomValue < acceptanceProbability then
19:                     currentAllocation ← neighborAllocation
20:                     if C(S_current) < C(S_best) then
21:                         bestAllocation ← currentAllocation
```

- **Initial Configuration**:
  Typical implementations of SA start with a random configuration that represents the initial solution. However, it might be better to start with an existing solution instead of randomly creating an initial solution. Therefore, we have used the current solution, which is the current VMs-to-PMs mapping from either the initial VM placement or from the previous interval of the dynamic placement. The main idea of using an existing solution is that the algorithm can keep it if there is no better solution and hence minimize the number of migrations. Lines 2 and 3 in Algorithm A.3.2 sets the initial configuration by saving the current VMs-to-PMs allocation as the best allocation.

- **Initial Temperature**:
  There is no widely accepted method for setting the initial temperature of the SA algorithm. On the one hand, the initial temperature should not be too low as this will not enable the algorithm to search for numerous neighbours. Choosing too low initial temperature may result in a final solution that is near to initial one without significant improvements. On the other hand, if the initial temperature is higher than required, then this might lead to a kind of random search, especially in the early stages. Line 7 in Algorithm A.3.2 sets the initial temperature.

- **Final Temperature**:
  The SA algorithm usually starts with the initial temperature; then, the temperature is reduced until a final temperature is reached. The final temperature should be close to zero; however, the SA algorithm should not wait until it reaches zero as this might take longer time without noticeable enhancement toward the final solution. Line 8 in Algorithm A.3.2 defines the final temperature.

- **Temperature Decrease Factor**:
  The temperature decrease factor affects the quality of the final solution, as a low decrease increases the execution time. In contrast, a rather high decrease may result in local minima. In our SA implementation, we adopt a geometric decrement to ensure that the algorithm runs fewer times for higher values and more times for lower ones. The geometric decrement is:

  $$temperature = temperature * \beta$$

  , where $0 < \beta < 1$; the algorithm will take more time when approaches one as the cooling rate will be slower. Previous experiments confirm that using a decrease factor in the range between 0.8 and 0.99 produces good results – a decrease factor with a value of 0.95 results in high-quality solutions for the bin packing problem [RI94]. The geometric decrement of the temperature is applied on the loop in Line 9 of Algorithm A.3.2.

- **Number of Iterations per Temperature**:
  For each iteration, a VM is randomly selected from the currently running VMs and assigned to a non-overutilized PM. This selection happens a fixed number of times based on the number of iterations per temperature. The number of iterations per temperature starts in the loop in Line 10 of Algorithm A.3.2.

- **Acceptance Probability**:
  The algorithm accepts the new candidate VMs-to-PMs assignment/allocation if the cost of the candidate assignment is less than the cost of the current one. As shown in Equation A.2 and Lines 12 to 14 of Algorithm A.3.2, the candidate solution is accepted if the difference between the cost of the current allocation and the candidate one is greater than zero.

$$\Delta(C) = cost(currentAllocation) - cost(neighbourAllocation). \tag{A.2}$$

  To avoid local minima, the algorithm can accept a worse solution under a specific condition. A worse solution that incurs a higher cost can be accepted based on a probability that depends on the Boltzmann distribution, which mimics the physical annealing. Equation A.3, and in Lines 15 to 19 of Algorithm A.3.2, show the probability of accepting a solution.

$$P(\Delta(C)) = e^{\Delta(C)/current\_temperature}. \tag{A.3}$$

  If the difference between the two solutions is less than zero, then the algorithm calculates the acceptance probability and compare it to a random double between zero and one. If the random value is less than the acceptance probability, then the new solution will be accepted.

## A.4 Experimental Evaluation

This section evaluates GA and SA for the dynamic reallocation of the VMs. It starts by describing the simulation settings, the performance metrics, and concludes with conducted experiments.

### A.4.1 Simulation Settings

The simulation environment represents a cloud data center consisting of 100 PMs and 200 VMs. The PMs are of two types HP ProLiant ML110 G4 and G5. The HP ProLiant ML110 G4 has two CPU cores with 1860 MIPS per core, and the G5 also has two cores with 2660 per core. The VMs are of four types; with 2500, 2000, 1000, and 500 MIPs for the first, second, third, and the fourth type of VMs [BB12]. The application data are randomly generated by CloudSim cloudlets. The reallocation of VMs happens every

five minutes, and the simulation of the experiments runs for one day of simulation time. This means there are 288 cycles/runs of the reallocation algorithm; 12 cycles per hour multiplied by 24 hours. All the experiments have been running on a Core i7 laptop with six GB of RAM.

