universität wien

DISSERTATION / DOCTORAL THESIS

Titel der Dissertation /Title of the Doctoral Thesis

„Autonomic Management of Virtual Machines in Cloud Data Centers Using Machine Learning“

verfasst von / submitted by

Seyed Saeid Masoumzadeh

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Doktor der technischen Wissenschaften (Dr. techn.)

Wien, 2017 / Vienna 2017

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	A 786 880
Dissertationsgebiet lt. Studienblatt / field of study as it appears on the student record sheet:	Informatik
Betreut von / Supervisor:	Univ.-Prof. Dipl.-Ing. Dr. Helmut Hlavacs

Acknowledgements

I would like to thank my supervisor Univ. Prof. Dr. Helmut Hlavacs who has guided me during my PhD program, helped me to develop new ideas and given me the support needed to pursue my research unimpeded.

I am also grateful to my research collaborators Luis Tomas Bolivar at Umeå University, Simone Brienza at University of Pisa and Sena Efsun Cebeci at KoÇ University for their help in our joint works and publications. A special appreciation goes to Prof. Erik Elmroth and Prof. Jean-Marc Pierson for hosting my research visit at Umeå University and Institut de Recherche en Informatique de Toulouse (IRIT).

I would like to thank my academic friends Pooyan Jamshdi and Mina Sedaghat who have given me useful hints for my papers and my publications. My big thanks goes to Soodeh Farokhi whose incredible support and advice, especially on my dissertation.

I would like to express my special gratitude to my family who were always there for me when I needed advice or encouragement.

Most of all, I would like to thank the most precious person in my life, my wife Gelareh, for her unconditional love and support during all the years that we have been together, and especially throughout my years of study.

Seyed Saeid Masoumzadeh
Vienna, September 2017

.

To my beloved and brilliant wife, ***Gelareh****, and my loving parents.*

.

Kurzfassung

Cloud Computing ist ein neues Paradigma in der Informations- und Kommunikationstechnologie. Mit dem steigenden Bedarf an Computerressourcen durch Kunden werden Cloud-Rechenzentren immer größer, was zu einem stetig steigenden Energieverbrauch und damit steigenden Betriebskosten führt. Eine Studie über den Stromverbrauch von Rechenzentrums-Arbeitslasten [20] zeigt, dass die Leerlaufleistung in Servern immer mehr als 50% der Spitzenleistung beträgt. Daher ist es aus Sicht des Energieverbrauchs sehr ineffizient, Server nicht voll auszulasten. Energieeffiziente Ressourcenmanagement-Strategien für Cloud-Rechenzentren sind Algorithmen, die Ressourcen auf der Grundlage des Ressourcenbedarfs von Anwendungen so zuweisen, dass die Auslastung pro Server steigt und gleichzeitig die Anwendungsbedingungen im Sinne von Service Level Agreements (SLAs) erfüllt werden.

Energieeffiziente Ressourcenmanagement-Algorithmen lassen sich in zwei allgemeine Klassen einteilen: Die erste Klasse ist die Klasse der Algorithmen, die sich hauptsächlich auf die dynamische Re-Konsolidierung virtueller Maschinen durch Live-Migration in Rechenzentren konzentrieren. Die zweite Klasse ist die Klasse der Algorithmen, die sich auf Zugangskontroll- und/oder Planungsmechanismen konzentrieren. Beide Strategien stellen sich einer Reihe von Herausforderungen, wie z.B. der Optimierung während des dynamischen Konsolidierungsprozesses virtueller Maschinen, den Skalierbarkeitsproblemen und dem Problem der "lauten Nachbarn", das durch VMs in einem überbuchten Rechenzentrum hervorgerufen wird. Diese Arbeit untersucht Methoden des Machine Learnings, von Multi-Agenten-System-Paradigmen und naturinspirierte Algorithmen zur Bewältigung dieser Herausforderungen. Im Rahmen dieser Arbeit wird zunächst das Problem der verteilten dynamischen virtuellen Maschinen-Konsolidierung untersucht, wobei ein kooperativer Multi-Agent Reinforcement Learning Algorithmus zur Overload-Detection und VM Auswahl vorgeschlagen wird. Danach wird eine umfassende Architektur zum Management virtueller Maschinen in Cloud-Rechenzentrem vorgeschlagen. Weiters werden Skalierungseigenschaften bei dynamischer Konsolidierung untersucht, und ein vollständig verteilten Algorithmus vorgeschlagen, der auf sogenannten Gossip-Algorithmen über P2P-Netzwerke beruht. Schließlich wird ein Fuzzy Q-Learning basierten Kapazitätsregler in überbuchten Rechenzentren vorgeschlagen, um das Problem der gegenseitigen Beeinflussungen virtueller Maschinen zu lösen. Die präsentierte Lösung basiert auf Reinforcement Learning, und balanciert die beiden Aspekte Utilization und Performance gegeneinander aus.

Abstract

Cloud computing is a new paradigm in information and communication technology. The key advantage of cloud computing is to provide virtually unlimited resources for costumers based on a pay-as-you-go model. With the rise in demand of computing resources by costumers, cloud data centers evolve in size, resulting in ever-increasing energy consumption and consequently operating costs. An important source of energy waste in cloud data centers lies in the inefficient usage of computing resources. A study on power usage at the scale of data center workloads [20] indicates idle power in servers is always above 50% of peak power. Therefore, keeping server underutilized is very inefficient from the energy consumption point of view. Energy efficient resource management strategies for cloud data centers are the type of algorithms trying to allocate the resources based on applications' resource demands in a way that increase utilization per server while fulfilling the application's constraints in terms of Service Level Agreements (SLAs).

Energy efficient resource management algorithms can be categorized in two general classes: The first one is the class of algorithms which mostly focus on dynamic re-consolidation of virtual machines by leveraging live migration in data center and the second one is the class of algorithms which focus on admission control and/or scheduling mechanisms. Both strategies raise several challenges such as optimality of the actions taken during the dynamic virtual machine consolidation process, the scalability issues and noisy neighbor problem which happens due to VMs interfering with each other in an overbooked data center. This thesis investigates the exploration of Machine Learning methods, Multi-Agent System paradigms and Nature-Inspired algorithms to tackle these challenges. In the scope of this thesis, we first address the problem of distributed dynamic virtual machine consolidation to tackle existing energy-performance trade-off in cloud data centers by proposing a Reinforcement Learning approach for overloading detection and virtual machine selection process as two main sub-problems of distributed dynamic virtual machine (VM) consolidation. We then propose a comprehensive solution to manage physical host nodes in dynamic virtual machine consolidation in a multi-agent system environment. In the next part of this thesis, we address the problem of scalability in dynamic virtual machine consolidation strategies and propose a novel fully distributed approach for dynamic virtual machine consolidation using a Gossip protocol over a P2P network of physical host nodes in data center. Finally, we address the problem of virtual machine interfering in overbooked data centers and propose a self-adaptive capacity controller by using a Reinforcement Learning approach to efficiently make a balance between utilization and the performance of the running applications inside a server.

Contents

List of Figures

List of Tables

List of Algorithms

Selected Publications

This thesis is based on work published in scientific conferences, workshops, and journals. For reasons of brevity, these core papers, which build the foundation of this thesis, are listed here once and will not be explicitly referenced again.

Refereed publications in journals

1. Brienza, S., Cebeci, S. E., **Masoumzadeh, S. S.**, Hlavacs, H., Ozkasap, O., Anastasi, G. (2016). A survey on energy efficiency in P2P systems: File distribution, content streaming, and epidemics. ACM Computing Surveys (CSUR), 48(3), 36. [13]

Refereed publications in conference proceedings and workshops

2. **Masoumzadeh, S. S.**, Hlavacs, H. (2013, April). An intelligent and adaptive threshold-based schema for energy and performance efficient dynamic VM consolidation. In European Conference on Energy Efficiency in Large Scale Distributed Systems (pp. 85-97). Springer, Berlin, Heidelberg. [54]

3. **Masoumzadeh, S. S.**, Hlavacs, H. (2013, October). Integrating VM selection criteria in distributed dynamic VM consolidation using Fuzzy Q-Learning. In Network and Service Management (CNSM), 2013 9th International Conference on (pp. 332-338). IEEE. [53]

4. **Masoumzadeh, S. S.**, Hlavacs, H. (2015, June). A cooperative multi agent learning approach to manage physical host nodes for dynamic consolidation of virtual machines. In Network Cloud Computing and Applications (NCCA), 2015 IEEE Fourth Symposium on (pp. 43-50). IEEE. [55]

5. **Masoumzadeh, S. S.**, Hlavacs, H. (2015, July). Dynamic virtual machine consolidation: A multi agent learning approach. In Autonomic Computing (ICAC), 2015 IEEE International Conference on (pp. 161-162). IEEE. [56]

6. **Masoumzadeh, S. S.**, Hlavacs, H. (2016, July). A Gossip-Based Dynamic Virtual Machine Consolidation Strategy for Large-Scale Cloud Data Centers. In Proceedings of the Third International Workshop on Adaptive Resource Management and Scheduling for Cloud Computing (pp. 28-34). ACM. [57]

7. **Masoumzadeh, S. S.**, Hlavacs, H., Tomas, L. (2016, September). A Self-Adaptive Performance-Aware Capacity Controller in Overbooked Datacenters. In Cloud and Autonomic Computing (ICCAC), 2016 International Conference on (pp. 12-23). IEEE. [58] **Received the Best Paper Award**

8. Tomas, L., **Masoumzadeh, S. S.**, Hlavacs, H. (2016, July). Self-Adaptive Capacity Controller: A Reinforcement Learning Approach. In Autonomic Computing (ICAC), 2016 IEEE International Conference on (pp. 233-234). IEEE. [85]

CHAPTER 1

Introduction

Cloud computing is becoming the prevalent way to deliver modern applications. A formal definition of cloud computing was given in [25] *as a large-scale distributed computing paradigm that is driven by economies of scale, in which a pool of abstracted, virtualized, dynamically-scalable, managed computing power, storage, platforms, and services are delivered on demand to external customers over the Internet.*

The worldwide market for public cloud services is projected to reach 204 billion dollars according to a Gartner report in 2016 [28]. The successful reason for cloud computing is that it allows users to get access to computational resources in a black box without having an expert knowledge required to set up and maintain all the underlying infrastructure. Therefore, the users can instead focus on their own domain of work resulting in enabling numerous research institutions, existing companies as well as emerging startups to innovate and address problems in all areas of human work.

The advantages of using cloud services have been increasingly encouraging customers to adopt them. The granularity of the offered service varies from low-level to high-level (see Figure 1.1) and are often categorized into three cloud delivery models:

- **Infrastructure-as-a-Service (IaaS)**. It allows customers to run and deploy arbitrary software applications on a set of infrastructure, including fundamental computing resources such as storage, and virtual machine (VM). In this model, the cloud customer has full control on both the software application and the infrastructure hosting that application. A notable example of IaaS offering is Amazon Elastic Compute Cloud (EC2[1]).

- **Platform-as-a-Service (PaaS)**. It offers customers to use a hosting environment, including the programming languages, libraries, and tools, for their applications. While the customers have the capability of controlling their application, they have

[1]Amazon EC2: `http://aws.amazon.com/ec2`

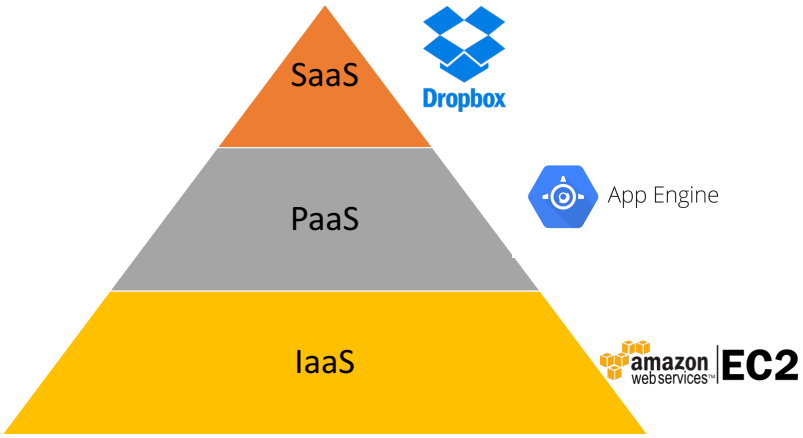


Figure 1.1: Cloud delivery models

limited control of the operating system and the infrastructure. A notable example of PaaS offering is Google App Engine[2].

- **Software-as-a-Service (SaaS)**. It offers software applications running on top of cloud infrastructures. The cloud customers can only use the application and they are fully restricted to control the infrastructure and the application. Notable examples of this model are storage services such as Dropbox[3], or streaming services such as Netflix[4].

In this thesis we focus on infrastructure as a service (IaaS). As reported at Gartner [27], infrastructure as a service (IaaS) has been the fastest growing segment of the cloud services market for several years. To describe the IaaS public cloud model in details, it is very useful to look at the individual components which are necessary for such a service. The service provider, i.e. the cloud provider owns one or more data centers (warehouse-like buildings equipped for housing, powering and cooling) including thousands of virtualized physical machines. By using virtualisation technology, each physical machine (PM) can host multiple virtual machines which are isolated from each other and having access to a certain amount of physical resources (CPU cores, Memory, storage, network bandwidth). The virtual machines can be migrated from one physical machine to another one through the network. In fact, the virtual machine (VM) is IaaS products which is offered as a computational resource to customers for usage over the Internet.

The high level goal of the research described in this thesis is to investigate the exploration of machine learning methods, multi-agent system paradigms and nature-inspired algorithms in tackling different challenges raised in energy efficient resource management techniques in IaaS cloud data centers including; 1) dynamic virtual machine consolidation

[2]Google App Engine: `http://developers.google.com/appengine`
[3]Dropbox: `http://www.dropbox.com`
[4]Netflix: `http://www.netflix.com`

techniques, where a central virtual machine management entity leveraging a virtual machine migration strategy to dynamically re-allocate the resources towards keeping a balance between energy consumption of the data centre (physical machines) and the performance of the applications running in the cloud; 2) resource overbooking techniques, where an admission controller/scheduler allocating VMs based on real utilization ratios instead of nominal requested capacity, i.e., provisioning more VMs that physical available resources.

To introduce the work presented in this thesis, in the following section we describe the problems motivating our research. In Section 1.2 we list the individual research questions we address. In Section 1.3 we summarily present the contributions of our thesis corresponding to the research questions. In Section 1.4 we give a general methodological framework used throughout the thesis. Section 1.5 presents the organization of the remainder of the thesis.

1.1 Problem Statement and Research Goals

With the rise in demand of computing power by costumers cloud data centers evolve in size, resulting in ever-increasing energy consumption and consequently operating costs. Apart from it, the way energy is generated for cloud data centers influences our environment either directly by the carbon footprint or indirectly by the waste created by power plants. Therefore, cloud data centers should be managed in an energy efficient manner, and the main sources for energy waste must be identified. An important waste of energy is due to the inefficient usage of computing resources. A study on power usage at the scale of data center workloads [1] shows the power usage dynamics of servers are very narrow. The same study indicates idle power in servers is always above 50% of peak power. Therefore, keeping server underutilized is very inefficient from the energy consumption point of view. There are two main strategies to manage resources in an energy efficient manner; 1) Dynamic Virtual Machine Consolidation strategies and 2) Resource Overbooking techniques.

1.1.1 Dynamic Virtual Machine Consolidation

In this strategies, the admission control unit allocating the VMs based on their nominal requested capacities, then the VMs are reallocated dynamically based on their real utilization ratio to keep the balance between resource utilization and performance degradation of the application running on the VMs as expressed by Service Level Agreements (SLAs). Most solutions found in the literature typically focus on centralized approaches, with a single management node making allocation decisions periodically [23] [35] [24]. These approaches suffer from poor scalability, as the management node may become a performance bottleneck if the number of physical and virtual machines grows. There are few researches [65] [31] [7] proposing VM consolidation based on dynamic decision makings (e.g, when a host can be considered as over-utilized, which virtual machine must be selected to migrate, which hosts must be selected to place migrated virtual machines,

when a host can be considered as under-utilized). These approaches have two major advantages compared with the periodical VM consolidation algorithms: (1) splitting the problem into dynamic decision making tasks simplifies the analytic treatment of each task; and (2) it can be implemented in a distributed manner by considering the physical machines as management entities. However, the most significant challenge here is how to make these decisions optimally to achieve the objective which is keeping the right balance between energy saving and performance degradation in data center.

1.1.2 Resource Overbooking

Overbooking is an strategy in which there is no live migration and the admission control unit allocating VMs based on real utilization ratios instead of nominal requested capacity, i.e., provisioning more VMs than physical available resources. However, overbooking techniques always expose the infrastructure to a risk of resource congestion upon unexpected situations and consequently to SLA violations. On top of this problem, not all the VMs require the same Quality of Service (QoS), meaning they do not need to maintain the same throughput or response times over time. Furthermore, not all the VMs are equally affected by this interference [84]. For instance, a deadline constrained application (e.g., a computationally intensive task) does not need to maintain an average performance all the time as long as it completes the task in time. By contrast, an interactive application may require a certain minimum performance all the time, as given for instance by e-commerce systems. One of the best strategy to treat VMs with different QoS requirements differently is to include QoS differentiation (high and low QoS levels) has been introduce in [88]. This QoS differentiation creates different isolation levels between VMs by pinning them to specific cores inside the servers (note core pinning is a mechanism provided by KVM [37]), thus limiting the impact they may have on others. However, another challenge rises here. Such QoS classification impacts the overall utilization when high QoS VMs are overprovisioned. Therefore, to deal with this problem a capacity controller is needed which would be able to share extra physical cores allocated to the high QoS applications dynamically with low QoS apllications while keep the performance isolation of the high QoS applications.

1.2 Research Questions

After giving the detailed problem description in the previous section as a context for the issues that motivate our research, we describe the open research questions that we addressed in this thesis. The first two research questions focus on existed trade-off between energy consumption and performance in a cloud data center, while the next consider the existed trade-off between server utilization and performance of its running applications.

RESEARCH QUESTION I. *How can we make a dynamic virtual machine consolidation strategy in IaaS cloud data centers to keep an optimal balance between energy and*

performance?

Dynamic Virtual machine consolidation is one of the most proposed strategies by researchers which can be employed to make the energy consumption efficient in cloud data centers. These strategies refer to a number of resource management algorithms aiming at finding the right balance between energy consumption and SLA violations by using live migration techniques in virtualized cloud data centers. The dynamic VM consolidation problem is a dynamic multi objective optimization problem. The first objective is consuming energy and the second one is performance of the applications running on the virtual machines. Amongst dynamic virtual machine consolidation strategies, Distributed Dynamic VM Consolidation (DDVMC) [8][5] has gained a lot attraction which proposing a distributed algorithm in order to increase scalability by enabling physical host nodes as management entities in a hierarchical system architecture. Each physical host node in this architecture contributes to the dynamic virtual machine consolidation procedure by managing itself, avoiding performance degradation in an overloading situation as well as avoiding inefficient energy consumption in an underloading one. The former can be done by migrating one or more virtual machines away from itself to other physical host nodes, the latter can be done by migrating all virtual machines away and going into a sleeping mode. Such self-management algorithm involves three dynamic decision making tasks for each physical host node to tackle the dynamism of cloud environment raised by variable and unexpected applications workload: 1) when must a host be considered as overloaded, 2) which virtual machine must be selected to migrate away from an overloaded host, and 3) when must a host be considered as underloaded. In addition, there is a global manager in this architecture is responsible for optimizing the placement of virtual machines that must be migrated away from overloaded and underloaded physical host nodes. This approach has two major advantages compared with traditional VM consolidation algorithms (1) splitting the problem simplifies the analytic treatment of decision making tasks; and (2) it can be more scalable as executing the overload detection and VM selection algorithms can be done on compute hosts, and the VM placement algorithm on replicated controller hosts. The open problem here is to find the best decision maker for each sub-problem which lead to a global optima in terms of energy end performance efficiency in cloud data center.

RESEARCH QUESTION II. *How can we make an energy efficient dynamic virtual machine consolidation to be scalable and fault tolerant?*

The problem of Distributed Dynamic VM Consolidation (DDVMC) proposes a hierarchical architecture, in which there is still a central entity which is responsible for underloading detection and placement of the virtual machines which need to be migrated from underloaded and overloaded host nodes. The system model still suffers from a single point of failure. Moreover, the central manager could still turn into a performance bottleneck if the number of physical host nodes dramatically grows in the respective data center. The open problem is how to design a fully distributed dynamic virtual machine consolidation which provide a performance very close to or even better than a hierarchical approach.

RESEARCH QUESTION III. *How can we dynamically reallocate resources among virtual machines in overbooked data centers while keeping performance of the applications based on the agreed service level?*

This low resource utilization is a big concern for cloud data center providers as data centers consume lot of energy and are being used in a rather inefficient way-energy consumption does not decrease linearly with resource usage. One way cloud providers can mitigate these resource utilization problems is by overbooking. However, this utilization increment as a consequence of overbooking also increases the possibilities of having VMs interfering with each other, as more VMs are competing for the same resources which may lead to delays when accessing them. This is a well known effect known as noisy neighbor problem [89], which may heavily impact application performance. On top of this problem, not all the VMs require the same Quality of Service (QoS), meaning they do not need to maintain the same throughput or response times over time. In order to treat VMs with different QoS requirements differently, [83] proposes a mechanism to creates different isolation levels between VMs by pinning them to specific cores inside the servers (note core pinning is a mechanism provided by KVM [37]), thus limiting the impact they may have on others. This QoS classification however impacts the overall utilization when high QoS VMs are overprovisioned. The open problem is how to design a capacity controller to dynamically shares extra physical core allocated to a high QoS VM with low QoS VMs to improve overbooking while keeping the performance of the applications running on the VMs.

1.3 Scientific Contributions

Based on the research questions identified in the previous section, we now list the individual contributions presented in this thesis. Each of the contributions was previously published as a peer-reviewed paper that is referenced with the corresponding contribution in this section.

SCIENTIFIC CONTRIBUTION I. *A self-adaptive threshold-based strategy to detect overloaded hosts for the problem of dynamic VM consolidation.*

In this thesis, we introduced an intelligent and adaptive threshold based schema for detecting overloaded hosts in distributed dynamic VM consolidation (DDVMC) procedure by using a machine learning technique. Our proposed algorithm as a centralized agent in a hierarchical system model learns how to tune a utilization threshold for each individual host node towards energy-performance tradeoff optimization in cloud data center. The tuned threshold is used in the process of host overloading detection in distributed dynamic VM consolidation procedure. This contribution addresses research question (RQ) 1 and target the first sub-problem in distributed dynamic VM consolidation strategy where overloaded hosts have to be detected proactively in an optimal manner to avoid perfor-

mance degradation of the application running on the VMs. We validate our approach with the CloudSim toolkit [14] using real world PlanetLab workload [68]. Experimental results show that using our method yields far better result w.r.t the energy-performance trade-off in cloud data centers in comparison to state of the art algorithms. The results were previously published in [54] and is presented in Chapter 4 of this thesis.

SCIENTIFIC CONTRIBUTION II. *A self-adaptive and performance-aware VM selection schema for the problem of dynamic VM consolidation.*

In this thesis, we propose an intelligent and self-adaptive VM selection schema in the migration process of distributed dynamic VM consolidation. Our proposed algorithm by using a machine learning method in a distributed fashion, learns how to choose an appropriate VM selection strategy dynamically from a set of possible strategies to achieve better results in terms of energy and performance efficiency than each of the individual strategies achieves when used alone. This contribution addresses research question (RQ) 1 and target the second sub-problem in distributed dynamic VM consolidation strategy where optimal selection of VMs in overloaded hosts can effect on the global performance of the dynamic VM consolidation process. We validate our approach with the CloudSim toolkit [14] using real world PlanetLab workload [68]. Experimental results show that an intelligent integration of multiple criteria to select VMs for migration benefits from all advantages and possible synergetic contributions of them in long-term learning. The results were previously published in [53] and is presented in Chapter 4 of this thesis.

SCIENTIFIC CONTRIBUTION III. *A distributed and self-adaptive management of physical host nodes for dynamic consolidation of virtual machines.*

In this thesis, we propose a comprehensive solution to manage physical host nodes for distributed dynamic virtual machine consolidation by combining (SC) 1 and (SC) 2 in a multi agent system environment. In fact, we employ a cooperative multi-agent learning paradigm [67] to solve the problem of physical host node management of dynamic VM consolidation in a distributed manner. In our system model, a data center is considered as a multi-agent learning environment in which each learning agent represents a decision-making engine inside each physical host node which is responsible for making decision about the threshold value in overloading detection process and responsible for selecting an appropriate VM selection strategy in migration process. In this environment, all the agents are capable of cooperating with each other, such that the global goal of the system (energy-performance tradeoff optimization) can be achieved by locally acting. On the other hand, the cooperation between learning agents can provide a further benefit in our system by speeding up the learning process and consequently increasing the decision-making quality. We validate our approach with the CloudSim toolkit [14] using real world PlanetLab workload [68]. This contribution addresses research question (RQ) 1 and target the first, second and third sub-problem together. Experimental results show

that using our method yields far better result w.r.t the energy-performance trade-off in cloud data centers in comparison to state of the art algorithms. The results were previously published in [55] [56] and is presented in Chapter 4 of this thesis.

SCIENTIFIC CONTRIBUTION IV. *A fully distributed dynamic virtual machine consolidation in large-scale cloud data centers.*

In this thesis, we propose a fully distributed dynamic virtual machine consolidation strategy by using a bio inspired algorithm on top of an unstructured P2P network of physical machines to be used in huge cloud data centers. In our proposed system model, dynamic virtual machine consolidation can be used by physical host nodes only within the scope of their neighbourhoods, allowing the system to be scaled by increasing the number of physical host nodes and virtual machines inside the data center, without any centralized global system knowledge. In our system model the physical host nodes can leave the data center (e.g. because of failure) or be added to the data center (e.g. in order to increase resources) without having any effect on the global performance of the virtual machine consolidation process. Our results illustrate that our proposed strategy can achieve global efficiency in terms of energy and quality of service very close to the optimal centralized approach if the neighbourhood size is large enough. In order to evaluate the performance of our proposed approach, we carried out simulation runs using our cloud data center simulator that we have developed in-house over PeerSim simulator tool [62]. This contribution addresses research question (RQ) 2 and the results were previously published in [57] and is presented in Chapter 5 of this thesis.

SCIENTIFIC CONTRIBUTION V. *A self-adaptive performance-aware capacity controller to adapt the isolation levels between high and low QoS VMs in overbooked data centers.*

In this thesis, we propose a self-adaptive capacity controller by using a machine learning technique in overbooked datacentre where a KVM core pining strategy make different isolation level in terms of number of dedicated CPU cores for high QoS and Low QoS applications running on virtual machines inside a physical server. As these isolation levels may lead to low utilization ratios if high QoS VMs are overprovisioned, our machine learning based capacity controller is responsible for dynamically deciding how much capacity each high QoS VM can share with low QoS VMs, i.e., how much the isolation level between them is reduced. This contribution addresses research question (RQ) 3 and the results were previously published in [58] [85] and is presented in Chapter 6 of this thesis. Our evaluation based on real cloud applications and workloads demonstrates that the efficient, adaptive mapping between VMs and physical resources reduces the interference between VMs, enabling the possibility of co-locating more VMs, increases overall utilization, and ensures the performance of critical applications while providing more resources to the low QoS applications.

1.4 Methodology

After an overview of the thesis contributions, in this section we describe the general methodology applied in contributions to evaluate their effectiveness towards addressing the corresponding research questions.

Scientific Contribution I and II

1. Conduct a research on dynamic VM consolidation strategies, comparing them and obtaining insights into designing such strategies and algorithms.
2. Conduct a research on machine learning approaches to find a proper method which would be effective in the process of online decision making of dynamic VM consolidation.
3. Develop a dynamic virtual machine consolidation by using a machine learning approach based on the conducted research in steps 1 and 2.
4. Evaluate the proposed algorithm through discrete-event simulation using the CloudSim simulation toolkit extended to support power and energy-aware simulations.

Scientific Contribution III

5. Conduct a research on cooperative multi agent learning systems to develop a multi-agent based system model for the problem of dynamic VM consolidation.
6. Develop a multi agent learning based distributed dynamic VM consolidation based on the conducted research in step 5.
7. Evaluate the proposed algorithm through discrete-event simulation using the CloudSim simulation toolkit extended to support power and energy-aware simulations.

Scientific Contribution IV

8. Conduct a research on P2P systems and nature inspired algorithms to develop a fully distributed dynamic VM consolidation.
9. Develop a fully distributed dynamic VM consolidation based on the conducted research in step 1, 5 and 7.
10. Evaluate the proposed algorithm through discrete-event simulation using a cloud data center simulator that we developed in-house over PeerSim simulator tools.

Scientific Contribution V

11. Conduct a research on overbooking management strategies and capacity controllers in overbooked datacenter, obtaining insight into designing such controller.

12. Develop a self-adaptive capacity controller based on the conducted research in steps 2 and 11.

13. Evaluate the proposed algorithm on a real infrastructure.

1.5 Thesis Organisation

According to the research questions and the scientific contributions presented in this chapter, the remaining of the thesis is organized as follows. The five scientific contributions outlined in Section 1.3 are grouped into three main chapters (Chapters 4 to 6).

- Chapter 2 discusses the cloud computing concepts as well as the primary well-established theories, techniques, and models from other domains used in the scope of this thesis.
- Chapter 3 systematically examines the various research branches grouping work related to ours. We cover the state-of-the-art methods in each of these branches and highlight some of the open challenges that we address in our work.
- Chapter 4 introduces distributed dynamic VM consolidation strategies that leverage machine learning and multi-agent system approaches towards energy efficiency in cloud environment. The goal of the presented strategies is to make an optimal balance dynamically between energy and performance of the application running on the VMs in cloud data centers.
- Chapter 5 presents a fully distributed dynamic VM consolidation on top of a P2P network of physical machines by leveraging a bio inspired algorithm. The main focus of the chapter is to propose a dynamic VM consolidation strategy which would be scalable and fault tolerant in very large scale data centers.
- Chapter 6 presents a self adaptive capacity controller for overbooked data center using a machine learning approach which is able to efficiently distribute the available capacity between the different VMs. This in turn enables the possibility of accepting a larger amount of applications, thus increasing overall utilization, without neither hurting low nor high quality applications' performance.
- Chapter 7 concludes the thesis while discussing the limitations of the presented contributions, and providing an outlook into possible future directions.

CHAPTER 2

Background

This chapter provides background information about well-established concepts and technologies which form the basis of the work carried out in the scope of this thesis. We first illustrate and clearly define the cloud computing concepts which are used in our research, and then introduce the models, theories, and concepts that are utilized in our solutions from other scientific domains.

2.1 Cloud Computing Concepts

Cloud computing is an emerging paradigm that is a combination of different technologies, such as grid computing, service-oriented architecture (SOA), Internet, and the virtualization technology. One of the key technologies that enable cloud computing to provide on-demand computing models is virtualization [26, 46]. The server virtualization technology can facilitate cloud resource management and would be able to provide efficient resource utilization by sharing resources in the form of virtual machines [93](see Figure 2.1). A virtual machine is typically a basic building block owning a separate operating system (OS), which can be easily started, stopped, hibernated, or migrated [42]. The virtualization technology also enables adding or releasing the resources such as CPU, memory, storage, and communication bandwidth from one VM to another, on demand at runtime [46]. The software that provides the virtualization is called a Virtual Machine Monitor (VMM) or hypervisor. A VMM virtualizes all the resources of physical machines, thereby supporting the execution of multiple VMs [78] in a single physical host. In the next subsection we present the state of the art server virtualization techniques.

2.1.1 Virtualization Techniques

Server virtualization techniques can divided into three categories [21]: full virtualization, paravirtualization, and OS-level virtualization [15] [95] [51]. Before going to the details

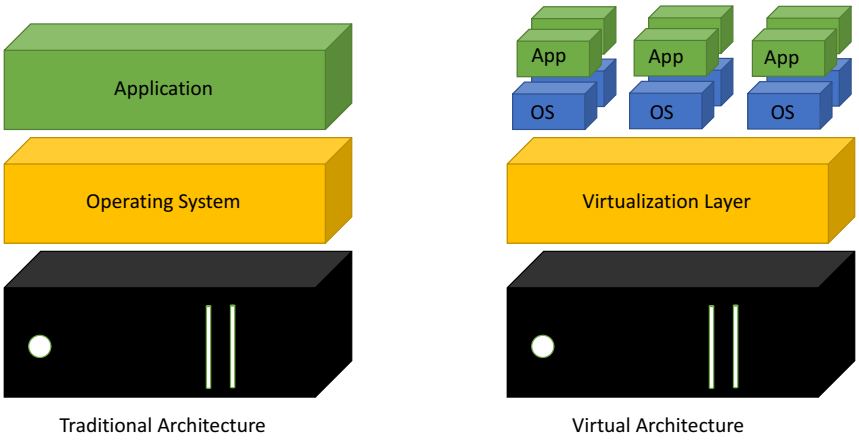


Figure 2.1: Traditional architecture vs. virtual architecture

of each category we need to briefly introduce the concept of privilege levels also known as protection rings [77] which is implemented on all the modern x86 CPUs.

Privilege levels is one of fundamental concept has been integrated into the x86 CPUs to enable security in modern operating systems. Privilege levels particularly allow the OS to avoid any uncontrolled access from users and processes to the shared physical resources such as CPU, memory, and I/O. For instance, a user application/process running on the OS should not be able to has a direct access to the disk as it can harm the system stability by initiating malicious activities such as deleting other users file or manipulating the hardware. In fact, it is the OS responsibility to grant/deny access to physical hardware and enforce the users' process. Figure 2.2 illustrates the privilege levels of the x86 architecture.

As the Figure shows, there are four privilege levels which are organized in increasing order form 0 to 3. OS (most of the modern OS such as Linux and Windows) running on the highest privilege (Level 0) and has a fully access to the physical hardware. In contrast, user processes typically running on the lowest privilege level (Level 3). The intermediate privilege levels (Level 1 and Level 2) remains unused. When a Virtual Machine Monitor (VMM) is used to control the physical hardware (i.e. CPU, memory, I/O) it must operates in the highest privilege level (i.e. Level 0) to be able to : 1) support physical hardware multiplexing between the guest OS's; 2) provide guest OS isolation and secure VMM. In this situation, the guest OS's kernel does not have any permission in the highest privilege level (i.e. Level 0). On the other hand, OS kernels have not been designed to operate at other privilege levels than zero. One solution is to let the guest OS kernels operates in a lower privilege level (e.g. Level 1) and let the VMM intercepts and handles all the instructions requiring higher privilege level. However, this is not feasible in traditional x86 CPU architecture as it has not been designed for virtualization. To fix this issue, there are couple of alternative approaches which require either to use emulation or complex mechanisms inside the VMM (e.g. binary translation [79]). A clean

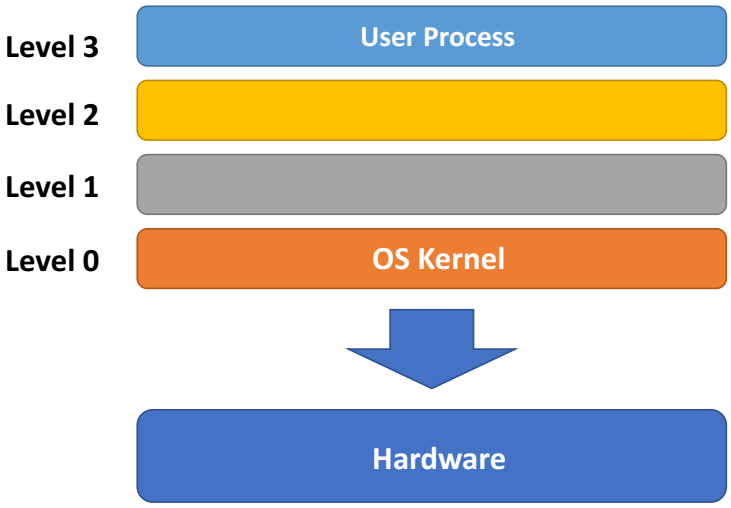


Figure 2.2: x86 architecture privilege levels

solution for the problem of the privilege level in virtualization has been introduced by both major vendors Intel and AMD in [90] and [1]. They proposed an extension to their CPU privilege hierarchy by adding a new level, called -1. It can be used for operating VMM while allowing the guest OS kernels to continue running in Level 0.

Full virtualization

In full virtualization, a guest OS can be run on top of the existing host OS without modifying the host OS or guest OS's kernels. Full virtualization is achieved by using two different approaches: 1) binary translation and 2) hardware acceleration. In binary translation, the main objective is to leverage the host OS I/O device support while providing close to native CPU performance by executing as many CPU instructions as possible on bare hardware. Figure 2.3 shows binary translation architecture. After installation, the virtualization solution first loads a driver into the host OS kernel. This driver is required by the user space (e.g. level 3) application to hand control over the physical hardware whenever it is needed. For instance, when the user space application is used to start a guest OS, it contacts the installed driver to reconfigure the host OS in order to start the guest OS kernel in privilege level 1. In addition, the driver hooks a Virtual Machine Monitor (VMM) beneath the host OS kernel to trap privileged instructions issued by the guest OS's. Finally, the VMM integrates binary translation to detect non-virtualizable instructions at run-time and replaces them with VM-safe code. This is achieved by examining the guest OS kernel binary stream using binary translation.

The main advantage of using binary translation technique in full virtualization is that the host and guest OS kernel do not need to be modified in this process. In fact, the guest OS kernel drivers simply use the driver interfaces of the original host OS

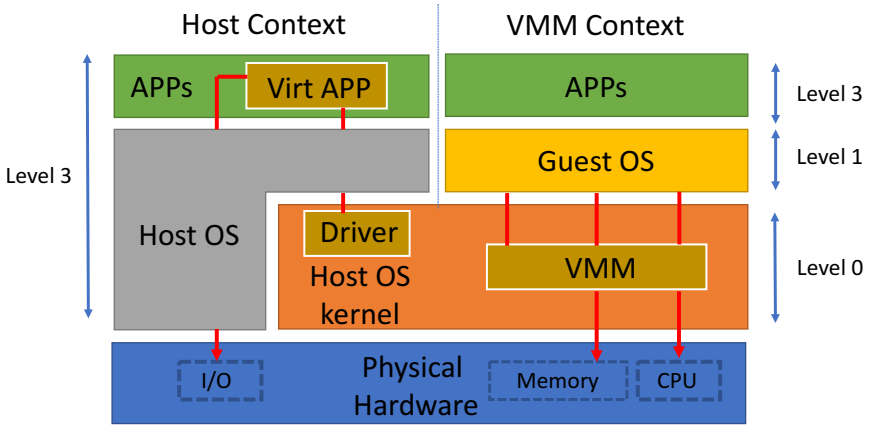


Figure 2.3: Binary translation technique architecture

kernel. However, this flexibility imposes a significant cost of decreased performance due to performing binary translation and emulating privileged CPU instructions. Parallels Workstation/Desktop[1], VMware Workstation/Fusion/Player[2], and VirtualBox[3] are some of the popular full virtualization solutions which are using binary translation technique.

The second approach for full virtualization is hardware acceleration. Intel and AMD introduced a new technology called Intel VT-x (resp. AMD-V) which provided a full utilization without the binary translation overheads. For example, Intel-VT-x introduces two different CPU modes: root and non-root. The root mode provides a privilege level mechanism as same as well-known x86 privilege while the non-root mode introduces a new structure called Virtual Machine Control Structure (VMCS). It allows to provide fine grained control over the CPU instructions by adding a new level to the previous levels, referred to as -1. In this technique the guest OS kernel operates in the non-root mode on the privilege level 0 and the VMM runs in the root mode. Figure 2.4 shows an overview of hardware acceleration technique. This techniques allows, the guest OS kernel operates without any need for binary translation for non-virtulizable instructions and remains unmodified. The context switches between the root and non-root modes called VMEntry (VMExit in AMD). It happens when the guest OS attempts to run a non-privileged instruction. In this situation, the information describing the root of the problem is saved in the VMCS structure. This information is used by the VMM in the root mode to resolve the issue. Some of the modern VMMs such as Kernel-based Virtual Machine (KVM)[4], Microsoft Hyper-V[5], VMware ESX/ESXi[6], Xen Hardware Virtual Machine[7] use this technique.

[1]Parallels Workstation/Desktop: `http://www.parallels.com`
[2]VMware Workstation/Fusion/Player: `https://www.vmware.com`
[3]VirtualBox: `https://www.virtualbox.org`
[4]KVM: `https://www.linux-kvm.org`
[5]Hyper-V: `https://www.microsoft.com/en-gb/cloud-platform`
[6]VMware ESX/ESXi: `https://www.vmware.com/`
[7]Xen: `https://www.xenproject.org/`

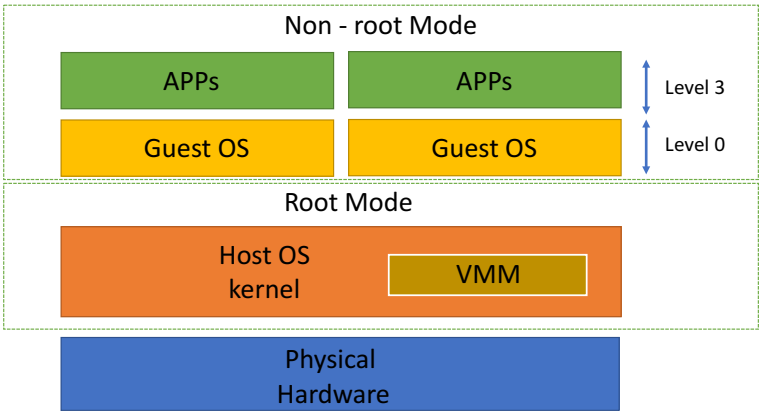


Figure 2.4: Hardware acceleration technique architecture

Paravirtualization

Another type of virtualization is Paravirtualization (PV) which involves operating a lightweight kernel on top of the bare hardware as VMM. In comparison with the full virtualization approaches which do not require any guest OS kernel modifications, paravirtualization modifies guest OS's and allows them to naively interact with the bare hardware. In fact there is no host OS in this architecture which consequently increases the guest OS performance. The most popular paravirtualization solution is called Xen Paravirtualization (PV)[8]. Figure 2.5 shows a high-level architecture overview of PV. As the VMM is a lightweight kernel, it is not able to provide any complex management decisions such as admission control, inter-VM CPU scheduling decisions. However, it exports a low-level control interface which can be employed to enforce such control decisions. During the Virtual Machine Monitor (VMM) process, a control VM also known as Dom0 (Domain 0) is started to operate as the first guest OS in the privilege level 1. It is responsible for providing a management layer and also performing other management decisions such as controlling the guest OS life-cycle (e.g. boot, reboot, shutdown) and physical memory allocations. As the Figure 2.5 shows, the VMM occupies the highest privilege, therefore the guest OS's need to be further modified for any operation. To this end, Xen moves the guest OS's to privilege level 1 and modify them to make them enable to delegate all their privileged (including non-virtualizable) instructions to the VMM using Xen specific software interrupts, also known as hyper-calls. The VMM, on behalf of the guests OS's, traps the hypercalls and executes the privileged instructions either by translating them into the native hardware instructions or using emulation.

OS-Level Virtualization

OS-level virtualization enforces a virtualization technique by creating containers inside the same OS (see Figure 2.6). They are managed and isolated form each other by a

[8]Xen: `https://www.xenproject.org/`

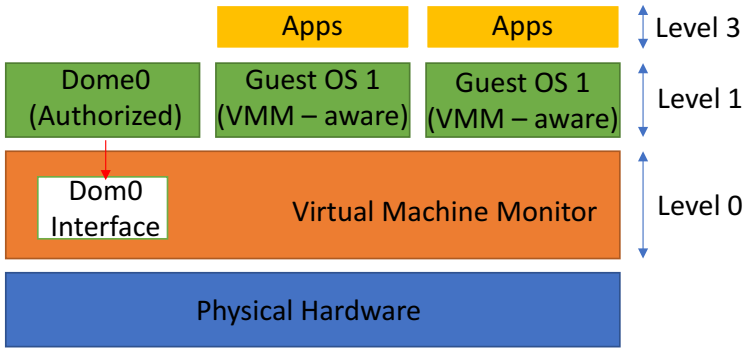


Figure 2.5: Paravirtualization technique architecture

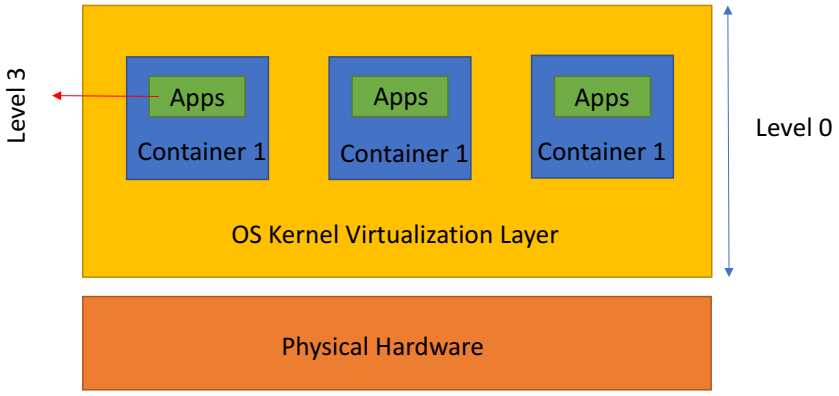


Figure 2.6: OS-level technique architecture

virtualization layer within the host OS kernel. They have their own resources such as root file system, applications, users, firewall and networking settings. Containers provide a near native performance to their applications. In addition, they can be migrated in the same manner as traditional VMs. The most significant drawback of OS-level virtualization is that the containers are limited to a single host OS kernel. The most popular OS-level virtualization solutions are OpenVZ[9] and Linux Containers[10].