### A.4.2 Performance Metrics

Three main performance metrics are used for the evaluation:

- Energy Consumption: The amount of energy consumed by the data centre in kWh.
- Overall SLA violations: The overall SLA violations for all the running VMs which is computed as follows:

$$overallSlaViolations = \frac{totalMipsReuestedByVMs - totalAllocatedMips}{totalMipsReuestedByVMs}$$

- Execution Time: The total execution time of running the experiment using both the genetic algorithm and the simulated annealing.

### A.4.3 Experiments

A total number of 400 experiments have been carried out on both GA and SA using different values for energy and violation costs. These experiments enabled us to measure the effectiveness of GA versus SA with the help of the previously stated performance metrics.

#### Genetic Algorithm versus Simulated Annealing without Limiting Execution Time

The scatter plot in Figure A.2 exhibits energy consumption to SLA violations using genetic algorithm and simulated annealing algorithms. The triangles represent results from GA, while the circles represent the ones from the SA algorithm. This scatter plot is a result of around 400 experiments on both algorithms. By and large, Figure A.2 demonstrates that the GA outperforms the SA algorithm. What stands out in Figure A.2 is that GA can reach better values that SA was not able to find without limiting the execution time. Furthermore, it is observed that increasing the total number of iterations for the SA (by increasing initial temperature and the number of iterations per temperature) reaches a plateau and did not show any further improvements.

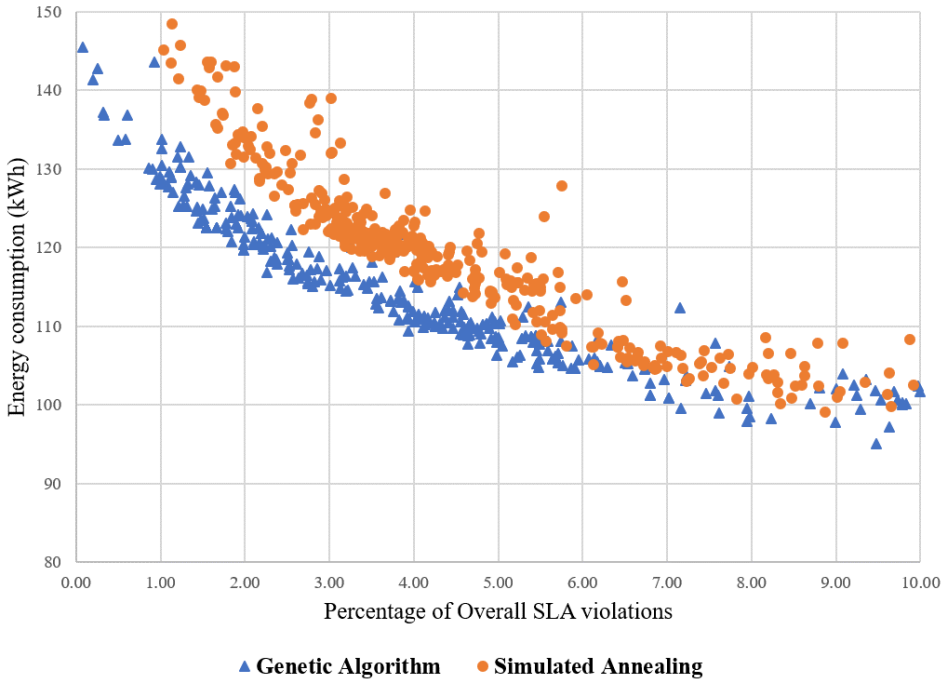



Figure A.2: Energy consumption to overall SLAVs using GA and SA.

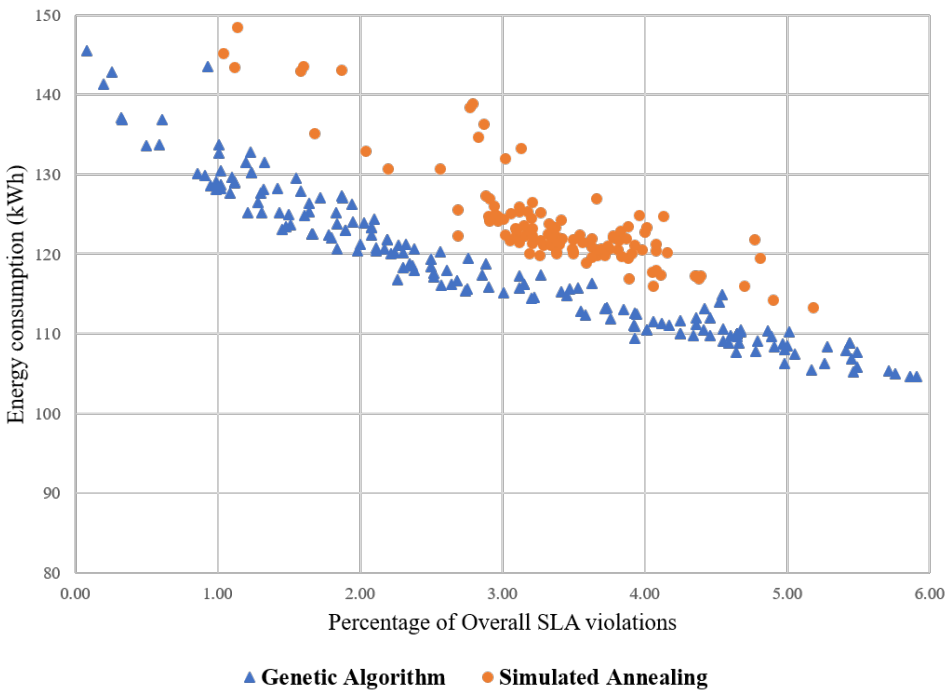



Figure A.3: Energy consumption to overall SLAVs using GA and SA; limiting execution time to one minute.

### Genetic Algorithm versus Simulated Annealing with Execution Time Constraint

This experiment aims to evaluate GA and SA when limiting the execution time. In this experiment, the execution time is limited to one minute to figure out the efficiency of both algorithms with the specified execution time constraint. With no confusion, Figure A.3 demonstrates that, for the same execution time, the genetic algorithm is more efficient than the simulated annealing algorithm for the dynamic reallocation of VMs in cloud data centres.

## A.5 Conclusion

This appendix follows a utility-based approach for the dynamic reallocation of VMs in cloud data centres. An empirical evaluation has been conducted to compare between a genetic algorithm and a simulated annealing algorithm for the VMs reallocation problem. Both the genetic algorithm and simulated annealing algorithm attempt to dynamically find a better VMs-to-PMs assignment that achieves the cloud provider's goals. Results demonstrate that using a genetic algorithm outperforms SA for the dynamic reallocation of the VMs in terms of energy-saving, reduction of SLA violations, and the execution time. Furthermore, results show that for the same execution time, the GA outperforms the SA. Accordingly, this appendix concludes that using genetic algorithms is recommended over simulated annealing for the dynamic reallocation of VMs in cloud data centres.

# Appendix B

# Dynamic Virtual Machine Placement Considering CPU and Memory Resource Requirements

## B.1 Introduction

A high proportion of current VM placement solutions examine only CPU resources and ignore other types of resources such as memory, storage, or network bandwidth. Currently, many VMs may be either memory-intensive or bandwidth-intensive, which suggests that the CPU is not the most important resource type in such cases. Therefore, considering different types of resources (or characteristics), including memory or network bandwidth, becomes critical. For example, Amazon EC2 offers VM instances that are optimised for computation, memory, or storage [Ama].

This appendix examines the problem of allocating VMs along more than one of these characteristics by proposing a dynamic demand-based VM placement solution that considers both CPU and memory requirements. The proposed solution adopts a genetic algorithm (GA) aiming to reduce underutilization as well as overutilization situations of both CPU and memory, which ensures better overall utilization and a reduction in the resulting SLAVs. Empirical evaluation using CloudSim highlights the importance of considering multiple resource types. In addition, it demonstrates that the genetic algorithm outperforms the well-known best-fit decreasing algorithm for dynamic VM placement. The work on this appendix is based on the paper [MS19].