2.1.2 Virtual Machine Live Migration

All of the virtualization techniques provide Virtual Machine (VM) control operations such as start, reboot, stop and suspend. However, some of them provide a specific control feature known as live migration. Live migration allows Virtual Machines (VMs) to move seamlessly between Physical Machines (PMs) across either a Local Area Network (LAN) or Wide Area Network (WAN). Seamless live migration refers to a VM movement which

[9]OpenVZ: `https://openvz.org`
[10]Linux Containers: `https://linuxcontainers.org/`

is not noticeable or barely noticeable (i.e. in the order of milliseconds) to the VMs users. A live migration can be triggered manually by a system administrator or a cloud management system. Based on some high-level objectives, a live migration involves some decision making tasks including determining which VMs and to which PMs they have to be migrated. For example, in case that the decision maker is a cloud system management, an objective would be consolidating multiple VMs on a fewer number of PMs for energy saving. In order to have a seamless VM live migration, the occupied resources (i.e. CPU cores, memory, storage and networking connections) by a virtual machine must be transparently move from the source Physical Machine (PM) to the destination PM. In general, VM live migration can be categorized into three approaches: 1) pre-copy, 2) post-copy, and 3) hybrid [34] [16]. In this section we assume there is a shared storage available to avoid VM disk migration (see Figure 2.7) and we just focus on the VM memory migration techniques.

Pre-copy Live Migration

In the pre-copy live migration approach, the memory pages of a running VM (which is supposed to be migrated) are iteratively copied by the Virtual Machine Monitor (VMM) over the network from the source to the destination PM. There are two phases in this process: 1) Warm-up phase, in which all the VM's memory pages are transferred to the destination PM, then the VMM examines the VMs memory to detect pages which were modified during the previous round. If the modified pages are detected ('dirty' pages), they will be re-copied until no memory modifications can be observed. 2) Stop-and-copy phase, in which the VM is stopped by the VMM on the source PM, its CPU state is transferred to the destination PM and the VM is resumed on the destination PM. The time between stopping the VM on the source PM and resuming it on destination PM is called "down-time".

Post-copy Live Migration

In the post-copy live migration approach [34] the VM is first suspended on the source PM and its CPU state is transferred to the destination PM, then the VM is resumed on the destination PM while memory pages still reside on the source PM. In this situation, page faults are generated on the destination side once the VM attempts to access any memory pages which are not yet available in the destination PM. Once a page fault is occurred, the VMM instructs the source PM to send the page involved in the page fault. The benefit of using post-copy approach is to reduce the network overhead, however, its major drawback is that the VM can experience a serious performance degradation during the on-demand memory page transfer process.

Hybrid Live Migration

Hybrid live migration approach is a combination of pre-copy and post-copy approaches. The idea of hybrid live migration is to start with VM pre-copy live migration and then

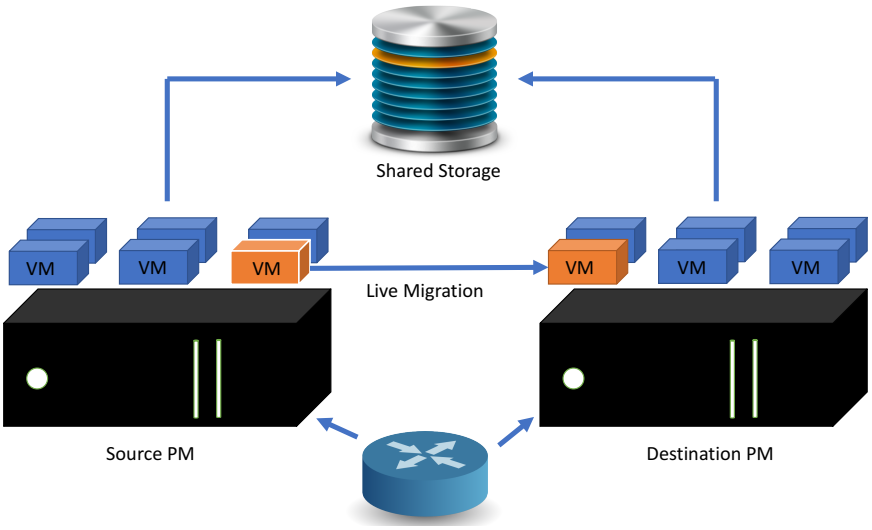


Figure 2.7: Live migration with shared storage

switch to post-copy approach after the first memory pre-copy iteration. In the process of hybrid live migration, all the VM memory is copied from source PM to the destination PM while the VM is runing inside the source PM, then the VM is suspended and its CPU state is transfered to the destination PM. The VM is resumed on the destination sided and the post-copy mechanism is started as explained above.

In this thesis, we used the concept of pre-copy live migration in our simulation and also leverage pre-copy live migration mechanism of the KVM hypervisor where the experiments have been implemented in a real infrastructure.

2.1.3 Cloud Deployment Models

The deployment models in clouds are the different ways of using cloud infrastructure and services. The most common categorization is as follows [76, 50]:

1. **Public cloud**. In a public cloud, services are available publicly to all customers through the Internet. The public cloud providers have well-defined pricing models and accounting mechanisms for their offered services.
2. **Private cloud**. This deployment model is used by organizations for deploying their private software applications in their in-house data centers. Just members of the organization will be granted to have access to a private cloud.
3. **Hybrid cloud**. A hybrid cloud deployment model represents a combination of at least two distinct clouds that connects two or more clouds in terms of their deployment models (e.g., public and private) [59]. Often a hybrid cloud model is used in the case of workload bursting where the private cloud is not sufficient for providing the service and it is provided by a public cloud [70].

In the case of a hybrid cloud model, if the underlying clouds, are restricted to only public clouds, it is called multiple clouds [17]. In other words, in a multiple cloud model, a cloud customer, e.g., an application owner, deploys the application simultaneously on multiple cloud providers. In general, two types of delivery models exist for multiple clouds [69]: *federated-cloud* and *multi-cloud*. These models differ in the degree of collaborations between the involved cloud providers and the way the cloud customer interacts with them:

- **Federated-cloud model**. In this model, the cloud providers are in agreement with each other to provide a federation in order to enhance the services offered to their customers. In this model the usage of multiple cloud services is transparent from the cloud customer viewpoint.
- **Multi-cloud model**. A multi-cloud model represents the usage of multiple, independent clouds by a cloud customer. This model does not imply interconnection and sharing among the clouds [70], i.e., there is no need for an agreement among them. Furthermore, in this model, the cloud customer is aware of using services from multiple clouds, and usually a third party, e.g., a *multi-cloud middleware*, is responsible for communicating among the providers involved.

2.1.4 Energy Consumption in Cloud Computing

Energy consumption by computing equipment raises two significant concerns; 1) environmental concerns and 2) system performance concerns. A recent study on power consumption of server farms in 2005 [45] showed that the electricity used by servers worldwide (including their auxiliary equipment and associated cooling systems) cost 7.2 billion dollars. The study also indicates that the energy consumption became doubled during 5 years from 2000.

There are also environmental issues with the generation of electricity. As discussed in [44], the number of transistors integrated into today's processors reaches nearly 1 billion. If the transistor density continues to growth, the heat (per cm2) produced by future processors would exceed that of the surface of the Sun. It is worth mentioning, the scope of energy-efficient design is not limited to main computing components such as processors, storage devices, and visualization facilities, but can expand into the other computing facilities including auxiliary equipment and cooling system. Some recent efforts in hardware technologies such as low power processors, solid state drives and energy-efficient monitors tried to alleviate the energy consumption. Energy-efficient resource management techniques are other solutions which are not competing with the hardware solutions but also should be seen as complementary ones. These techniques were applied to mobile devices which are usually battery powered to improve their life time [13], however due to continues growth of cloud data centers and consequently their power and energy consumption the focus of energy management techniques has been switched to such systems as well.

According to an investigation by Intel Labs [60] on the sources of power consumption in a single server, CPU has the highest fraction in the power consumption which is

followed by the memory and the losses due to the power supply inefficiency. The data provided by [60] also show the CPU is no longer dominates power consumption by a single server, thanks to power-saving techniques (e.g., DVFS) that enable active low-power modes and also the adoption of multi-core architectures. Apart from power inefficiency in hardware, an important waste of energy in the scale of data center is due to the inefficient usage of computing resources. A study on power usage at the scale of data center workloads [20] shows the power usage dynamics of servers are very narrow. The same study indicates idle power in servers is always above 50% of peak power. Therefore, keeping servers underutilized is very inefficient from energy consumption point of view. In a cloud data center, virtualization can provide server consolidation by allowing multiple Virtual Machine (VM) instances to run on a single physical server, resulting in fewer running physical servers with higher per-server utilization. However, server consolidation enabled by virtualization introduces a new problem to the cloud data center. Since the amount of resources occupied by VMs inside a physical server might change due to the dynamic and unexpected behavior of applications running on the virtual machines, resource underutilization or overutilization might occur inside a physical server, causing either inefficient energy consumption or undesired performance degradation.

2.1.5 Energy Efficient Resource Management

The reduction in energy consumption in data centers can be achieved by improving the utilization of the servers. Based on this definition, the energy efficient resource management algorithms for cloud data center can be categorized in two general classes: The first one is the class of the algorithms which mostly focus on dynamic re-consolidation of virtual machines by leveraging live migration in data center and the second one is the class of algorithms which focus on admission control and/or scheduling mechanisms.

Dynamic Virtual Machine Consolidation

As stated before, a capability provided by virtualization is live migration [16], which is the ability of transferring a Virtual Machine (VM) between physical servers with a very low down time close to zero. Using live migration [16] VMs can be dynamically consolidated to always keep the number of active physical hosts at the minimum. Dynamic VM consolidation has two basic processes; 1) migrating all VMs from underutilized hosts to minimize the number of active hosts; and 2) offloading VMs (by migrating them to another hosts) from overloaded hosts to avoid performance degradation experienced by the application running on the VMs, which could lead to violate the QoS requirements. Once a host is becoming idle, it automatically switches to a low-power mode to eliminate the static power and reduce the overall energy consumption by the system. A host with low-power mode is reactivated when it is required to accommodate new VMs being migrated due to an offloading or under-utilization process.

Admission Control/Scheduling

Admission control (AC) techniques try to handle the trade-off between increase of utilization of cloud data center's resources and risk of performance degradation of the applications, determining whether a new service should be admitted into the data center by evaluating the associated long term risk [83]. Once the services are accepted in a data center, a scheduler is in charge of finding a suitable physical resource to allocate them. In that process, it is decided which VMs are suitable to be collocated for improved utilization and stable performance. Overbooking is one of the well-known admission control technique which is based on accepting and allocating VMs based on real utilization ratios instead of nominal requested capacity, i.e., provisioning more VMs that physical available resources.

2.1.6 Scalability in Resource Management

Scalability is particularly important for IaaS cloud computing systems in order to enable the management of a large number of physical servers, virtual machines, and users. An example of the importance of scalability of resource management systems is the fact that Rackspace, a well-known IaaS cloud provider, currently manages tens of thousands of servers. Moreover, the number of servers continuously grows: Rackspace reported, it has increased the total number of servers in the second quarter of 2012 to 84,978 up from 82,438 servers at the end of the first quarter [71]. Another benefit of having scalable resource management is the improved fault tolerance by eliminating single points of failure: even if a compute or controller node fails, it would not render the whole system inoperable. The resource management found in the literature typically focus on centralized approaches, with a single management node. These approaches suffer from poor scalability, as the management node may become a performance bottleneck if the number of physical and virtual machines grows in data center.

2.2 Scientific Models, Theories, and Methods used in this Thesis

In this section, we briefly explain the well-established models, theories, and concepts used in the proposed contributions of this thesis. They are either concepts from the Artificial Intelligence domain, such as Reinforcement Learning, Multi-Agent Systems, and bio-inspired algorithms such as gossip/epidemics protocols.

2.2.1 Autonomic Computing and Machine Learning

Most of the researches on self-managing systems aims to build various kind of automated self-* management policies by engineering human expert domain knowledge. For example, in real-time system performance management, most of the researchers focused on designing and constructing explicit system-performance models, such as queuing-theoretic or control-theoretic models, that effectively optimize in real time the performance of complex

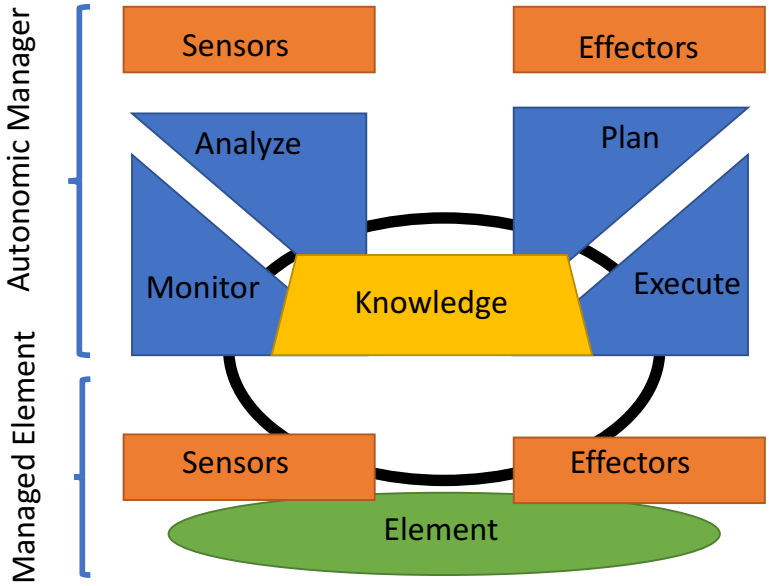


Figure 2.8: A standard autonomic computing loop

computing systems. As illustrated in Figure 2.8, domain knowledge is the most important feature of autonomic computing's general operation (IBM-proposed vision) [81]. The figure shows a generic relationship between an autonomic manager software element and a managed element which would be hardware or software. The autonomic manager continually executes a monitor-analyze-plan-execute (MAPE) loop. It first observes sensor readings then analyzes and plans an appropriate management decision, and finally executes that decision via effectors. As figure shows, other components have always access to the central knowledge base which contains knowledge pertaining to the likely effectiveness of various possible management decisions in achieving the objective of the manager's overall policy. Such Knowledge typically is presented in the form of an explicit system model. It is responsible to estimates how various management actions/decisions would affect subsequent states of the managed element. It might include predictions of any external processes that generate work-flow or traffic processed by the managed element. The bottleneck for developing such self-* managing systems is the difficulty of engineering an accurate knowledge module that can achieve a reasonable performance in deployed systems which are nowadays very complex and distributed.

Machine learning (ML) is able to overcome the described knowledge bottleneck [81]. ML is a sub-field of artificial intelligence that aims to develop methods for automatically extracting knowledge from data. For example, the Supervised Learning uses a data set of (input, output) pairs to learn a classification or regression model providing a deeper "understanding" of the relation between inputs and outputs beyond the specific training instances. Ideally, the learning algorithm will correctly generalize to new instances not seen during training. Likewise, Unsupervised Learning uses data without any target

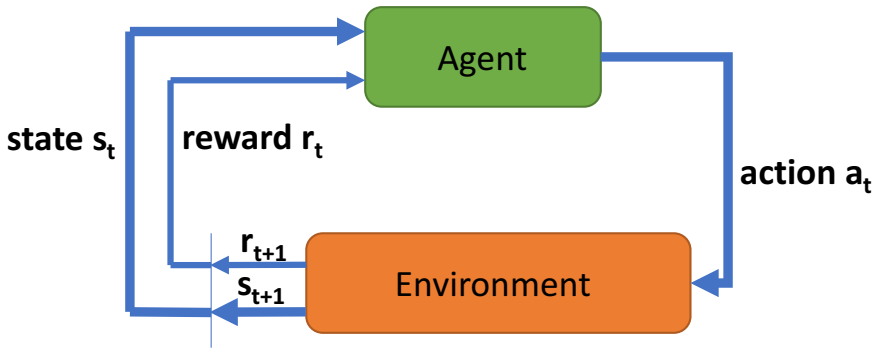


Figure 2.9: A standard reinforcement learning

outputs to discover unknown structures or relationships. Such knowledge would be very useful in data mining or for grouping data into closely related clusters. The third group of the machine learning algorithm which are not supervised nor unsupervised is Reinforcement Learning (RL). In this thesis, we focus on the application of RL in autonomic computing. In its most commonly studied form, RL aims to learn effective management policies in the absence of explicit system models, with little or no domain-specific initial knowledge and in the next section we discuss about it in details.

2.2.2 Reinforcement Learning

Reinforcement Learning (RL) provides a novel approach to systems management that differs radically from more traditional explicit model-building approaches. RL ideally offers a way to avoid knowledge bottlenecks by automatically learning high-quality management policies with little or no built-in system-specific knowledge [81]. Moreover, RL is grounded in a powerful sequential decision theory; in principle, it should be able to learn optimal policies, fully accounting for a decision's long-range dynamic consequences. In normal operation of an RL agent, illustrated in Figure 2.9, an agent learns effective decision-making policies through an online trial-and-error process by interacting with an environment. As Figure 2.9 illustrates, the agent at each time-step t, as a learner interacts with the environment, observes the current state s_t of the environment, and selects an action a_t from the set of possible actions corresponding to the state. One time-step later, as a consequence of the chosen action, the agent receives a numerical reward r_{t+1}, and finds itself in a new state s_{t+1}. The learning's goal simply, is to find best possible action in each state that will result in the maximum total sum of the reinforcement signals, rewards.

The model has been constructed on top three formal bases:

- Set of environment states, S: It can discrete or continuous, finite or infinite.
- Set of agent actions, A: It can be discrete or continuous.

- Set of scalar reinforcement signals or reward: Either Boolean 0,1, or continuous real-valued.

RL is a goal-directed ML technique, mostly based on trial and error paradigm in which the agent will keep trying to explore its environment (by an exploration/exploitation strategy) and perform possible actions until finding the best policy, i.e. the best set of actions/decisions among possible actions, which results in maximum cumulative reward. RL problems generally consists five components [80] as follows:

1. **Agent:** The learner that can interact with the environment through a set of sensors and actuators.

2. **Environment:** A time varying deterministic/non-deterministic stationary media, expressed by Markov Decision Process (MDP). Non-deterministic means the resulting state and reward signal of taking the same action in the same state on two different occasions may differ; whilst stationary means that the probabilities and expectation of getting such results, i.e. special state transitions or rewards, are constant and not time varying.

3. **Policy π:** A mapping from perceived states set S to the actions set A; it also determines what action should be taken afterwards.

4. **Reward function (goal definition) R:** A mapping from state-action pairs to a scalar number indicating the intrinsic desirability of that state-action. It stems from the goal of the RL task. It maybe delivered to the agent with a delay and scarcely.

5. **Value function:** The total amount of reward an agent can expect to accumulate over future, starting from that state. It depends also on the choice of optimality model. Main difference in RL algorithms is about how to approximate this value function. Note that a reward function indicates what is good in an immediate sense, while value function specifies what is good in the long run.

General Challenges for RL Applications

There are significant challenges in RL applications [63] as follows:

- **Delayed reward:** In many reinforcement learning problems often a sequence of actions will be taken before receiving any reward signal from the environment. These type of problems actually happen when for instance the agent receives rewards not immediately for every action applied to the environment, but after having a period, for a while. The problem with delayed reward is how to credit or blame past actions which have contributed to the delayed reinforcement signal which received later. This problem known as temporal credit assignment problem. Famous class of solutions for this problem is called temporal difference learning will be discussed later.

- **Exploration/Exploitation trade-off:** At each time slot, applying the best policy determined so far is called exploiting the learned skill or behavior. In the meantime in order to be fully confident about optimality of learned behavior all possibilities must first be tried (explored) statistically enough. Since exploration and exploitation can not be handled simultaneously, there is always a trade-off between them during the learning process of the agent. There exist some methods to somewhat soften this problem but still with the lack of no precise solution. In practice mostly a hybrid approach will be used.

- **Curse of dimensionality:** In many real life problems the input state space dimension is large and consequently the number of possible states is so huge, rather infinite or sometimes even uncountable number in continuous environments. Having these limitations exposed, the classical RL algorithms fails to be implemented, and are usually intractable to represent since they use mainly large memory tables as lookup tables (LUT) for saving and keeping the track of state-action quality values. This type of problems are called curse of dimensionality and will be treated with the use of more advanced RL methods and generalization techniques which will be discussed later.

- **Partially observable and hidden states:** In real life environments sometimes that agent does not have full state information of the environment but just a partial perception because of its limited field of view; so in these situations we have partial observability. Also the presence of hidden states can cause indirectly this problem. Hidden states are those states which exist in actual model of the system but the agent will never discover them and they are beyond the agent's perception.

Markov Decision Process

All characteristics of the interaction between the agent and environment can be fully represented in the form of Markov Decision Process (MDP). MDP is a multi state stochastic system in which the states change by transition probabilities. An MDP usually is represented by a directed graph in which nodes are the states, the edges are the transitions from one state to another by applying an action and a reward associated to that and the weights are the transition probabilities. Figure 2.10 shows an example of a MDP state diagram.

In Figure 2.10, a_k is the k^{th} possible action in each state. p_{ij} is the transition probability from state i to state j. Therefore $a_k : p_{ij}$ on the edge means a transition from state i to state j by taking action a_k with probability of p_{ij}. Note that in the context of reinforcement learning there is a scalar value as the reinforcement signal associated to each state transition. It is worth mentioning that an MDP can be generally non-deterministic or deterministic. In a formal way and with respect to time, if random variable S_t denotes the state of the environment at time step t, A_t the action taken at this time, and $s^{'}$ and r as the resulting state and reward respectively, then the state transition probabilities are governed by:

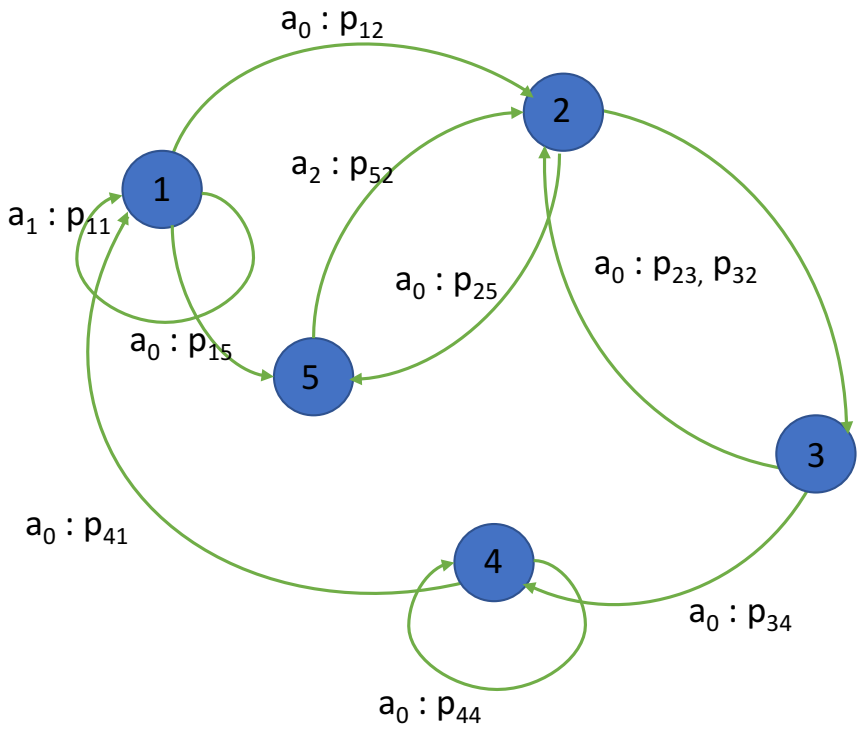


Figure 2.10: An MDP state diagram

$$P_{s_t s_{t+1}}(A_t) = P_r(S_{t+1} = s^{'}, r_{t+1} = r \mid S_t, A_t) = \\ P_r(S_{t+1} = s^{'}, r_{t+1} = r \mid S_t, A_t, r_r, S_{t-1}, A_{t-1}, r_{t-1}, ..., r_1, S_0, A_0) \tag{2.1}$$

meaning, that the likelihood of arriving in state $s^{'}$ depends entirely on the current state S_t and the corresponding action A_t. This property, which is crucial to the application of reinforcement learning algorithms, is called Markov Property. In above equation r means the reinforcement signal received previously by transiting to the state S_t. Recall that S represents the state set, and A the action set, then in mathematical notation, reward function R and transition probability function p are as follow:

$$R : S \times A \to \Re \\ p : S \times A \times S \to [0, 1] \tag{2.2}$$

Markov chain is another expression in MDP methodology assigned to the sequence of successive environmental states, resulting from the actions taken by the learning agent over time [33].

Optimality Models

As stated before, in Reinforcement Learning (RL) problem, policy π is a mapping form states to actions $\pi : S \rightarrow A$, consequently each policy has a value function V^{π} determines a numerical value as the quality of mapping each state $s \in S$ to each action $a \in A$. In order to calculate V the optimality model should be specified. Generally there are three models of optimal behavior based on the way to take the future consequences of the actions into account, as surveyed in [41]:

- **Finite-horizon model:** In this model the agent should optimize its expected total reward for the next h steps as its life time.

$$E(\sum_{t=0}^{h} r_t). \tag{2.3}$$

  Here the agent has a limited h-deep foresight into the future and not further.

- **Infinite-horizon model:** This model is based on the long-run reward covering whole period. Infinite-horizon discounted model treats future rewards in an exponentially discounted manner with respect to time (who far they are), as expressed in:

$$E(\sum_{t=0}^{\infty} \gamma^t r_t), \qquad 0 \leq \gamma \leq 1. \tag{2.4}$$

  Where γ is the discount factor and infinity sign, ∞, means the whole period of the agent's life time in which it will learn.

- **Average-reward model:** In this model the agent tries to maximize the average total long-run reward as follows:

$$\lim_{h \rightarrow \infty} E(\frac{1}{h} \sum_{t=0}^{h} r_t). \tag{2.5}$$

In this thesis we are using the infinite-horizon model due to its generality. After having infinite-horizon optimality model now we can calculate the value function V^{π} as follow:

$$\begin{aligned} V^{\pi}(s) &= E_{\pi}[\sum_{t=0}^{\infty} R(S_t, A_t) | S_0 = s] \\ &= E_{\pi}[\sum_{t=0}^{\infty} R(S_t, \pi(S_t)) | S_0 = s], \qquad 0 \leq \gamma \leq 1 \end{aligned} \tag{2.6}$$

Where E is the expectation of the total reward obtained by taking the policy π from the start state s, $\pi(S_t)$ is the action taken under policy π in state S_t, and $R(S_t, A_t)$, or simply r_{t+1}, is the expected reward after performing action At.

MDP has a formal way to handle the problem of credit assignment (raised by delayed reward) trough selecting of the optimality model which have been described above. Note that the original MDP methodology is basically founded on finite discrete states and actions. However, there are some strategies allow it to be extended to infinite or even uncountable number of states and actions. In addition, MDP can even be approximated where the assumption of Markov property may not hold due to system complexity, having limited, inconsistent or faulty perception.

Dynamic Programming

Dynamic Programming (DP) can be used for solving the problem modeled completely by MDP. DP provides algorithms to find optimal policy π^* with corresponding optimal value function V^* for perfect model of MDP. Optimal policy is the policy that its corresponding value function provides the highest value for all states of the environment:

$$V^*(s) = \max_{\pi} E_{\pi}[\sum_{t=0}^{\infty} \gamma^t r_t], \qquad \forall s \in S,\ 0 \leq \gamma \leq 1 \tag{2.7}$$

Let's come back to Eq. 2.6 and try to find an approximation for that. It can be done by Bellman equation as follow :

$$V^{\pi}(s) = \sum_{a \in A} \pi(s,a) \sum_{s' \in S} P_{ss'}(a)[R(S_t, A_t) + \gamma V^{\pi}(s')], \qquad \forall s \in S,\ 0 \leq \gamma \leq 1 \tag{2.8}$$

Where $\pi(s,a)$ is the probability of getting action a in state s under policy π. The application of Eq. 2.8 yields a set of linear equations for each state, the solution of which results in V^{π}. Therefore the Bellman optimality equation (Eq. 2.9) based on Eq. 2.7 and Eq. 2.8, for approximating V^*, can be derived which yields the expected reward of each state if the best possible action is always taken. The derivation is straight forward since here the attention is on V^* , and hence concerning solely with the best policy (best action which maximizes the future reward), not with all possible actions:

$$V^*(s) = \max_{a} \sum_{s' \in S} P_{ss'}(a)[R(S_t, A_t) + \gamma V^*(s')], \qquad \forall s \in S,\ 0 \leq \gamma \leq 1 \tag{2.9}$$

Eq. 2.9 represents a set of simultaneous linear equations. For having less computational cost, there are two model-based method [41]: 1) Value Iteration 2) Policy Iteration which try to find optimal value function and optimal policy respectively by iteration. However, these solutions are suitable when the complete model of the environment (including reinforcement function $R(S_t, A_t)$, and transition probabilities $P_{ss'}(a)$) to be known in advance. Otherwise, more enhanced form of methods, usually called Adaptive DP, should be employed to overcome the problems emerged. Adaptive DP methods have two categories: 1) Model-free and 2) Model-based. In Model-free or direct techniques, value function and policy will be directly learned via interaction with the environment, without learning a model in prior. While in model-based or indirect adaptive control methods, the system first learns a model and then uses that to find an optimal policy. In

this thesis we just consider model-free methods which are more general and applicable in this context such as temporal difference and Q-learning.

Temporal Difference Learning and Q-Learning

Temporal Difference (TD) Learning is a combination of Monte Carlo and Dynamic Programming (DP) ideas. TD resembles a Monte Carlo method because it learns by sampling the environment, and is related to dynamic programming techniques as it approximates its current estimate based on previously learned estimates. In general, TD methods can learn directly from raw experience without a model of the environment's dynamics and update estimates based in part on other learned estimates, without waiting for a final outcome [80]. TD deals with an approximation of value function $V(s)$, denoted by $\widetilde{V}(s)$, that is updated based on the difference between the $\widetilde{V}(s)$ and $\widetilde{V}(s^{'})$, where $s^{'}$ is the immediate state visited after s. The basic TD algorithm called TD(0) update rule is:

$$\widetilde{V}_{t+1}(s) = \widetilde{V}_t(s) + \alpha[r_{t+1} + \gamma\widetilde{V}_t(s^{'}) - \widetilde{V}_t(s)], \qquad 0 < \alpha < 1 \tag{2.10}$$

Where α is the learning rate. Whenever a state s is visited, its estimated value is updated to be closer to $r + \gamma\widetilde{V}_t(s^{'})$, since r is the immediate reward received and $\widetilde{V}(s^{'})$ is the estimated value of the actually occurring next state. The key idea is that $r + \gamma\widetilde{V}_t(s^{'})$ is a sample of the value of $\widetilde{V}(s)$, and it is more likely to be correct because it incorporates the real r. The criterion for being close is the term in square bracket in Eq. 2.10 which is called TD Error.

$$\widetilde{\varepsilon}_{t+1} = r_{t+1} + \gamma\widetilde{V}_t(s^{'}) - \widetilde{V}_t(s) \tag{2.11}$$

If the learning rate α is adjusted properly (it must be slowly decreased), and each state is sufficiently visited under a fixed policy π, then TD(0) guarantees to converge to the optimal value function, [80] [41]. Q-leanring (QL) is an off-policy TD control, it is one of the most popular RL methods developed by Watkins [94], it involves in finding the quality of state-action rather than just state values. Assume that $Q^*(s,a)$ be the expected discounted reward of taking action/decision a in state s, and then choosing actions optimally afterwards. Then an estimation of optimal state-action value function, $Q^*(s,a)$, denoted by $\widetilde{Q}(a,s)$ will be:

$$\widetilde{Q}_{t+1}(s,a) = \widetilde{Q}_t(s,a) + \alpha[r_{t+1} + \gamma \max_{a^{'}} \widetilde{Q}_t(s^{'},a^{'}) - \widetilde{Q}_t(s,a)], \qquad 0 < \alpha < 1. \tag{2.12}$$

Where α is the learning rate and $a^{'}$ is the next action to be taken from state $s^{'}$. In general Q-leanring has a lower rate convergence in comparison with the other RL algorithms; however, it is exploration insensitive which means that Q values will converge to optimal values no matter what exploration/exploitation strategy used in action selection mechanism [41].

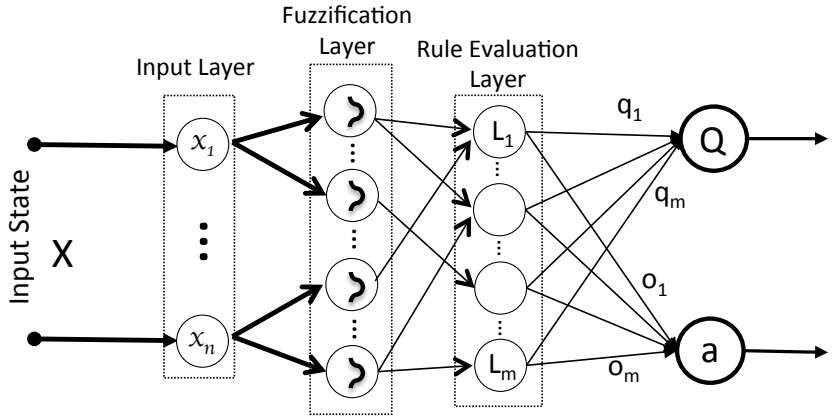


Figure 2.11: FQL structure

Fuzzy Q-learning

The most significant drawback of Q-learning is that it cannot be used in case that the state space is large, a situation found in many real world problems, since it uses mainly large memory for saving Q-table (for keeping the track of state-action quality values). Apart from it, even if the system provides such large memory, the learning agent needs a lots of trials and episodes to learn desired behavior resulting in increasing the cost of learning.

Fuzzy Q-learning (FQL) [30] is a fuzzy extension of the Q-learning algorithm being able to overcome this problem. In addition, it allows us to encapsulate expert knowledge into the learning table resulting in speeding up the learning process. In FQL, the decision making part is represented by a Fuzzy Inference System (FIS) that considers continuous/large discrete states as input. The idea of the FQL algorithm is to use a so-called q-table as a compact version of the Q-table to represent learning knowledge.

In FQL, the FIS is represented by a set of rules J with a rule $j \in J$ defined as

$$\textbf{IF}(x^1 \ is \ L_j^1) \ldots \textbf{AND} \ (x^n \ is \ L_j^n) \ldots \textbf{AND} \ (x^N \ is \ L_j^N)$$
$$\textbf{THEN} \ a = o_j \text{ with } q(L_j, o_j). \quad (2.13)$$

L_j^n is a fuzzy label of the input variable x^n of the state vector $x = [x^1, \ldots, x^n, \ldots, x^N]$ participating in the j^{th} rule, and $o_j = [o_j^1, \ldots, o_j^k, \ldots, o_j^K]$ is the output action set of the j^{th} rule. The vector $L_j = [L_j^1, \ldots, L_j^n, \ldots, L_j^N]$ is called the modal vector corresponding to the rule j. $q(L_j, o_j^k)$ is called the q-value function of state L_j and action o_j^k of the j^{th} rule.

In the Fuzzification Layer of FIS (see Figure 2.11) each membership function (μ_L) maps a state component into the *degree of membership* to a fuzzy set corresponding to a given label. Let J_x denote the set of all rules in the Rule Evaluation Layer of FIS (see Figure 2.11). The membership of a state vector x, or the *degree of truth* in the fuzzy

logic terminology (represented by α), with respect to the j^{th} rule, $j \in J_x$ is defined as the product of the corresponding member functions of the rule:

$$\alpha_j(x) = \prod_{n=1}^{N} \mu_{L_j^n}(x_n). \tag{2.14}$$

In the FQL algorithm, we have a two-level action selection. In the first level (we call it local action selection) a set of actions is chosen according to an Exploration/Exploitation (EE) policy. An EE policy allows the agent to explore untried actions to gain more experience, and combine this with exploitation of the already known successful actions to ensure high long-term reward. The ϵ-greedy method [82] is used as the EE policy for our experimental studies. With ϵ-greedy, at each time step, the agent selects a random action with a fixed probability $1-\epsilon$, where ϵ is typically chosen close to and below 1, instead of selecting greedily one of the learned optimal actions with respect to the q-table:

$$\begin{cases} \text{with Prob. } \epsilon & \forall\ j \in J_x : o_j^l = \text{argmax}_{k \in K} q(L_j, o_j^k) \\ \\ \text{with Prob. } 1-\epsilon & \forall\ j \in J_x : o_j^l = \text{random}_{k \in K}(o_j^k) \end{cases} \tag{2.15}$$

Where o_j^l is the selected local action of the j^{th} rule. In the second level of the action selection (we call it inferred action selection) a nominated action for input vector x is selected from the set of local actions as follow:

$$a = \max_{j \in J_x} \alpha_j(x) o_j^l. \tag{2.16}$$

Let us add time index to our equation to highlight time dependency from now on. The approximation of the quality value of the state x_t is calculated based on the following equation:

$$Q(x_t, a) = \sum_{j \in J_{x_t}} \alpha_j(x_t) \ \times \ q_t(L_j, o_j^l). \tag{2.17}$$

After taking action a the system goes to the next state x_{t+1} and observes the reward r_{t+1}. The state value for the input vector x_{t+1} is calculated as follow:

$$V(x_{t+1}) = \sum_{j \in J_{x_{t+1}}} \alpha_j(x_{t+1}) \times \ \max_k q_t(L_j, o_j^k). \tag{2.18}$$

Based on the equation (2.17) and (2.18) the temporal-difference (TD) error is calculated as follows:

$$\triangle Q = r_{t+1} + \gamma\, V(x_{t+1}) - Q(x_t, a). \tag{2.19}$$

Here γ again being the discount factor. Finally the q-function is updated for each activated rule $j \in J_{x_t}$ according to the rule:

$$q_{t+1}(L_j, o_j^l) = q_t(L_j, o_j^l) + \beta\, \alpha_j(x_t) \triangle Q \tag{2.20}$$

where β is the learning rate. The FQL algorithm is shown in Algorithm 2.1.

Algorithm 2.1: FQL Algorithm

initialize(q) {for all $j \in J_x$, $q(L_j, o_j)$ is initialized by zero}
α = calculateTruthValue(x_0) {Eq.(2.14)}
repeat
  o = actionSelection(q) {Eq.(2.15)}
  a = inferredAction(α, o) {Eq.(2.16)}
  Q = calculateQValue(α, q) {Eq.(2.17)}
  {the agent execute action a and observes new state x_{t+1}}
  r_{t+1} = receiveReward()
  α = calculateTruthValue(x_{t+1})
  V = calculateStateValue(α, q) {Eq.(2.18)}
  $\triangle Q$ = calculateQualityVariation(Q, V, r_{t+1}) {Eq.(2.19)}
  q = updateqValue(q, α, $\triangle Q$) {Eq.(2.20)}
  $t = t + 1$
until the FQL is terminated

One of the limitation of Fuzzy Q-learning (FQL) is that, a prior or expert knowledge must be available to partition the input space and design the fuzzy sets for the input parameters. Dynamic Fuzzy Q-learning (DFQL) [40] is an extension for Fuzzy Q-learning (FQL) adopting an "aligned clustering algorithm" to generate fuzzy sets during learning. In this thesis, we used a fuzzy c-means (FCM) [10] clustering algorithm in our proposed DFQL algorithm.

2.2.3 Multi-Agent Systems

In recent years, the interest in decentralized approaches to solving complex real-world problems has been increased. Many such approaches fall into the area of distributed systems in which a number of entities cooperate together to solve problems. The combination of distributed systems and artificial intelligence (AI) is known as distributed artificial intelligence (DAI). Multi-agent systems (MAS) is one of the subset of distributed artificial intelligence, emphasizes the joint behaviors of agents with some degree of autonomy and the complexities arising from their interactions. An agent is a computational unit which has a high degree of autonomy, performing actions in its environment based on information received from that. The information can be perceived through sensors or can be in the form of a feedback. A multi-agent environment is one in which there is more than one agent which can interact with one another. In addition, there are constraints on multi-agent environment such that agents may not, at any given time, know everything about the world that other agents know (including the internal states of the other agents themselves) [67].

Multi-Agent Learning

Multi-Agent Learning is the application of machine learning to problems involving multiple agents. There are two features of multi-agent learning which make it specific

to be study as a separated field from ordinary machine learning. First, As multi-agent learning deals with problem domains involving multiple agents, the search space involved can be unexpectedly large, therefore, due to the interaction of those agents, small changes in learned behaviors can often result in unpredictable changes in the resulting macro-level ("emergent") properties of the multi-agent group as a whole. Second, multi-agent learning may involve multiple learners, each learning and adapting in the context of others; this introduces game-theoretic issues to the learning process which are not yet fully understood [67].

Because of the inherent complexity in the interactions of multiple agents, various machine learning methods- notably supervised learning methods-are not easily applied to the problem because they typically assume a critic that can provide the agents with the "correct" behavior for a given situation. Thus the very large majority of papers in this field have used reward-based methods. The reward-based learning literature may be approximately divided into two subsets: reinforcement learning methods which estimate value functions; and stochastic search methods such as evolutionary computation, simulated annealing, and stochastic hill-climbing, which directly learn behaviors without appealing to value functions. One large class of learning in multi-agent systems is cooperative learning where a team of independent learning agents try to coordinate their individual behavior to reach a coherent joint behavior [67].

Cooperative Reinforcement Learning

There are several ways to enforce cooperation between Reinforcement Learning agents. In this thesis we make use of a blackboard communication schema for this reason [99], which is simple to be implemented. It is a shared memory that can be accessed by every agent for reading and writing. Therefore, there is no direct communication between agents and the data flow between agents is redirected through the blackboard. In our cooperative reinforcement learning algorithm, the blackboard maintains a single lookup table for all agents inside the coordinator unit. Therefore, cooperation can be enforced by using a global reward comprising of the sum of rewards of all the individual learning agents, which are feeding a single shared lookup table.

2.2.4 Gossip/Epidemic Protocol

Gossip/epidemic as a nature-inspired algorithm is of interest for large scale distributed systems as is has several attractive properties like speed, simplicity, robustness, and a lack of central control and bottlenecks. These properties are very important for information dissemination and collective information processing (aggregation) that are both key components of large scale distributed systems [39]. In the research carried out in the scope of this thesis we mainly focus on the aggregation component of gossip algorithm/protocol and we treat it as inspiration for the design of robust and fault tolerant self-organizing resource management strategy in cloud data centers.

Data Aggregation

The most natural application of gossip (or epidemics) in computer systems is information dissemination. However, the gossip communication paradigm can be generalized to applications as well. In these applications, there is still some implicit of spreading information, but the emphasis is on its processing on the fly. Computing a global function over the set of nodes based only on gossip-style communication can be a process for creating a summaries of distributed data. As an example, imagine we want to have maximum or average of some attributes of the nodes. The problem of computing such global functions is called data aggregation or simply aggregation. If the problem becomes more complex to compute, such as fitting models on a fully decentralized data, we need to switch on the topic of distributed data mining which in not in the scope of this thesis. In data aggregation community, a lot of effort has been directed at a specific problem: calculating averages, which has been considered in this thesis as our main problem to propose a fully distributed dynamic virtual machine consolidation strategy. Averaging can be considered the archetypical example of aggregation. Let y_i be an attribute value at node i for all $0 < i \leq N$. We are interested in the average $\overline{y} = \sum_{i=1}^{N} y_i/N$. One of the simplest algorithm to calculate average is push-pull averaging, presented in Algorithm 2.2 [39].