A large body of literature has investigated the use of metaheuristics to reallocate VMs among PMs dynamically based on predefined objective functions. Some of these

solutions adopt a genetic algorithm [JCC15, MP16], an ant colony system [FAP+15], while others use particle swarm optimization [WLZ+13]. However, many of these solutions only consider a single type of resource, often CPU [BB12, WLZ+13, MP16, WLFJ13]. There are other solutions that consider multiple resources using a genetic algorithm. However, they only solved the initial VM placement and did not consider the dynamic reallocation of the VMs after they are initially allocated [XF10b]. Therefore, this appendix proposes a CPU- and memory-aware dynamic VM placement solution based on a genetic algorithm.

## B.2 Placement Objective

The main objective for the VM placement is to utilize the PMs efficiently while meeting SLA requirements to reduce energy consumption and operational costs. Therefore, the objective function seeks to utilize the PMs efficiently by minimizing the estimated underutilization per each resource type (CPU and memory), while reducing SLAVs by minimizing the estimated overutilization at each scheduling interval. Underutilization is defined as the ratio of available (or unused) capacity divided by the total capacity of a given resource type. However, Overutilization is defined as the demand that exceeds the capacity of a given resource type.

$$\text{Minimize } \left(\sum_{r=1}^{n} (\bar{W}^r + \bar{O}^r)\right) \tag{B.1}$$

In Equation B.1, $n$ represents the number of resource types (CPU and memory in this appendix, but this could be generalized to storage, bandwidth). The average underutilization per resource type $r$ of all PMs running at a scheduling interval is represented by $\bar{W}^r$. Finally, $\bar{O}^r$ is the average overutilization per resource type $r$ for all PMs running on the current scheduling interval.

## B.3 Dynamic VM Placement using a Genetic Algorithm

The proposed dynamic VM placement solution solves both the initial and dynamic VM placement problems. For the initial VM placement, it adopts a power-aware best-fit decreasing (BFD) heuristic [BB12] that allocates VMs based on the resource requirements defined by VM types. Then, the dynamic reallocation of the VMs may happen

during each scheduling interval (periodically every five simulated minutes) using a genetic algorithm. The GA examines possible VMs-to-PMs mappings/assignments, and choose the best one that can reduce underutilization and overutilization based on the objective function defined by Equation B.1.

Algorithm 6 shows the pseudo-code of the GA for the dynamic reallocation of the VMs after initial placement. The GA runs at each scheduling interval for a specified number of times according to the predefined number of generations. At each scheduling interval, the algorithm tries to find a valid source and target PMs. The source and target PMs are the PMs that are either a source or a target of VMs' migrations. Source PMs are the ones that are not switched off and host running VMs, while target PMs are the ones that are not overutilized before of after the migration of the VMs from source PMs. The GA creates the initial population of candidate VMs-to-PMs mappings by mutating the current mapping. For each generation, the GA first selects the parent using tournament selection, applies uniform crossover, mutates the VMs, and evaluates the candidate VMs-to-PMs mapping based on the objective function (Section B.2).

Once the GA finds the best possible VMs-to-PMs mapping for a given scheduling interval, it will start the VM migration process to reflect the changes in the new VMs-to-PMs mapping.

**Algorithm 6** Dynamic reallocation of VMs using a GA

```
1:  Input: schedulingIntervals, generations;
2:  for all (schedulingIntervals) do
3:      Store current VMs-to-PMs mapping;
4:      Find source and target PMs;
5:      Build initial population of VMs-to-PMs mapping;
6:      Evaluate candidate mappings using Equation B.1;
7:      for all (generations) do
8:          Select parents for crossover;
9:          Crossover based on the cross over ratio;
10:         Mutate VMs using only the source PMs (active);
11:         Evaluate candidate mappings using Equation B.1;
12:         Select population for next generation;
13:     Save best allocation found so far;
14:     Build migration map from the best found allocation;
15:     Apply VM migration based on the migration map;
```

## B.4 Experimental Evaluation

### B.4.1 Simulation Settings

The CloudSim simulation toolkit [CRDB09] is used to simulate a cloud data centre, which consists of two types of PMs, namely, HP ProLiant ML110 (two cores at 1860 MHz each) and HP ProLiant ML110 G5 (two cores at 2660 MHz each) that can host four different VM types. CloudSim generates synthetic CPU and RAM utilization traces every five simulated minutes based on a uniform normal distribution with values between zero and one. The simulation works for a whole day, and the dynamic reallocation of the VMs takes place every five minutes (scheduling interval). For testing different capabilities, we also have skewed memory utilization to be more than 50% in some experiments to measure the efficiency of the algorithm in case of memory-intensive VMs.

This appendix compares the GA dynamic reallocation with a baseline BFD heuristic for dynamic VM placement. The baseline BFD for the dynamic reallocation starts migrating VMs from overutilized PMs when the utilization level of the CPU or memory is 100%. Moreover, it chooses VMs with minimum memory sizes for migration from overutilized PMs to minimize the total migration time.

### B.4.2 Performance Metrics

1. *Average overall underutilization (AOUU)*:
   AOUU is the average of ACUU and ARUU, which correspond to average underutilization per CPU and memory, respectively.

2. *Average overall SLA violations (AOSLAVs)*:
   AOSLAVs estimates the average number of SLA violations and is calculated as follows:
   $$AOSLAVs = \frac{1}{2}(OCSLAVs + ORSLAVs),$$
   where OCSLAVs represents CPU SLA violations for all active PMs and is estimated as follows:
   $$OCSLAVs = \frac{RequestedCPU - AllocatedCPU}{RequestedCPU}.$$

The ORSLAVs represents memory SLA violations for all active PMs and is estimated as follows:

$$ORSLAVs = \frac{RequestedRam - AllocatedRam}{RequestedRam}.$$

### B.4.3 Experiments

Two experiments have been conducted to evaluate the performance of the GA proposal against the baseline BDF for the dynamic reallocation of the VMs. The first experiment uses the normal CPU and memory utilization traces, while the second experiment increases VMs' memory requirements to over 50%.

**Experiment 1: Normal CPU and memory utilization**

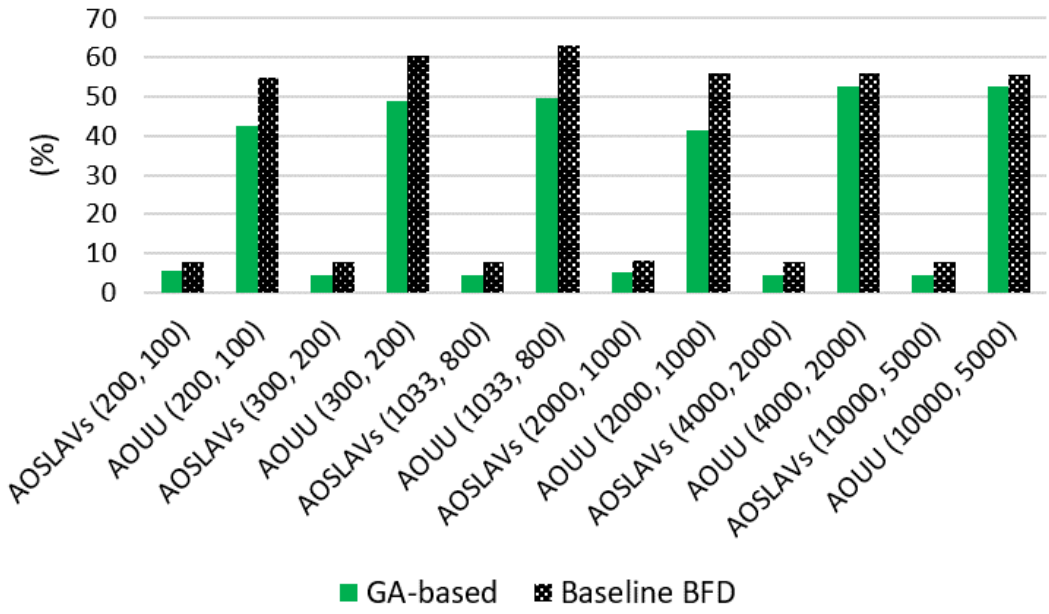



Figure B.1: AOSLAVs and AOUU using different data centre sizes (#VMs,#PMs)

Figure B.1 summarizes the results from the proposed GA-based solution as opposed to the baseline BFD heuristic using different data centre sizes, ranging from 100 to 5000 PMs that host 200 to 10,000 VMs. The X-axis shows the AOSLAVs and AOUU (for each size of the data centre), while the Y-axis is showing their corresponding values as a percentage. The experiments run for a whole day of simulation time except the instances with 4,000 and 10,000 VMs that run for only 6 hours due to

memory constraints. Figure B.1 confirms that GA outperforms BFD for the dynamic reallocation of VMs while considering CPU and memory requirements.