Algorithm 2.2: General Algorithm for Push-Pull Averaging

```
loop
  wait(△)
  p ← random peer
  send push(x) to p
end loop
ONPUSHUPDATE(m)
send pull(x) to m.sender
x ← (m.x + x)/2
ONPULLUPDATE(m)
x ← (m.x + x)/2
```

Each node periodically selects a random peer to communicate with, and then sends the local estimate of the average x. The recipient node then replies with its own current estimate. Both participating nodes (the sender and the one that sends the reply) make an average of the two previous estimates as a new estimate and save it. Now let us first look at the convergence of the algorithm [39]. It is clear that the state when $x_i = \overline{y}$ for all i is a fixed point with this assumption that there are no node failures and message failures, and the messages are delivered without any delay. It is not very difficult to convince oneself that the algorithm converges to this fixed point in probability. We omit the technical details of the proof; the trick is to first observe that the sum of the approximations remains constant throughout. More precisely,

$$\sum_{i=1}^{N} x_i(t) = N\overline{y} \tag{2.21}$$

for any t. This property is very important and is called mass conservation. We then look at the difference between the minimal and maximal approximations and show that this difference can only decrease and, furthermore, it converges to zero in probability, using the fact that peers are selected at random. But if all the approximations are the same, they can only be equal to $\overline{y}$ due to mass conservation. The really interesting question, however, is the speed of convergence. The fact of convergence is easy to prove in a probabilistic sense and it is possible to characterize convergence speed by studying the variance-like statistic

$$\sigma^2{}_N = \frac{1}{N}\sum_{i=1}^{N}(x_i - \overline{y})^2 \tag{2.22}$$

which describes how accurate the set of current estimates is.

$$E(\sigma^2{}_N(t+1)) \leq \frac{1}{2}\sigma^2{}_N(t) \tag{2.23}$$

where the time index t indicates a cycle. That is, variance decreases by a constant factor in each cycle. In practice, 10-20 cycles of the protocol already provide an extremely accurate estimation: the protocol not only converges, but it converges very quickly as well.

CHAPTER 3

Related Work

In this chapter, we present the related work, organized into the following structure: Section 3.1 presents the literature on dynamic virtual machine consolidation strategies and Section 3.2 describes the existing work related to overbooking techniques to improve the utilization in cloud infrastructure. Since machine learning approaches are used in our work, in Section 3.3, we carefully explore the most relevant machine learning approaches, that have been applied to enable resource management in the cloud computing domain.

3.1 Dynamic Virtual Machine Consolidation

Virtual Machine (VM) consolidation is one of the key mechanisms to design an energy-efficient dynamic Cloud resource management system. It is based on this assumption that packing VMs into fewer number of Physical Machines (PMs) can increase the utilization of the servers and consequently reducing the energy consumption of the Cloud data center. However, a performance degradation can occur due to an aggressive VMs packing into a single server, since VMs share the underlying physical resources of the PM. To address this problem, VM Consolidation (VMC) algorithms are designed to periodically/dynamically make optimal decisions by considering the impact on the level of QOS in a packing process. Energy efficient dynamic VM consolidation proposed in the literature can be divided into two different categories: (1) VM consolidation based on periodic adaptation and (2) VM consolidation based on dynamic decision making.

3.1.1 VM Consolidation Based on Periodic Adaptation

In this category, the virtual machine consolidation is mostly addressed as a multi-dimensional bin packing problem (where virtual machines and physical host nodes are considered as objects and bins respectively) and is solved periodically by a heuristic or meta heuristic algorithm. The solution is a migration map whereby virtual machines are reallocated on the least number of physical host nodes. Feller et.al. [23] [21] model the

problem of virtual machine consolidation into a bin packing problem and propose an Ant Colony Optimization algorithm to solve it. However, the migration cost was not considered in the objective function and the authors assumed that the algorithm is triggered after predefined, sufficiently long periods of time to minimize the amount of migrations and limit the degree of performance degradation. In [35], a genetic algorithm is proposed to solve the problem of dynamic VM consolidation as a multi-dimensional bin packing problem. The objective function (fitness function in evolutionary computing terminology) includes a penalty which can be adopted to a energy prices or live migration overheads. Another ant-colony based consolidation algorithm has been proposed in [24]. The authors integrate Ant Colony Optimization algorithm with balanced usage of computing resources based on vector algebra. However, They just address power consumption and resource wastage and don't consider any performance indicators such as migration cost indicators in their proposed multi-objective function.

In this category, there are also some researches which consider scalibility in VM consolidation problem and propose some fully distributed algorithms to solve that. Feller et.al. propose an extended version of their previous work in [22] [21] and present a migration-cost aware ant-colony based algorithm for the problem of dynamic VM consolidation. In this paper, they also leverage the swarm intelligence nature of the ant colony algorithm to propose a fully decentralized algorithm on an unstructured P2P overlay. Marzolla et al. [52] present a fully distributed VM consolidation algorithm called V-MAN based on a simple gossip protocol. The algorithm is executed periodically to identify a new arrangement of existing VM instances in a scope of neighborhoods in order to maximize the number of empty servers (i.e., servers running no VM) inside the data center. The proposed algorithm assumes that the physical and virtual machines are homogeneous. The authors presented a rather simple performance criterion by determining a capacity limit for physical host nodes in terms of number of VMs (i.e., each physical server can host just C virtual machines). In addition, they do not consider migration costs in the decision making process.

3.1.2 VM Consolidation Based on Dynamic Decision Making

One of the first works, in which dynamic decision making has been applied in VM consolidation has been performed by Nathuji and Schwan [65]. The authors propose a two-level policy to manage the resources. At the local level, the system coordinates and leverages power management policies of guest VMs at each physical machine. An example of such a policy is the on-demand governor which is integrated into the Linux kernel. At this level the decisions are about changing in power states which are issued externally by guest OS specific policies to maintain the QoS level. The global policies are responsible for managing multiple physical nodes and use knowledge of rack- or blade-level characteristics and requirements. These policies consolidate VMs using live migration in order to offload resources and place them into power saving states. They consider both the energy consumption and the VM migration cost in their strategy. However, the authors do not provide a detailed description of employed global policies resulting in limiting the analysis of the approach. Gmach et al. [31] proposed a two level management

for the problem of dynamic VM consolidation. The first level is responsible for placing virtual machines, while the second level is responsible for migrating them. The first level, being a trace-based workload placement controller, collects data on resource usage by VMs instantiated in the data center, and uses this historical information to optimize the allocation, while meeting the specified quality of service requirements. The second level controller is a fuzzy logic based feedback controller operating as a reactive migration controller. An advisor module of the controller continually monitors the servers resource utilization and triggers a fuzzy logic based controller whenever resource utilization values are too low or too high. When the advisor detects an underutilized or overutilized situation, the fuzzy controller module identifies appropriate decisions to remedy the situation. In [7] the authors proposed a distributed system management architecture to design a distributed dynamic VM consolidation for cloud data centers. In this work the authors proposed a software layer being tiered and comprising local and global managers. The local managers reside on each physical host node as a module of the VM monitor (VMM). Their objective is to continuously monitor a node's CPU utilization, resizing the VMs according to their resource needs, and deciding when and which VMs have to be migrated from the host node. The global manager resides on a master node and collects information from the local managers to maintain the overall view of the utilization of resources. The global manager issues commands for the optimization of the VM placement. The authors in the next papers [6] [8] presented several heuristic algorithms for overloading detection, VM selection and also VM placement in the aforementioned system architecture.

In this category, Barabagallo et al. [3] modeled a fully decentralized data center as a P2P network of self-organizing nodes that collaborate with one-another using a bio-inspired algorithm. This collaboration allows the nodes to redistribute the load among servers in order to increase the system's energy efficiency. Their idea is to have a number of entities known as scouts that investigate and gather information about virtual machines in the data center. This information is then used by other virtual machines to make decisions about whether they should initiate migrations or not.

3.2 Overbooking Techniques

Although resource overbooking has been previously studied [36], there are still no complete solutions that make an holistic management of all the steps needed to perform a safe overbooking, such as admission control, mitigating VMs interference, QoS differentiation, fairness performance degradation, etc. In the literature we find, among many others, the work proposed by Urgaonkar et al. [91] where a feedback control approach to safely overbook cluster resources is presented, guaranteeing applications performance, but assuming that users are capable of providing information regarding the overbooking tolerance of their applications. As previously studied in [83] [84], this is strongly coupled to the underlying physical infrastructure, as well as to the collocated applications. There are other works also focusing on the admission control problem, such as the ones presented in [12] [29] and [83]. However, they do not present a complete solution, and once the

applications are accepted, there is no mechanism to deal with possible resource shortages due to flash crowds, mispredictions or VMs interference.

There are also works focusing on detecting, mitigating, and/or avoiding VMs interference, such as [9, 11, 97], or [100]. However, their solutions are either based on detecting the overloaded/interference problem and migrating VMs to alleviate the problem, or categorizing the VMs to avoid possible co-locations, instead of trying to adapt to the applications behavior and thus reduce possible VMs interference over time. Following that autonomic approach based on current applications needs, as well as to mitigate the impact of unexpected situations, we presented in [84] a feedback controller approach that self-optimizes the overbooking pressure based on running VM behavior, in cooperation with an application performance steering approach that ensures graceful degradation during load spikes. However, once again the focus is on mitigating the problem first and trying to avoid it in the near future, instead of directly reducing the VMs interference. Consequently it has an impact on overall resource utilization.

Regarding VM co-location and interference problems, the work presented in [47] concluded that queuing delays, scheduling delays and load imbalances all significantly impact performance. An interesting approach is presented by Delimitrou et al. [18]. They propose a scheduler that classifies and allocates the incoming applications based on their profiled and expected interference with other already running VMs. Note that our work focuses on the VM isolation inside a single server, hence both approaches could complement each other. A similar idea is presented in [89], where VMs are pinned to specific cores in order to avoid VMs interference. The pinning decisions are based on affinity values estimated by a fuzzy logic programming engine. Additionally, Nathuji et at. present the Q-Cloud [64] scheduling approach that deals with VMs interference by allocating some extra resource capacity to the VMs needing it to meet their SLAs. However, unlike our work, they are not targeting an overbooked environment. Therefore they assume there is always enough unused capacity in the server that can be used for helping the VMs suffering from the interference problem. There are similar works presented by Lo et al. [49] and Leverich et al. [47], where a set of hierarchical controllers and sub-controllers are used to both rise utilization but also ensure that best effort applications will not hurt latency-sensitive applications. However, they take the correction actions in a reactive way, based on the current situation while our approach takes proactive actions and learns over time from past behavior for each allocated latency-sensitive (high QoS) application. Besides, unlike our approach, they always prioritize the performance of the latency-sensitive applications, regardless of how far/close they are from their KPIs.

3.3 Machine Learning Approaches for Cloud Management

With the advent of autonomic computing, exploiting control theory, machine learning and soft computing techniques to construct adaptive systems (software that monitors and modifies its own behavior to meet goals) has been always attractive in this area. Fuzzy logic as a soft computing technique has been exploited by researchers to inject human

intuitive knowledge into controllers in different domains. The weakness of these controllers is that they can not be sophisticated enough in a highly dynamic environment as the human knowledge is not always available or accurate at design-time. With this respect, in [98] the authors proposed an adaptive resource allocation controller for virtualized datacenters by combining a fuzzy controller with a clustering algorithm. Although using a clustering algorithm as an extension to the fuzzy controller allowed it acted without requiring a-priori knowledge, it is still inefficient in a highly dynamic environment as it needs to run clustering algorithm every time new data is received due to the changing workload. Reinforcement learning (RL) algorithms amongst other approaches have been popular in this field. This popularity lies in the fact that they do not need any knowledge or training data which usually create a bottleneck to design adaptive systems dealing with a dynamic/time-variant environment. In fact, RL can generate the knowledge to make the system adaptive through learning over time. It is worth mentioning that, RL has been also appealing for researchers in artificial intelligence field to improve the learning speed. For instance, we can point to deep reinforcement learning [61] as one of the most recent efforts to improve reinforcement learning algorithm. Reinforcement learning has been employed several times in similar problems to ours, related to the problem of dynamic resource allocation in IaaS cloud computing, such as [72] [4] [38]. The significant different between our approach and these approaches is that they do not consider the VMs interference problem and therefore decisions are based on the amount of physical resources needed (e.g., number of cores) and not in how to use/share them more efficiently. By contrast, our approach considers the VMs interference problem and decides on the level of isolation needed among VMs that are sharing the same resources, thus VMs make dynamic decisions about its own resources to make the overall performance better, helping to increase the overall utilization. With this respect, Rao et al. [72] presented a reinforcement learning approach for virtual machine auto configuration in cloud environments. In this work, the RL agent learns how to change the configuration of the VMs in terms of memory size, number of virtual CPUs, and scheduler credit (running on a host node) with respect to the applications demands to maximize overall performance. In [4], the authors proposed a parallel reinforcement learning approach for the problem of auto scaling in IaaS cloud environment. Unlike our work that is based on the capacity management within the server, they focus on learning agents that can learn optimal scaling policies in terms of add, remove or maintain the amount of virtual machines allocated to the applications, i.e., in a different server. In [38] the authors proposed a Fuzzy Q-Learning (FQL) algorithm for the problem of auto-scaling. The FQL agent learns how to increment or decrement the number of virtual machines as a scaling action to meet a performance requirement for a given user.

CHAPTER 4

Intelligent Decision Making in Distributed Dynamic Virtual Machine Consolidation

One of the most important challenges in a virtualized cloud data center is to optimize the energy-performance tradeoff, i.e., finding the right balance between saving energy and attaining best possible performance. Distributed dynamic virtual machine consolidation (DDVMC) is a virtual machine management strategy that uses a distributed rather than a fully centralized algorithm for finding such optimums [8]. The general goal of DDVMC in data centers is to (1) manage physical host nodes in order to avoid overloading and under-loading, and (2) to optimize the placement of VMs. Managing physical host node in this strategy needs the controller to be involved in three decision making tasks: (1) determining when a host must be considered as overloaded (2) determining which virtual machines must be selected to be migrated away from an overloaded host node and (3) determining when a host must be considered as under-loaded. The most significant challenge here is that the optimally of DDVMC strategy is highly dependent on the quality of the decision-making tasks.

In this chapter, we address research question I by presenting their corresponding contributions I, II and III, introduced in Chapter 1. We first motivate the effect of intelligent decision making in the process of distributed dynamic virtual machine consolidation at Section 4.1. Then, in the next two following sections (Section 4.2 and Section 4.3) we present our approaches to make the host overloading detection and the VM selection process intelligent. Finally in Section 4.4 we propose a comprehensive solution to manage physical host nodes for distributed dynamic virtual machine consolidation by making use of a multi agent learning system paradigm.

4.1 Motivation

With the rise in demand of computing power by costumers cloud data centers evolve in size, resulting in ever-increasing energy consumption and consequently operating costs. Apart from it, the way energy is generated for cloud data centers influences our environment either directly by the carbon footprint or indirectly by the waste created by power plants. Therefore, cloud data centers should be managed in an energy efficient manner, and the main sources for energy waste must be identified.

An important waste of energy is due to the inefficient usage of computing resources. A study on power usage at the scale of data center workloads [20] shows the power usage dynamics of servers are very narrow. The same study indicates idle power in servers is always above 50% of peak power. Therefore, keeping server underutilized is very inefficient from the energy consumption point of view. In a cloud data center, virtualization can provide server consolidation by allowing multiple Virtual Machine (VM) instances to run on a single physical server, resulting in fewer running physical servers with higher per-server utilization.

However, server consolidation enabled by virtualization introduces a new problem to the cloud data center. Since the amount of resources occupied by VMs inside a physical server might change due to the dynamic and unexpected behavior of applications running on the virtual machines, resource underutilization or overutilization might occur inside a physical server, causing either inefficient energy consumption or undesired performance degradation. To remedy this situation, several dynamic VM consolidation algorithms implementing different management strategies have been proposed by researchers. The main principle of theses strategies is to migrate virtual machines to different physical machines dynamically in order to minimize both inefficient energy consumption, while maximizing the performance of the system in terms of quality of service (as expressed e.g. by Service Level Agreements (SLAs)).

Most solutions found in the literature typically focus on centralized approaches, with a single management node making allocation decisions periodically [35] [23] [32]. These approaches suffer from poor scalability, as the management node may become a performance bottleneck if the number of physical and virtual machines grows. Distributed dynamic virtual machine consolidation (DDVMC) [8] is a management strategy proposing a distributed algorithm in order to increase scalability by enabling physical host nodes as management entities in a hierarchical system architecture. Each physical host node in this architecture contributes to the dynamic virtual machine consolidation procedure by managing itself, avoiding performance degradation in an overloading situation as well as avoiding inefficient energy consumption in an underloading one. The former can be done by migrating one or more virtual machines away from itself to other physical host nodes, the latter can be done by migrating all virtual machines away and going into a sleeping mode. Such self-management algorithm involves three dynamic decision making tasks for each physical host node to tackle the dynamism of cloud environment raised by variable and unexpected applications workload: 1) when must a host be considered as overloaded, 2) which virtual machine must be selected to migrate away from an overloaded host, and

3) when must a host be considered as underloaded. In addition, there is a global manager in this architecture is responsible for optimizing the placement of virtual machines that must be migrated away from overloaded and underloaded physical host nodes.

The most significant challenge in such strategy is how to make the aforementioned decisions optimally in a real time manner to minimize both energy consumption and SLA violations inside the system. In addition, the optimal solution need to assure scalability due to increasing the number of hosts in data center.

4.2 A Self-Adaptive Host Overloading Detection Schema

In this section, we concentrate on host overloading detection sub-problem of dynamic virtual machine (VM) consolidation procedure and propose a reinforcement learning algorithm to tackle this problem. With respect to the dynamic nature of cloud environment which is raised by dynamic behaviour of the applications workload, host overloading detection can be considered as a dynamic decision making task. In a threshold-based approach, a host overloading detection algorithm consider a physical host node as overloaded if it's CPU utilization exceeds a threshold value. As a result of being detected as an overloaded host, one or more virtual machines (VMs) must be migrated until the CPU utilization falls below the threshold value. Defining the threshold value is a big challenge since a low threshold value can be against more resource overbooking resulting inefficient energy consumption in data center and in contrast, a high threshold value can increase the probability of performance degradation due to experiencing 100% CPU utilization in the host node. Note that, if a host is experiencing 100% utilization, the performance of the applications (running on the host) is bounded by the host capacity, therefore, virtual machines (VMs) are not being provided with the required performance level. To make the energy-performance trade-off optimal, we need to make optimal decision regarding the value of the utilization threshold dynamically. To this end, the DFQL method, previously explained in the background chapter (see Section 2), is employed to learn how to set the threshold values (for each physical host node independently) dynamically towards energy-performance trade-off optimization through interaction with dynamic characteristics of each physical host node's environment in data center.

4.2.1 Model Description

Our system model is an IaaS environment represented by a large scale data center including N physical host nodes. Each node is characterized by its CPU performance, disk storage, amount of RAM and network bandwidth. As Figure 4.1 shows, the software model represents a hierarchical system management architecture in which there is a global manager resides on a master node, which is responsible for collecting information from the local managers to manage the allocation of Virtual Machines (VMs) by issuing Virtual Machine (VM) migration commands and changing the power states of the nodes. Local managers reside on each physical node as a module of the Virtual Machine Monitor (VMM). They monitor the node CPU utilization, resize Virtual Machines (VMs) according

to their resource demands and decide when and which Virtual Machines (VMs) have to be migrated from their host node. The DFQL agent, as a decision maker, resides on the master node, using the information collected by the global manager to perceive current state of each host node periodically. Then it uses the global manager's commanding service to inform local managers the threshold values as decisions/actions. Figure 4.2 illustrates a closer view of communication between DFQL and each host node. At each time step t the global manager collects the current state of each host node and keeps them in an input queue. Data received from each host node in the queue is considered as input data for DFQL. Note that in our implementation we neglected the queuing delay. With regards to the current state of each host node, DFQL determines a threshold value and sends it to the corresponding local manger. One time step later, as a consequence of using the threshold values in the host overloading detection process, the global manager collects reinforcement signals from each local manager.

Energy Model

In our energy model, the power consumption of each physical host node is a linear function of the CPU utilization. We use a simple linear approximation due to Fan et al. [20]:

$$P = P\left(U_{CPU}\right) = P_{idle} + \left(P_{peak} - P_{idle}\right) U_{CPU}. \tag{4.1}$$

Since P_{idle} is always a fraction of P_{peak}, (4.1) can change as follows:

$$P = P\left(U_{CPU}\right) = k\, P_{peak} + P_{peak}\left(1 - k\right) U_{CPU}. \tag{4.2}$$

In our experimental studies the value of k has been set to 0.7 [20]. Furthermore, P_{peak} is set to 250 W, which is a usual value for modern computing servers. Consequently, the total energy consumption by a server during a time slice can be defined as follows:

$$EC = \int_{t_1}^{t_2} P\left(U_{CPU}\left(t\right)\right) dt. \tag{4.3}$$

SLA Violation Metric

Service Level Agreements (SLAs) define some quality of service that cloud users expect to receive from the cloud for their deployed VMs in an IaaS environment, such as response times being below a specified threshold. Experimental studies [92] show that live migration has a negative impact on the performance of applications running in a VM during migrations. They have found that performance degradation and downtime is depended on the application's behavior, for example, how many memory pages the application updates during its execution. However, for the class of applications with variable workloads, such as web-applications, the average performance degradation including the downtime can be estimated as approximately 10% of the CPU utilization. On the other hand, the performance of applications running on a VM can be negatively

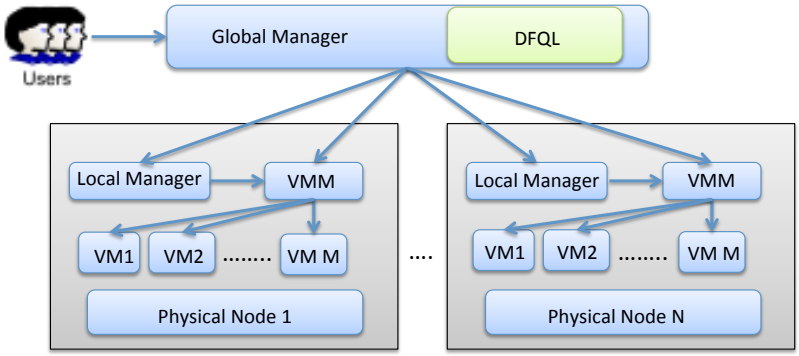


Figure 4.1: A system model for dynamic vm consolidation

affected if the physical node hosting the VM experiences a CPU utilization of 100%. Both cases, migration and 100% CPU utilization, result in degrading application performance in our system model.