**Experiment 2: Skewed memory utilization**

This experiment skews memory utilization (over 50%) to assess the reaction of both solutions to the increase in memory requirements in case of memory-intensive applications. Figure B.2 shows that the ACUU of the proposed GA-based solution is 45.95% compared to 57.97% of the baseline BFD solution, which means around 20% improvement in CPU utilization. Furthermore, the ARUU is 22.56% and 38.69% for the proposed GA-based solution and the baseline BFD solution, respectively, which indicates that using GA can reduce memory underutilization by around 40% as opposed to the baseline BFD. In Figure B.2, the size of the circle defines the AOSLAVs, which shows that the proposed GA-based solution managed to reduce the AOSLAVs by around 70%.

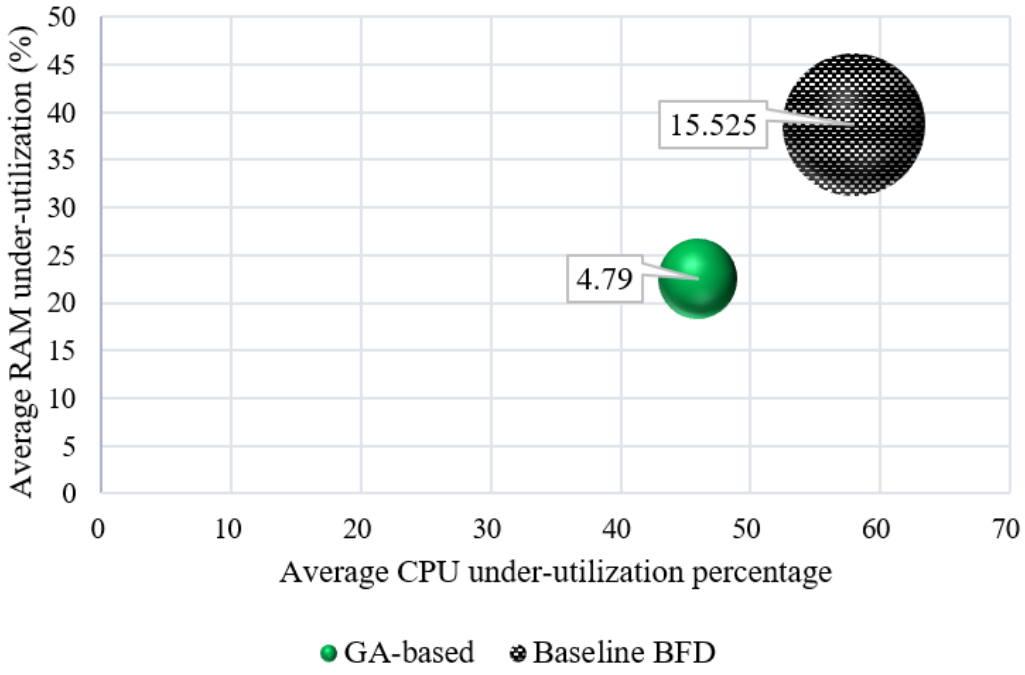



Figure B.2: Average CPU and memory underutilization along with AOSLAVs in a memory-intensive scenario.

## B.5 Conclusion

This appendix addresses the dynamic VM placement problem by considering both CPU and memory resource requirements of the hosted VMs. The appendix makes use of a genetic algorithm for the dynamic reallocation of the VMs. The candidate solutions are evaluated against an objective function that aims to reduce overutilization as well as underutilization of the running PMs to minimize the resulting SLAVs and improve overall utilization. The empirical evaluation proves that the dynamic VM placement using the genetic algorithm outperforms the well-known best-fit decreasing algorithm. Eventually, considering multiple types of resources, including memory or network bandwidth becomes critical as the applications running on the VMs can be memory- or network-intensive.