In our experiment studies we employed a generic and application-independent metric for SLA violations due to Beloglazov and Buyya [8]. They proposed two metrics to quantify SLA violations in cloud data centers based on the aforementioned performance degradation definitions: 1) SLA violation Time per Active Host (SLATAH): The percentage of time, during which the host nodes have experienced 100% of the CPU utilization. The reasoning behind this metric relying on this fact that in 100% utilization, the performance of the application running on the VMs is bounded by the host capacity and therefore they can not be provided with the required performance level and 2) Performance Degradation due to Migration (PDM): The overall performance degradation caused by migration process.

$$SLATAH = \frac{1}{N}\sum_{j=1}^{N}\frac{T_{s_i}}{T_{a_i}}, \tag{4.4}$$

$$PDM = \frac{1}{M}\sum_{j=1}^{M}\frac{C_{d_j}}{C_{r_j}} \tag{4.5}$$

Here, N is the number of physical host nodes in the data center; T_{s_i} is the total time, in which physical host node i has experienced utilization of 100%, and T_{a_i} is the total time, in which physical host node i has served virtual machines; M is the number of virtual machines in data center, C_{d_j} is the performance degradation estimation raised by migrations for the VM j (As stated above, in our study 10% of the CPU utilization in MIPS) and C_{r_j} is the total CPU capacity requested by VM j during its lifetime. We are aware that other system resources such as memory, storage, and network bandwidth should also be taken into account in the definition of QoS requirements. Since the CPU is the main resource which is usually oversubscribed by cloud providers, in this thesis we only consider the CPU utilization of the physical host nodes.

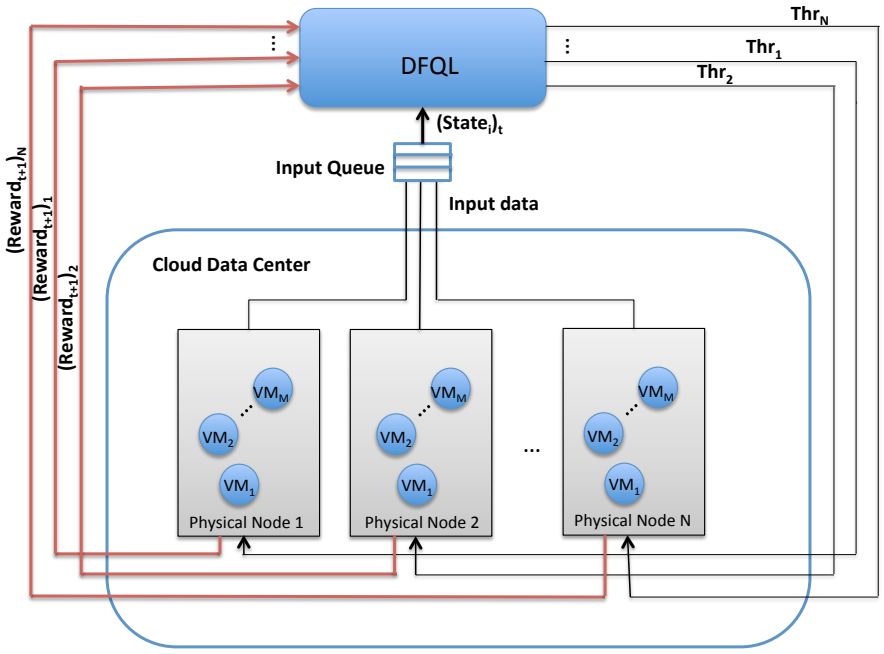


Figure 4.2: DFQL host overloading detection

4.2.2 Formulating Host Overloading Detection in Dynamic Fuzzy Q-Learning

The components of DFQL agent residing on the master node are described in the following:

- **State:** In each physical host node there exist two parameters changing over time and describe dynamic and uncertain behaviour of each physical host node. The first one is the total CPU utilization CU changing due to unexpected behavior of applications workload running on the virtual machines. The second one is the number of virtual machines $NumVm$ residing on the host node changing due to migration of virtual machines to/from the physical host node. The combination of the CPU utilization and the number of virtual machines in each time step can describe the current state of each physical host node. Therefore, the input state vector to the DFQL is defined as:

$$x = [CU, NumVM]. \tag{4.6}$$

- **Actions:** Each element of the action set denotes the possible threshold value for host node CPU utilization, it can be defined as follow:

$$o = [thr_1, thr_2, thr_3, \ldots, thr_n] \tag{4.7}$$

thr_i is a discrete value denoting a particular CPU threshold.

- **Reward Function:** A reward function defines the goal in a reinforcement learning problem. Roughly speaking, it maps each perceived state-action pair of the environment to a single number, a reward, indicating the intrinsic desirability of that state-action. Note that, whereas a reward function indicates what is good in an immediate sense, a q-value function specifies what is good in the long run (see Section 2, for more details). Therefore, we seek actions that result in states of highest q-value, not highest reward, because these actions obtain the greatest amount of reward for us over the long run in our system.

  The reward function in our system returns a value determining a success rate, here minimizing both performance degradation and energy consumption in data center. In fact, our DFQL agent receives a global reward as a consequence of jointly performing independent actions (setting threshold values) in the environment. In Section 4.2.1 we presented three metrics to measure the SLA violation (SLATAH and PDM) and the energy consumption (EC) in data center. A standard combined metric [8] to capture both energy consumption and SLA violation in data center can be defined as a product combination of the SLATAH, PDM and EC (called Energy SLA Violation (ESV)) as follow:

$$ESV = SLATAH \times PDM \times EC. \tag{4.8}$$

  Consequently, the global reward function can be defined as the inverse of the ESV value as follow:

$$r = 1/ESV. \tag{4.9}$$

  Note that, once an agent takes an action in a given state, it will receive a reward as a consequence of the chosen action in the next time step. The time step is a period of time (epoch/iteration in machine learning terminology) which the system needs to monitor itself so as to provide the reward. Therefore, it is worth mentioning that the PDM, SLATAH and EC, for providing rewards, must be captured during each period iteratively.

  PDM has an important role in our reward function, imagine the reward function without PDM, one possible valid solution could be decreasing the threshold value to have less SLATAH (even close to zero) and more energy consumption. This solution is completely against our objective to have energy efficient data center. PDM is a variable which is able to control this situation as it is opposite of setting the threshold value very low and also very high.

4.2.3 Experimental Setup

In order to evaluate the performance of our proposed approach, we carried out simulation runs using the CloudSim toolkit [14]. We simulated a data center comprising of 800 physical host nodes and used a real life data of workload provided by the CoMon project

Data	Number of VMs	Mean	Standard deviation	Quartile 1	Median	Quartile 2
Workload 1	1052	12.31%	17.09%	2%	6%	15%
Workload 2	1516	9.26%	12.78%	2%	5%	12%
Workload 3	1078	10.56%	14.14%	2%	6%	14%

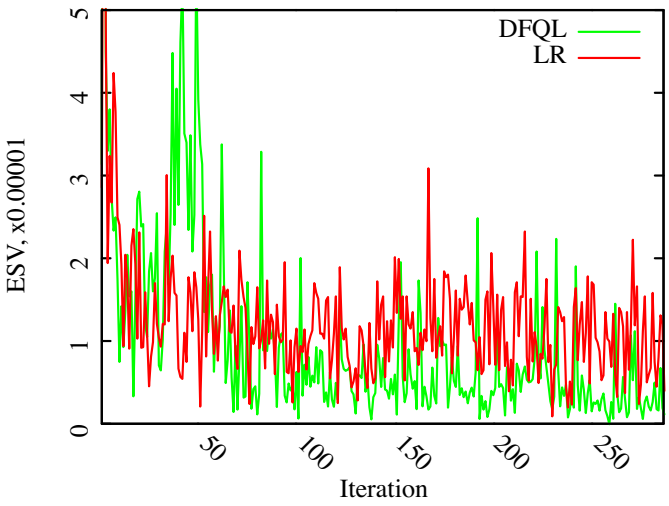
Table 4.1: Workload data characteristics (CPU utilization)

monitoring the infrastructure of PlanetLab [68] and collected on the CPU utilization from more than a thousand VMs from servers located at more than 500 places around the world [8]. The interval of CPU utilization measurement is 300 seconds. The characteristics of the workload data for three different days are shown in Table 4.1. Virtual machine types correspond to Amazon's EC2 instances, including small, medium, and extra large. Initialization of VMs is done based on the resource requirements which defined by the type of VMs. However, in order to have the opportunities for dynamic consolidation, the VMs utilize less resource according to the workload data during their lifetime. During the simulation, a workload trace of a virtual machine from a corresponding day is assigned randomly to a virtual machine in data center.

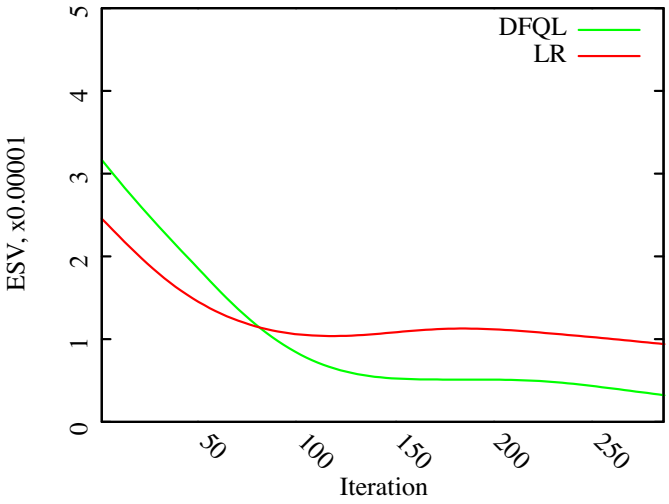
In our dynamic VM consolidation procedure, for all of the experiments we used Minimum Migration Time (MMT) policy [8] for VM selection. Based on this policy a VM is migrated if it requires the minimum time to complete a migration relatively to the other VMs allocated to the host node. For VM placement, we utilized a Best Fit Decreasing algorithm [7] and for host underloading detection we employed a simple strategy that selects the host with the minimum utilization compared to the other host nodes in data center. In the DFQL setup, the discount factor γ and the learning rate α have been empirically set to 0.5 and 0.1 respectively. The q-table has been initialized by zero. We applied Gaussian shaped membership functions for the input state variables (here, $NumVM$ and CU). The number of membership functions for each input element has been set to three experimentally. The Gaussian parameters (μ and $\sigma 2$) have been initialized randomly and they will be tuned during the learning as a result of running a clustering algorithm periodically (here, a Fuzzy c-means algorithm [10]).

4.2.4 Experimental Results

In this section, we compare the result of the DFQL for the problem of host overloading detection in dynamic virtual machine (VM) consolidation with three heuristic algorithms which have been proposed by Beloglazov and Buyya [8]. Interquartile Range (IQR) and Median Absolute Deviation (MAD) [8] are two proposed methods for auto adjustment of the CPU utilization threshed based on statistical analysis of historical data collected during the lifetime of VMs. The third method using a Local Regression (LR) [8] to fit a trend polynomial to the last k observations of the CPU utilization to estimate the next observation. Then they use an inequality to make decision about whether the host node is overloaded or not. In comparison with the DFQL method, none of these methods have



(a)



(b)

Figure 4.3: Comparing ESV value over time

memory to consider the consequence of choosing an action in a given state for the future decision makings. In addition, they don't have the ability of estimating the consequence of a chosen action in long term.

In the first experiment, we compare DFQL with Local Regression (LR) algorithm. In order to highlight the learning ability of the DFQL algorithm, we quantified the total ESV value for both DFQL and LR in the simulated data center in each 300-seconds (DFQL epoch). Figure 4.3(a) shows this comparison and for the sake of clarity a smoothed version

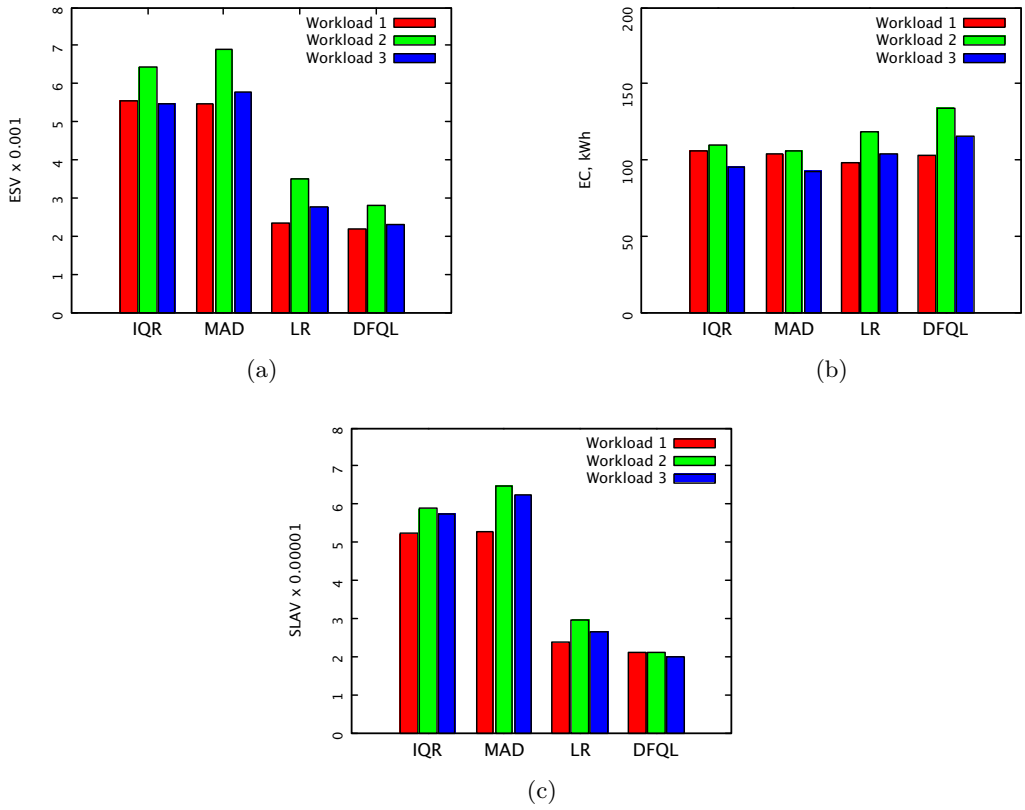
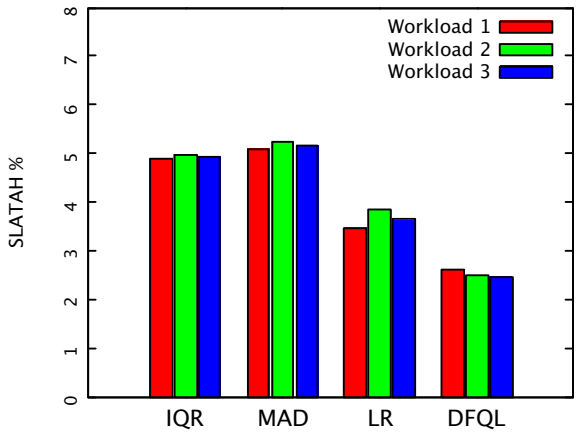
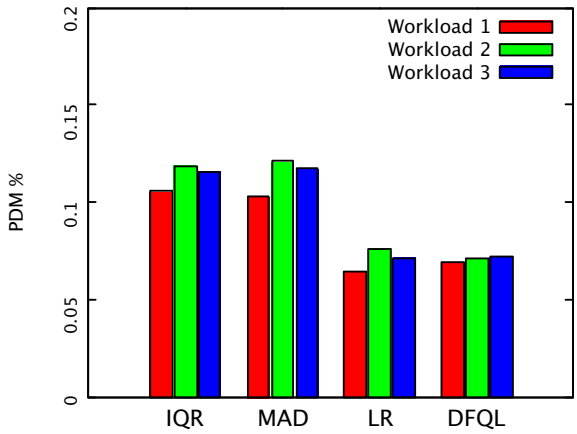


Figure 4.4: Comparison of ESV, SLAV and energy consumption.

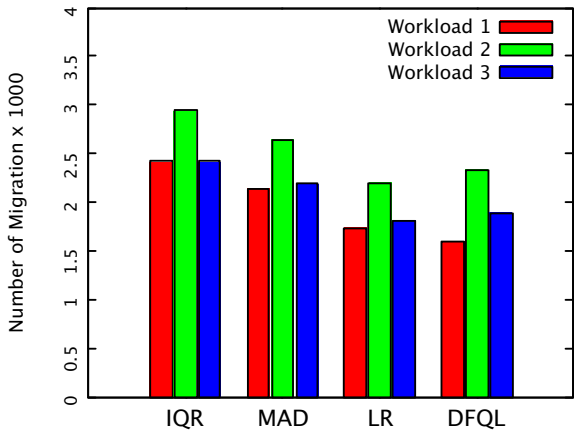
of the curve using a cubic splines function has been depicted in Figure 4.3(b). As the figures illustrate, thanks to the learning ability, the DFQL shows an improvement to act in a way to obtain more reward over time (less ESV value). In the next experiment, we compared DFQL with MAD, IQR and LR in total ESV value, total Energy Consumption (EC), total SLA Violation ($SLAV = PDM \times SLATAH$), total Performance Degradation Due to Migration (PDM), total SLA violation Time per Active Host (SLATAH) and number of migrations obtained for three different workloads (see Table 4.1). As Figure 4.4(a) shows, DFQL outperformed other methods in total ESV value by being a bit less energy efficient (higher total EC, see Figure 4.4(b)) but gaining much more performance in terms of reducing the level of SLA violation in data center (lower total SLAV, see Figure 4.4(c)). Finally Figure 4.5 shows SLATAH, PDM and the number of migration obtained for each algorithm in the simulated data center.



(a)



(b)



(c)

Figure 4.5: Comparison of SLA violation metrics and number of migrations

4.3 A Self-Adaptive Virtual Machine Selection Schema

When a host is overloaded, the dynamic VM consolidation procedure calls the VM selection task in order to make it resolved by migrating VMs. The VM selection task must consider both energy and performance in the process of choosing VMs to migrate away from an overloaded host node. In this thesis, we propose a utilization-based VM selection criterion to decide which virtual machine must be migrated away when a host node is overloaded. Based on this criterion, VMs are selected to migrate according to their contributions to the CPU utilization of their physical host node. Maximum Utilization ($MaxU$) and Minimum Utilization ($MinU$) as two general sub-criteria can be defined here. Based on the $MaxU$ criterion, a VM is selected if it has the highest CPU utilization in comparison to all others. In contrast, based on the $MinU$ criterion, a VM is selected if it has the lowest CPU utilization in comparison to all others. Let's imagine an overloaded physical host node, if the VM selection uses the Maximum Utilization ($MaxU$) criterion to select virtual machines, the probability of performance degradation due to overloading will be decreased (as the busiest VM is always sent away), but instead it will increase its own contribution to inefficient energy consumption inside the data center. In contrast, if the host node uses the Minimum Utilization ($MinU$) criterion, it will increase the probability of performance degradation due to overloading, but it will decrease its own contribution to inefficient energy consumption inside the data center. Apart from it, if the host node uses the Maximum Utilization ($MaxU$) criterion, it has more chance to bring utilization below the threshold with a low number of migrations which has a positive impact on the performance degradation due to migration, while using this criterion might also has a negative impact on the performance degradation as the virtual machines have the highest CPU utilization are always migrated (Note that, the performance degradation due to migration is a function of the number of migrated virtual machines and their corresponding CPU utilization, see equation 4.5). On the contrary, using the Minimum Utilization ($MinU$) criterion yields an opposite result on the performance degradation due to migration. To have better insight, we carried out a three days simulation of a cloud data center, where aforementioned VM selection criteria have been used in the VM selection process of distributed dynamic virtual machine consolidation (DDVMC). As Table 4.2 shown, $MaxU$ achieved a better result in performance degradation due to overloading (SLATAH, less is better, see Section 4.2.1) than the $MinU$ criterion. In contrast, it increased the performance degradation due to migration (PDM, less is better). Apart from them, it showed a more energy consumption (EC) in comparison with $MinU$. Therefore, using one fixed criterion for selecting VMs might some times lead to satisfactory results, but in other situations to poor results. In this section, we consider the VM selection task as a Dynamic Decision-Making (DDM) task and model it using Fuzzy Q-Learning (FQL). We show how FQL, previously explained in the background chapter (see Section 2), is able to integrate multiple VM selection criteria to benefit from all advantages and possible synergetic contributions of them in long-term learning. In fact, our approach learns how to find an optimal strategy to applying multiple criteria for selecting VMs during a dynamic VM consolidation procedure towards improving the

Criterion	PDM	SLATAH	EC (kWh)	Num. Migrations
MinU	1.26	4.22	257.49	94669
MaxU	1.31	3.92	265.87	29939

Table 4.2: Comparison between MinU and MaxU startegies

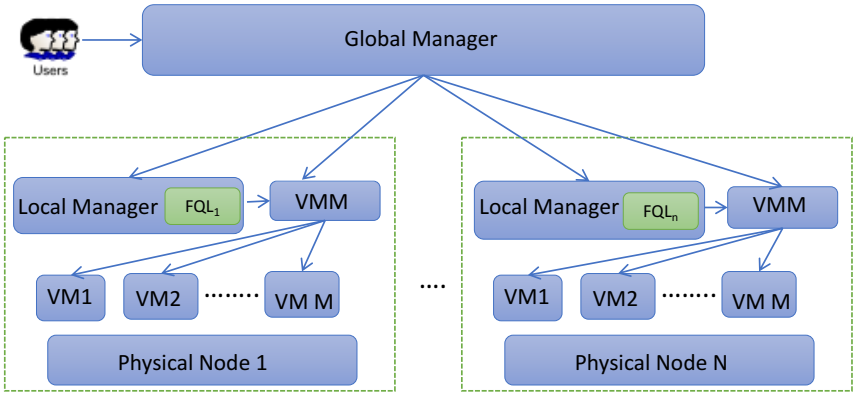


Figure 4.6: A system model for integrating VM selection strategies

energy-performance trade-off.

4.3.1 System Model

Our target system is an IaaS environment represented by a large scale data center including N physical nodes. As Figure 4.6 shows, the software model represents a hierarchical system management architecture in which there is a global manager resides on a master node is responsible for collecting information from the local managers to manage the allocation of Virtual Machines (VMs) by issuing Virtual Machine (VM) migration commands and changing the power states of the nodes. Local managers reside on each physical node as a module of the Virtual Machine Monitor (VMM). They monitor the node CPU utilization, resize Virtual Machines (VMs) according to their resource demands and decide when and which Virtual Machines (VMs) have to be migrated from their host node. In comparison with the previous system model for host overloading detection, our new proposed system model using a distributed reinforcement learning algorithm rather than a centralized one. In this system model, each physical host node is associated with an FQL agent. Each agent by having interaction with dynamic characteristics of its corresponding host node learns, which VM selection criterion must be chosen to optimize the global energy-performance trade-off in data center.

Energy Model and SLA Violation Metrics

We have used the same energy model previously discussed in Section 4.2.1, we also modified the previous metrics, SLATAH and PDM (Equation (4.4) and Equation (4.5)) in order to describe SLA violations inside each physical host node independently. These two metrics called 1) SLA Violation due to Overloading (SLAVO) , and 2) SLA Violation due to VM Migration (SLAVM), calculated as follows:

$$SLAVO_i = \frac{T_s}{T_a}, \quad 1 \leq i \leq N, \tag{4.10}$$

$$SLAVM_i = \frac{1}{M} \sum_{j=1}^{K_i} \frac{C_{d_j}}{C_{r_j}}, \quad 1 \leq i \leq N. \tag{4.11}$$

Here, N is the number of physical host nodes in data center; T_s is the total time, in which physical host node i has experienced utilization of 100%, and T_a is the total time, in which physical host node i has served virtual machines; M is the number of virtual machines in data center, K_i is the total number of VMs that have been migrated away from physical host node i, C_{d_j} is the performance degradation estimation raised by migrations for the VM j (in our study 10% of the CPU utilization in MIPS) and C_{r_j} is the total CPU capacity requested by VM j during residing inside physical host node i. Therefore, a metric for describing SLA violations inside each physical host node can be denoted by the product of SLAVO and SLAVM as follows:

$$SLAV_i = SLAVO_i \times SLAVM_i, \qquad 1 \leq i \leq N \tag{4.12}$$

This metric can encompass both performance degradation due to host overloading and VM migrations. Consequently, to represent the energy-performance trade-off we use the product combination of SLA Violation and Energy consumption (EC) that the authors of [8] denoted as the total Energy SLA violation (ESV):

$$ESV_i = SLAV_i \times EC_i, \qquad 1 \leq i \leq N \tag{4.13}$$

4.3.2 Formulating VM Selection Integration in Fuzzy Q-Learning

The components of the FQL agent as a module of each local manager is as follows:

- **The Reward Function:** The optimal long-term cumulative reward is the target of the FQL. The reward is a performance feedback on the resulted new VM selection criterion chosen from the action set. Therefore, the reward function can be defined as follows:
$$r_i = \frac{1}{ESV_i}, \qquad 1 \leq i \leq N \tag{4.14}$$
- **The Input State:** The FQL must be able to trace the physical host's behavior by merely observing the input states. Consequently, dynamic characteristics of the host node must be included into the input state, which have enough information

about the host's behavior. In addition they must be able to make inter-dependency between state and actions. The host CPU utilization and the number of VMs residing on each host were considered as the element of the input state of that host. These are time-varying factors caused by dynamic workloads and live migration. Therefore, the input state is defined as follows:

$$x_i = \{CU_i, NumVM_i\}, \qquad 1 \leq i \leq N \tag{4.15}$$

Here, CU_i is the CPU Utilization of the host i, and $NumVM$ is the number of VMs residing on the host i in each time-step.

- **The Action:** Each element of the action set denotes a VM selection criterion. Therefore, it can be defined as follows:

$$o = \{Criterion_1, Criterion_2, ..., Criterion_n\} \tag{4.16}$$

When the VM selection task is called for a host, the FQL agent associated with that host observes the current state S_{t_i} and collects information about the input state x_{t_i} (including the current CPU utilization and the number of VMs) and immediately choses a VM selection criterion as an action from the action set o in order to make a decision. One time-step later, the FQL agent receives a numerical reward, $r_{i_{t+1}}$ and finds itself in a new state S_{t+1}. In a trial and error interaction, finally the FQL agent learns how to map the states to the actions in order to maximize the discounted sum of rewards. In other words, the FQL agent learns the best sequence of VM selection criteria, corresponding to the states observed during its lifetime.

4.3.3 Experimental Setup

In order to evaluate the performance of our proposed approach, we carried out simulation runs using the CloudSim toolkit [14]. We simulated a data center comprising of 800 physical host nodes and used a real life data of workload provided by the CoMon project monitoring the infrastructure of PlanetLab [68] and collected on the CPU utilization from more than a thousand VMs from servers located at more than 500 places around the world [8]. The interval of CPU utilization measurement is 300 seconds. The characteristics of the workload data for three different days are shown in Table 4.1. Virtual machine types correspond to Amazon's EC2 instances, including small, medium, and extra large. Initialization of VMs is done based on the resource requirements which defined by the type of VMs. However, in order to have the opportunities for dynamic consolidation, the VMs utilize less resource according to the workload data during their lifetime. During the simulation, a workload trace of a virtual machine from a corresponding day is randomly assigned to a virtual machine in data center. In our dynamic VM consolidation procedure, for all of the experiments we used a static threshold based algorithm [8] to detect host overloading detection, a Best Fit Decreasing (BFD) algorithm [7] for VM placement and a simple strategy for host overloading detection which always selects the host node with the minimum utilization compared to the other host nodes in data center.

In the FQL setup, the discount factor γ and the learning rate α have been empirically set to 0.5 and 0.1 respectively. The q-table has been initialized randomly. We used three fuzzy sets (small, medium and large) with Gaussian membership functions for each input set element in the Fuzzy Inference System (FIS). The elements of the action set are:

$$o = \{MINU, MAXU\} \tag{4.17}$$

MAXU (Maximum Utilization) means that the VM selection task selects VMs having the highest utilization. *MINU (Minimum Utilization)* means that the VM selection task selects VMs, having the lowest utilization. The FQL iteration or time-step for perceiving next state and receiving the reward of the previous state has been set to 300 seconds.

4.3.4 Experimental Results

In the first experiment we compare the results of the dynamic VM consolidation procedure when the selection of VMs is based on the strategy has been chosen locally by each FQL module in each physical host node in the data center, with the same procedure when the VM selection strategy has been chosen randomly. Both strategies integrate MaxU and MinU criteria together with one fundamental difference, the FQL strategy benefits from a learning procedure to choose the criteria intelligently but the random strategy employs them randomly without any knowledge. In this comparison we have measured the total ESV value in the data center $\left(\sum_{i=1}^{N} ESV^i\right)$ per FQL iteration. This experiment has been evaluated with the same workload for both procedures. In addition, the result of the procedure with the random strategy is an average of five independent simulations. Since host overloading and underloading, as well as VM placement policies are the same in both procedures, the curves depicted in Figure 4.7 show us the effect of the two different strategies described above. As Figure 4.7 shows, the FQL strategy in the preliminary iterations acted without any learning knowledge. Therefore, it is natural that its result is very close to the random strategy or even worse than it. But after a while, the FQL strategy following a trial and error interaction learns how to find a policy (mapping states to actions), to maximize the discounted sum of rewards obtained. Indeed, this policy is a sequence of criteria where decision making for selecting VM based on them leads to the maximization of the reciprocal of the Energy SLA Violation (ESV) value discussed in Section 4.3.2.

In the next experiment we repeat the previous experiment, with this difference that we compare the result of the FQL strategy with fixed criterion (either MinU or MaxU) for making decisions (see Figure 4.8). It shows how FQL learns the appropriate policy to choose VM selection criteria, to utilize the synergetic contributions of them to effectively decrease the ESV value during the simulation.

In the third experiment we compared all metrics involved to represent our energy-performance tradeoff when the VM selection task uses the FQL strategy with ones uses minimum utilization and maximum utilization as a fixed criterion. This experiment was evaluated with three different workloads on three different days. Figure 4.9(a) and Figure 4.9(b) show the total value of the energy consumption and the total value of

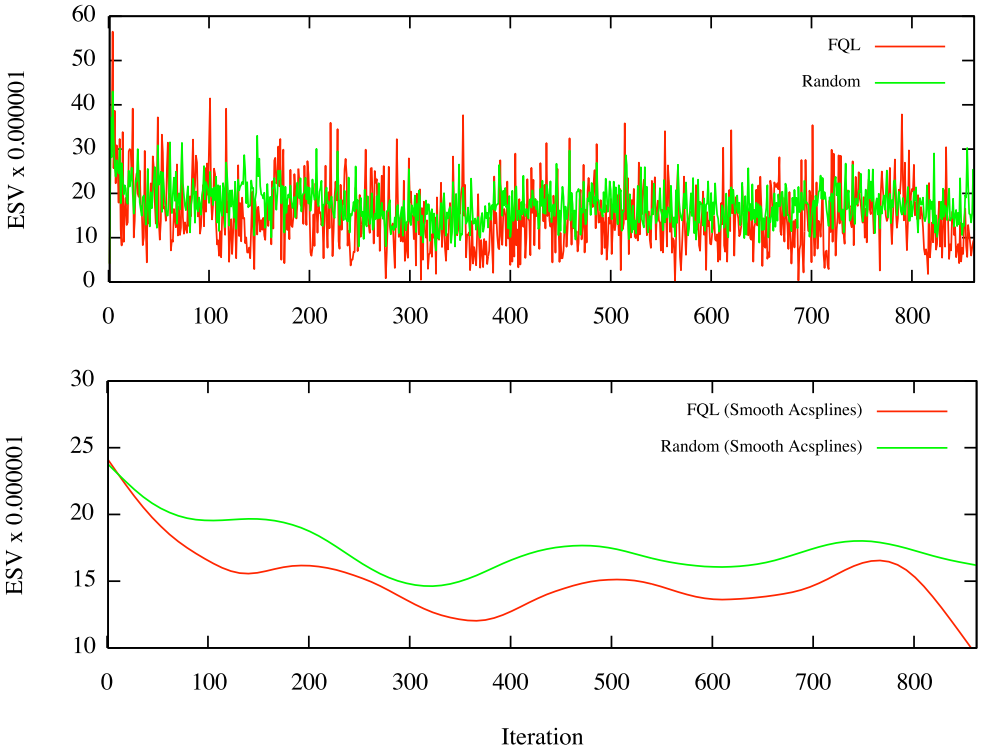


Figure 4.7: Comparison between the FQL strategy and the random strategy

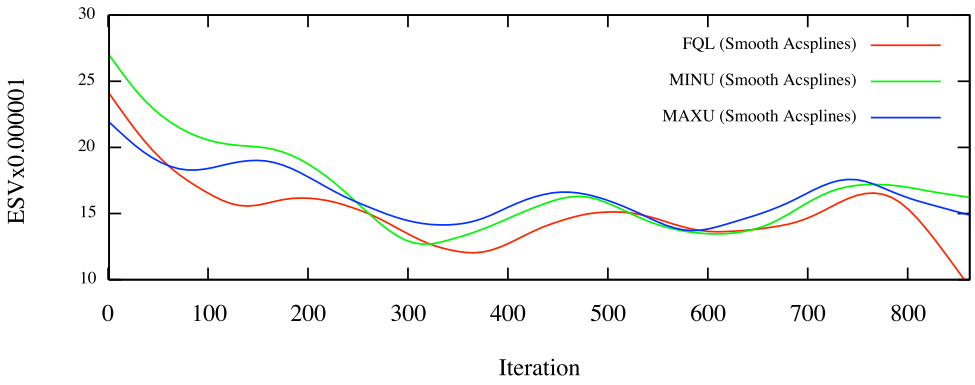
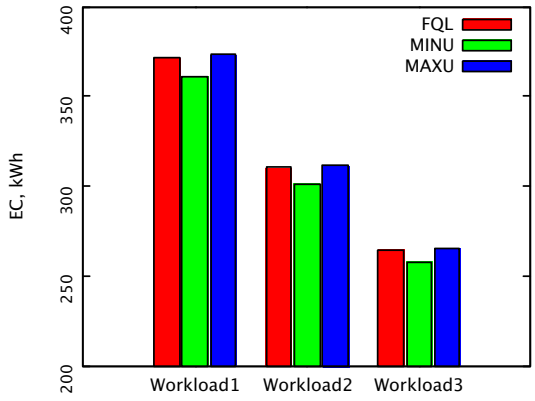
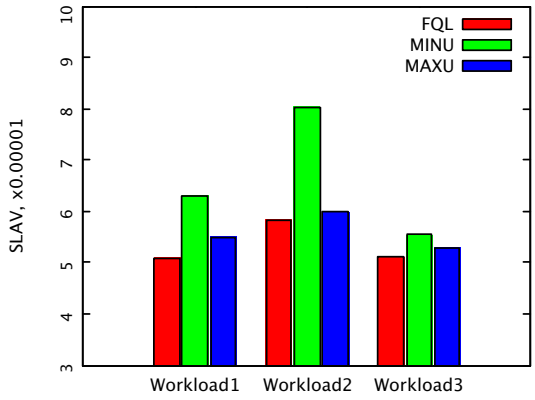


Figure 4.8: Comparison between VM selection using the FQL strategy and VM selection using fixed criteria

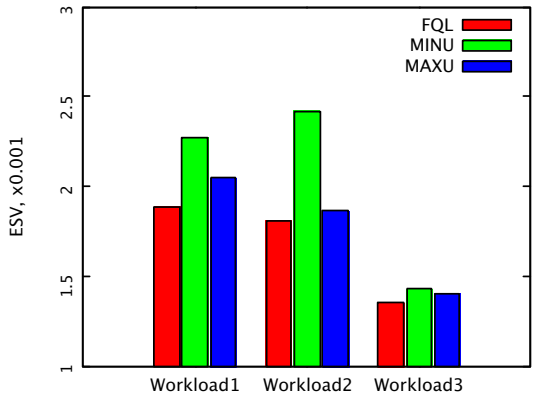
SLA violations respectively, Figure 4.9(c) illustrates ESV as a tread-off value between energy and SLA violations. These figures show that the FQL strategy has learned how to find a sequence of applying MAXU and MINU criteria to select VMs during dynamic VM consolidation procedure in order to keep balance between energy consumption and



(a)



(b)



(c)

Figure 4.9: Energy SLA violations

SLA violation and thus improves the ESV.

Figure 4.10 shows the SLA violation metrics (SLAVM and SLAVO) and the number of migrations during simulation. It is clear to see that how the FQL applies different criteria to control number of migrations and SLA violation metrics.

4.4 A Multi-Agent System Architecture

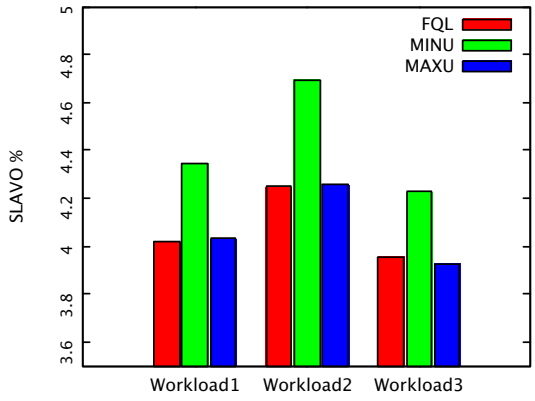
In the previous sections we first proposed a centralized Fuzzy Q-learning (FQL) and then a distributed Fuzzy Q-learning (FQL) approach to make the host overloading detection and the VM selection process self-adaptive and self-learner for distributed dynamic virtual machine consolidation (DDVMC) [8] in cloud data center. In this section, we combine these two approaches together and propose a comprehensive solution to manage physical host nodes in the process of distributed dynamic virtual machine consolidation (DDVMC). To this end, we propose a cooperative multi-agent reinforcement learning paradigm, previously explained in the background chapter (see Section 2), to solve this problem in a distributed manner. In our system model, a data center is considered as a multi-agent learning environment in which each learning agent represents a decision making engine inside each physical host node. In this environment, all the agents are capable of cooperating with each other, such that the global goal of the system (energy-performance tradeoff optimization) can be achieved by locally acting. On the other hand, the cooperation between learning agents can provide a further benefit in our system by speeding up the learning process and consequently increasing the decision-making quality.

4.4.1 System Model

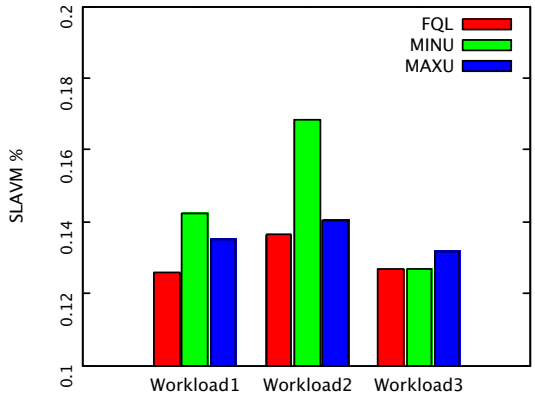
Our system model is an IaaS environment, represented by a large-scale data center including N physical host nodes. Each host node in the data center is characterized by its CPU performance, its amount of RAM, disk storage and network bandwidth. Users are able to submit virtual machines provisioning requests based on requirements to these characteristics. The software architecture of the system is hierarchical, including two layers (Figure 4.11). In the second layer, Local Managers (LMs) reside on physical host nodes as modules of the respective virtual machine monitors. LMs are responsible for monitoring the host node's CPU utilization and resizing virtual machines according to their resource demands. In addition, LMs are responsible for managing underloading and overloading situations. To this end, they were associated with intelligent agents which operate as decision making engines. At the first layer, there exists a computing unit as a Global Manager (GM) which is responsible for optimizing the placement of virtual machines that must be migrated away from overloaded and underloaded physical host nodes.

Energy Model and SlA Violation Metrics

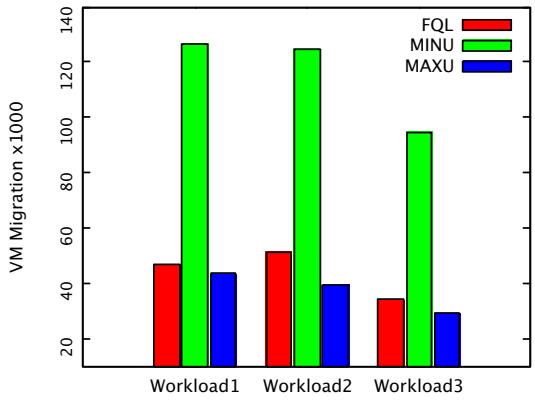
We have used the same energy model previously discussed in Section 4.2.1, we also modified the previous metrics (PDM and SLATAH) in order to describe SLA violations inside each physical host node. These two metrics are 1) Performance Degradation due



(a)



(b)



(c)

Figure 4.10: SLA violation metrics and number of migrations

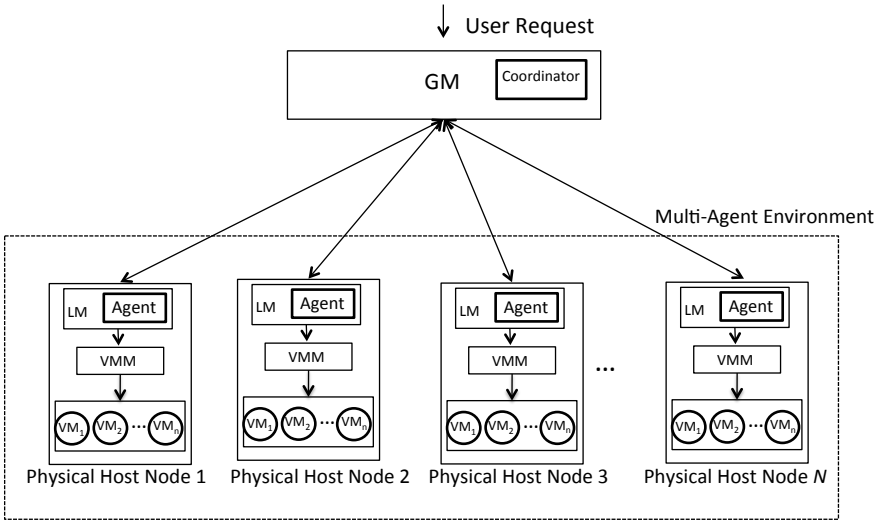


Figure 4.11: A multi-agent system model for distributed dynamic VM consolidation.

to Overloading (PDO), and 2) Performance Degradation due to VM Migration (PDM), calculated as follows:

$$PDO^i = \frac{T_s}{T_a}, \quad 1 \leq i \leq N, \tag{4.18}$$

$$PDM^i = \frac{1}{M} \sum_{j=1}^{K_i} \frac{C_{d_j}}{C_{r_j}}, \quad 1 \leq i \leq N. \tag{4.19}$$

Here, N is the number of physical host nodes in the data center; T_s is the total time, in which physical host node i has experienced utilization of 100%, and T_a is the total time, in which physical host node i has served virtual machines; M is the number of virtual machines in data center, K_i is the total number of VMs that have been migrated away from physical host node i, C_{d_j} is the performance degradation estimation raised by migrations for the VM j (in our study 10% of the CPU utilization in MIPS) and C_{r_j} is the total CPU capacity requested by VM j during residing inside physical host node i.

4.4.2 Distributed Dynamic VM Consolidation (DDVMC) Strategy

Our distributed dynamic VM consolidation strategy is split into two management algorithms: 1) a physical host node management algorithm running on each host node inside local managers, and 2) a VM placement optimization algorithm running at a centralized computing unit inside a global manager. The physical host node management algorithm manages overloading and underloading situations inside the respective host nodes. The former by migrating one or more VMs away from an overloaded host nodes to other nodes, the latter by migrating all virtual machines away from an underloaded host, and switching the node to sleep mode. At the central server, the global VM placement algorithm manages the placement of virtual machines that must be migrated away from

overloaded and underloaded host nodes. The physical host node management involves three dynamic decision making tasks: 1) when must a host be considered as overloaded, 2) which virtual machine must be selected to migrate away from an overloaded host, and 3) when must a host be considered as underloaded. In this section, we utilize a modified best fit decreasing algorithm [7] to optimize the central placement strategy, and concentrate just on the local physical host node management. In the following, we explain our proposed algorithm for managing physical host nodes and clarify why we need dynamic decision-making and how we can make optimal decisions in a distributed manner.

Physical Host Node Management Algorithm

The physical host node management algorithm in our proposed strategy employs three criteria to make decisions. In order to decide about when a host node must be considered as overloaded, we utilize a threshold-based criterion, i.e. once the node's CPU utilization CU exceeds a threshold value thr, i.e. $CU > thr$, the host node is considered as overloaded. In this situation, the host's virtual machines must be migrated away until the node's CPU utilization CU falls below the threshold value thr. In addition, we propose a utilization-based VM selection criterion to decide which virtual machine must be migrated when a host is overloaded. Based on this criterion, VMs are selected to migrate according to their contributions to the CPU utilization of their physical host node. Maximum Utilization $MaxU$ and Minimum Utilization $MinU$ as two general sub-criteria can be defined here. Based on the $MaxU$ criterion, a VM is selected if it has the highest CPU utilization in comparison to all others. In contrast, based on the $MinU$ criterion, a VM is selected if it has the lowest CPU utilization in comparison to all others. Decision making in underloading situations is done through a competitive strategy, whereby each node is considered as underloaded if it shows the minimum utilization amongst all nodes. In each physical host node there exist two parameters changing over time and making the physical host node as a dynamic and uncertain environment. The first one is the total CPU utilization CU of the host node changing due to unexpected behavior of applications workload running on the virtual machines. The second one is the number of virtual machines $NumVm$ residing on the host node changing due to migration of virtual machines to/from the physical host node.

Let us consider a physical host node making its own decisions based on the aforementioned criteria. From the energy efficiency point of view, the host node should accept as many VMs as possible, therefore, it should set the threshold value thr as high as possible. In this case, the probability of performance degradation due to overloading will be increased. On the other hand, if the host node decreases the threshold value thr, it will increase the probability of performance degradation due to migration (as it increases the number of migration) and also increases its own contribution to inefficient energy consumption inside the data center (as it keeps the CPU underutilized). Another challenge is to decide which virtual machines must be selected to be migrated. If the host node uses the maximum utilization $MaxU$ criterion to select virtual machines, the probability of performance degradation due to overloading will be decreased (as the

busiest VM is always sent away), but instead it will increase its own contribution to inefficient energy consumption inside the data center. In contrast, if the host node uses the minimum utilization criterion $MinU$, it will increase the probability of performance degradation due to overloading, but it will decrease its own contribution to inefficient energy consumption inside the data center. Apart from it, if the host node uses the maximum utilization $MaxU$ criterion, it has more chance to bring utilization below the threshold with a low number of migrations which has a positive impact on the performance degradation due to migration, while using this criterion might also has a negative impact on this performance degradation as the virtual machines have the highest CPU utilization are always migrated (Note that, the performance degradation due to migration is a function of the number of migrated virtual machines and their corresponding CPU utilization, see equation 4.5). On the contrary, using the minimum utilization criterion $MinU$ yields an opposite result on the performance degradation due to migration. Therefore deciding about the size of threshold value and the VM selection criterion is a non-trivial task. Dynamic behavior of the physical host nodes due to changes in the CPU utilization and the number of virtual machines might make it necessary to dynamically adapt the sizes of the threshold values and the VM selection criterion at run time.

In fact, each physical host node deals with a dynamic multi-objective optimization problem. To solve this problem, each physical host node needs a management system in which the dynamic decisions (the size of the threshold value and the VM selection criterion) can be made proactively. To this end, our intelligent agent resides in each physical host node (as shown in Figure 4.11) using the FQL algorithm, previously explained in the background Chapter 2, in a cooperative multi-agent setting to solve the problem of managing physical host nodes in a distributed fashion. In the following, the general algorithm for managing physical host nodes running at each local manager as well as for VM placement running at global manager in DDVMC are shown in Algorithm 4.1 and Algorithm 4.2 respectively.

Algorithm 4.1: General Algorithm for Managing Physical Host Node

```
[utilizationThreshold, vmSelectionCriterion]← learningAgent()
if amIOverloaded (utilizationThreshold) then
   vmsToMigrate ← vmSelection (vmSelectionCriterion)
   waitingForVmPlacement()
end if
if amIUnderloaded () then
   vmsToMigrate ← getVmList()
   waitingForVmPlacement()
end if
```

Algorithm 4.2: General Algorithm for VM placement

```
for host:hostsList do
    vmsListToMigrate.add ( getVmsToMigrate(host))
end for
migrationMap.add ( findNewPlacement (vmsListToMigrate))
vmsListToMigrate.clear
return
migrationMap
```

4.4.3 Cooperative Multi-agent Learning Algorithm

The goal of our system design is to optimize the energy-performance trade-off in data center as a global objective. Therefore, each single learning agent needs to have a global perspective of the system environment while making decisions locally (Note that, the agents can influence other agents' environment by their own actions). To this end, we propose a cooperative learning strategy (with the assumption that all the physical host nodes in data center are homogeneous) whereby the learning agents would be able to jointly solve the energy-performance tradeoff as an optimization problem. In addition, by using our cooperative strategy, the agents would be able to share their own learning knowledge with each other. As a consequence, a learning agent can speed up the time it takes to learn, since it does not have to visit every state and action in the given environment. This is especially true for host nodes sleeping for a long time, and thus not being able to learn by themselves. In a cooperative setting, they can use the learning knowledge of other host nodes when they wake up.

To enforce cooperation between learning agents we make use of a blackboard communication schema [99], which is simple to be implemented. It is a shared memory that can be accessed by every agent for reading and writing. Therefore, there is no direct communication between agents and the data flow between agents is redirected through the blackboard. In our proposed Cooperative FQL (CO-FQL) algorithm, the blackboard maintains a single q-table for all agents inside the coordinator unit (see Figure 4.11). Therefore, cooperation can be enforced by using a *global reward* comprising of the sum of rewards of all the individual learning agents, which are feeding a single shared q-table.

Fuzzy Q-Learning Components

The components of a FQL agent residing on each local manager are described in the following:

- **State:** As stated in before, the combination of the CPU utilization and the number of virtual machines can describe each state for decision making inside each physical host node. In addition, these two characteristics have enough information to represent a predictable physical host node behavior. Therefore, the input state vector to the FQL is defined as

$$x = [CU, NumVM]. \tag{4.20}$$

CU and $NumVM$ are the total CPU utilization of the physical host node, and the number of virtual machines residing on the physical host node respectively.

- **Actions:** Each element of the action set o denotes a pair of criteria: the first criterion is a utilization threshold value to detect overutilization and the second one is a VM selection criterion to choose a VM for migration. The action set is then defined as follows:

$$o = [\{thr_1, thr_2, thr_3, \ldots, thr_n\} \times \{MinU, MaxU\}] \tag{4.21}$$

thr_i is a discrete value denoting a particular CPU threshold. $MinU$ and $MaxU$ are the VM selection criteria based on minimum utilization and maximum utilization respectively.

- **Reward Function:** A reward function defines the goal in a reinforcement learning problem. Roughly speaking, it maps each perceived state-action pair of the environment to a single number, a reward, indicating the intrinsic desirability of that state-action. Note that, whereas a reward function indicates what is good in an immediate sense, a q-value function specifies what is good in the long run. Therefore, we seek actions that result in states of highest q-value, not highest reward, because these actions obtain the greatest amount of reward for us over the long run in our system. The reward function in our system returns a value determining a success rate, here minimizing both performance degradation and energy consumption for the agent as a consequence of taking an action in a given state. In Section 4.4.1 we presented three metrics to measure the SLA violation (PDO and PDM) and the energy consumption (EC) inside each physical host node. A standard combined metric [8] to capture both energy consumption and SLA violation can be defined as a product combination of the PDO, PDM and EC inside each physical host node (called Energy SLA Violation (ESV)) as follow:

$$ESV^i = PDO^i \times PDM^i \times EC^i. \tag{4.22}$$

Consequently, the reward function for each physical host node i can be defined as the inverse of its ESV, and the global reward can be expressed as:

$$r = 1/\sum_{i=1}^{N} ESV^i. \tag{4.23}$$

Note that, once an agent takes an action in a given state, it will receive a reward as a consequence of the chosen action in the next time step. The time step is a period of time (epoch/iteration in machine learning terminology) which the system needs to monitor itself so as to provide the reward. Therefore, it is worth mentioning that the PDM, PDO and EC, for providing rewards, must be captured during each period iteratively.

Data	Number of VMs	Mean	Standard deviation	Quartile 1	Median	Quartile 2
Workload 1	1052	12.31%	17.09%	2%	6%	15%
Workload 2	1463	12.39%	16.55%	2%	6%	17%
Workload 3	1033	10.43%	15.21%	2%	4%	12%

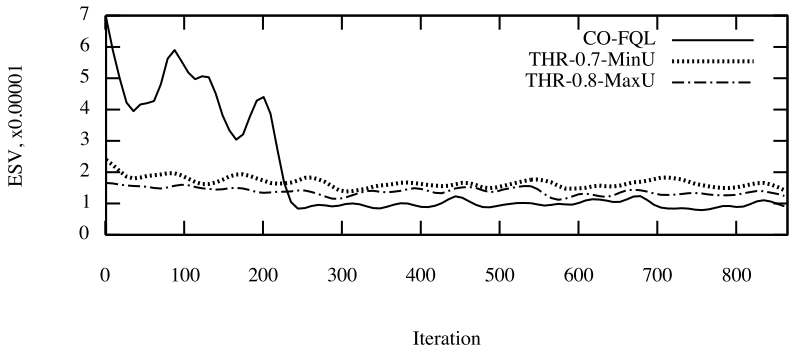
Table 4.3: Workload data characteristics (CPU utilization)

4.4.4 Simulation Setup

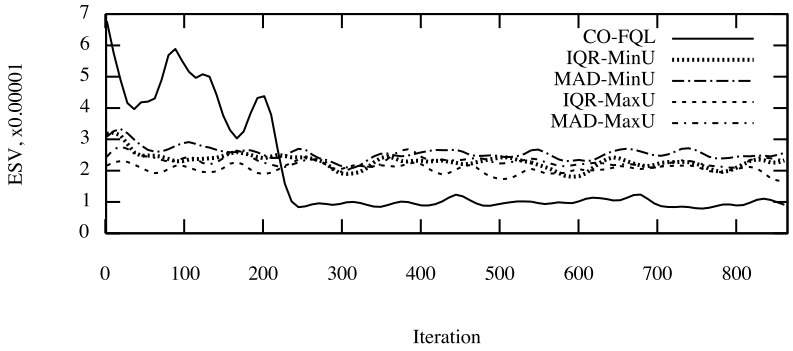
In order to evaluate the performance of our proposed approach, we carried out simulation runs using the CloudSim toolkit [14]. The layered architecture of CloudSim provides a platform for simulation of our multi-agent environment. We simulated a data center comprising of 800 physical host nodes and used a real life data of workload provided by the CoMon project monitoring the infrastructure of PlanetLab [68] and collected on the CPU utilization from more than a thousand VMs from servers located at more than 500 places around the world [8]. The interval of CPU utilization measurement is 300 seconds. The characteristics of the workload data for three different days are shown in Table 4.3. Virtual machine types correspond to Amazon's EC2 instances, including small, medium, and extra large. During the simulation, a workload trace of a virtual machine from a corresponding day is assigned randomly to a virtual machine in data center. In the CO-FQL setup, the discount factors γ and β as learning rate for all FQL agents have been empirically set to 0.5 and 0.1 respectively. Furthermore, the q-table has been initialized by zero. Three fuzzy labels (High, Medium, and Low) for each component of the state vector and the Gaussian fuzzy membership function has been used. The FQL iteration for perceiving the next state and receiving the reward for the previous action has been set to 300 seconds.

4.4.5 Experimental Results

The first result was obtained by running distributed dynamic VM consolidation in three-days simulation using Workload-1 (Table 4.3 shows the workload characteristics). Figure 4.12(a) illustrates the ESV value of the data center $\left(\sum_{i=1}^{N} ESV^i\right)$ at each iteration when the decisions for physical host node management have been made by the Cooperative FQL (CO-FQL) agents. In order to show the superiority of our proposed strategy, we compared its result with the best results obtained by using static thresholds and predefined VM selection criteria according to Table 4.4. For the sake of clarity, we show a smoothed version of the time series of the ESV value using cubic splines. The figure clearly shows the learning phase and the optimal behavior of the CO-FQL afterwards, yielding a much better result than the static approaches. In the same way we compare the CO-FQL algorithm with state-of-the-art adaptive threshold-based algorithms. The Interquartile Range (IQR) and Median Absolutes Deviation (MAD) [8] algorithms employ a statistical approach to make the CPU utilization threshold adaptive. We combined these two algorithms with the Minimum Utilization ($MinU$) and Maximum Utilization ($MaxU$) criteria for the VM



(a) In comparison with the best static approaches



(b) In comparison with the state-of-the-art adaptive approaches

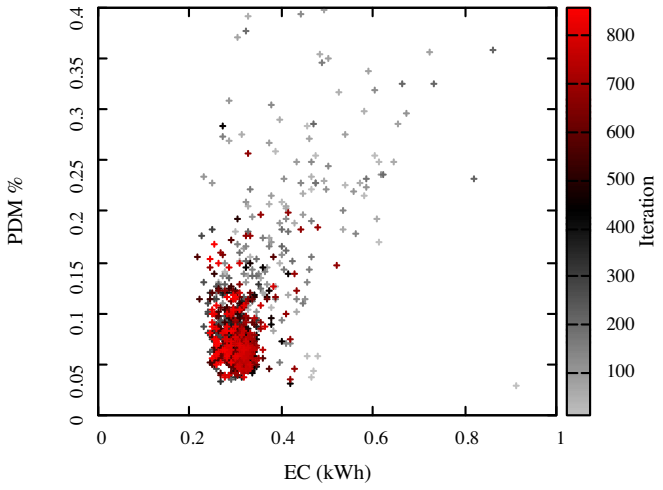
Figure 4.12: Energy SLA Violation (ESV) improvement by CO-FQL

	THR-0.9	THR-0.8	THR-0.7	THR-0.6	THR-0.5	THR-0.4	THR-0.3	THR-0.2	TH-0.1
MinU	0.017	0.025	**0.0143**	0.0144	0.021	0.034	0.061	0.147	0.254
MaxU	0.016	**0.011**	0.015	0.017	0.025	0.037	0.054	0.108	0.225

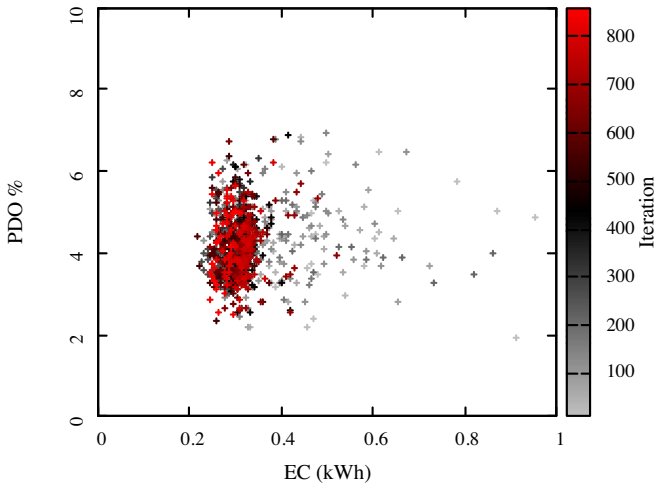
Table 4.4: Total Energy SLA Violation (ESV) in three-days simulation using Workload- 1. For example, setting threshold value to 0.9 (THR-0.9) along with using $MinU$ criterion for VM selection yields the total ESV value at 0.017

selection procedure, yielding four different physical host node management algorithms. Figure 4.12(b) shows how CO-FQL outperforms the aforementioned algorithms.

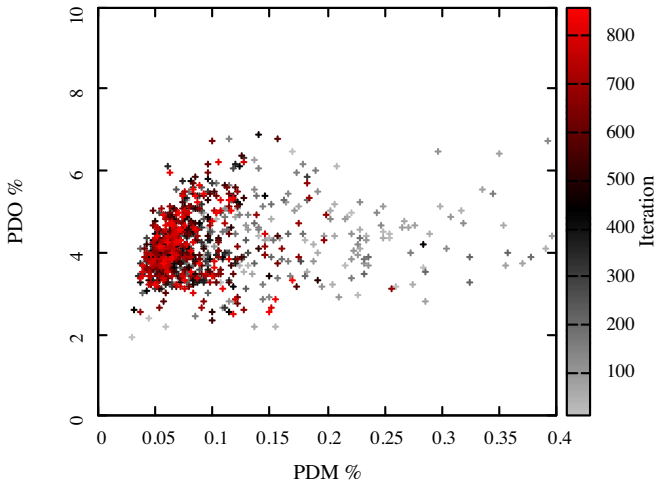
Due to (4.22), the ESV is a product combination of three objectives PDO, PDM and EC. In turn, CO-FQL tries to balance the influences of these objectives in order to minimize the total ESV inside data center. Figure 4.13 shows how CO-FQL progresses over time by showing the intermediate values of EC, PDO and PDM with respect to each other for subsequent iterations inside data center. The number of the respective iteration is depicted by the color of dots.



(a)

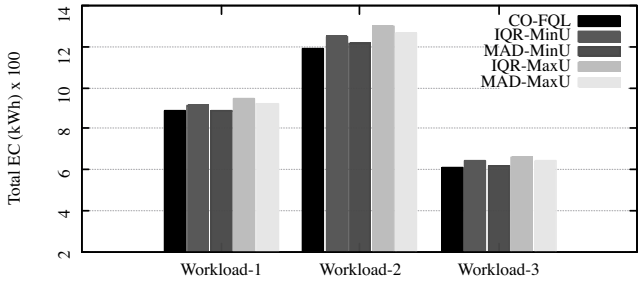


(b)

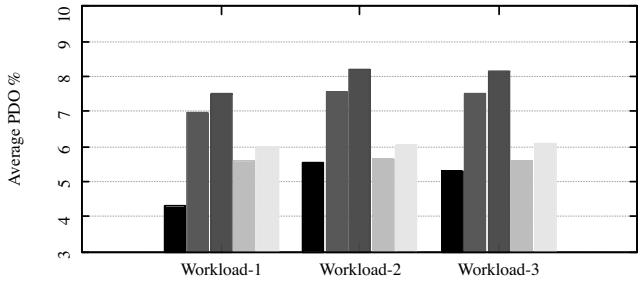


(c)

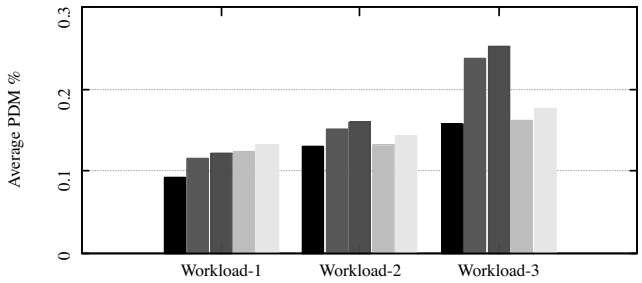
Figure 4.13: The progression of EC(kWh), PDM and PDO during a three-days simulation.



(a) Total Energy Consumption



(b) Average PDO



(c) Average PDM

Figure 4.14: Energy consumption and performance degradation obtained by CO-FQL wrt state-of-the-art algorithms.

In the next experiment we compared the total Energy Consumption (EC) and the average Performance Degradation due to Overloading (PDO) as well as average Performance Degradation due to Migration (PDM) obtained in the data center by using CO-FQL with the state-of-the-art adaptive threshold-based algorithms during ten-days simulation by three different workloads (see Table 4.3). Figure 4.14 shows that the CO-FQL algorithm outperforms other ones w.r.t. the total Energy Consumption, PDO and PDM. This experiment also shows that the performance of the CO-FQL is workload independent and the cost of learning has just a negligible effect on the performance of the algorithm in the long-term.

CHAPTER 5

Dynamic Virtual Machine Consolidation for Large Scale Cloud Data Centers

Dynamic virtual machine consolidation strategies refer to a number of resource management algorithms aiming at finding the right balance between energy consumption and SLA violations by using live migration techniques in virtualized cloud data centers. Most strategies found in the literature typically focus on centralized approaches, with a single management node responsible for VM placement. These approaches suffer from poor scalability, as the management node may become a performance bottleneck if the number of physical and virtual machines grows. In this chapter, we address research question II by presenting its corresponding contribution IV, introduced in Chapter 1. We first motivate the need of a fully decentralized dynamic virtual machine consolidation strategy in large scale cloud data centers at Section 4.1. Then, in the next two following sections (Section 5.3 and Section 5.4) we present our approaches to build a fully decentralized dynamic virtual machine consolidation strategy on top of an unstructured P2P network of physical host nodes.

5.1 Motivation

The main principle of dynamic VM consolidation strategies is to place virtual machines dynamically in the least number of physical host nodes while trying to fulfill the performance of the applications running on them. Most solutions found in the literature consider the problem of virtual machine placement as a multi-objective bin packing optimization problem and solve it periodically to keep the system optimal over time in terms of energy and performance. The objectives are usually decreasing energy consumption, decreasing the number of virtual machine migrations and increasing the satisfactory rate of the

applications running on the VMs in terms of requested resources. The most significant challenge in theses approaches is that they focus on centralized approaches, with a single management node. Therefore, they suffer from poor scalability, as the management node may become a performance bottleneck if the number of physical and virtual machines grows.

Distributed dynamic VM consolidation (DDVMC) [8] is a management strategy proposing a distributed algorithm in order to increase scalability by enabling physical host nodes as management entities in a hierarchical system architecture. Each physical host node in this architecture contributes to the dynamic virtual machine consolidation procedure by managing itself, avoiding performance degradation in an overloading situation which can be done by migrating one or more virtual machines away from itself to other physical host nodes. There is also a centralized manager in this architecture which is responsible for: (1) detecting underloaded physical host nodes and switching them to the sleep mode to avoid inefficient energy consumption and (2) placement of migrated virtual machines. Such distributed algorithm can alleviate the complexity of VM placement as the central node just optimizes the placement of VMs migrated away from underloaded and overloaded physical host nodes. However, the system still suffers from a single point of failure. Moreover, the central manager could still turn into a performance bottleneck if the number of physical host nodes dramatically grows in the respective data center. To address this problem, we propose a fully decentralized dynamic virtual machine consolidation strategy on top of a P2P network of physical host nodes using a gossip-based protocol, previously explained in the background chapter (see Section 2). In our proposed strategy, each physical host node is considered as an independent management entity (can join the data center as an additional computing resource or leave the data center due to failure anytime), is able to make all the decisions and take all the actions about its own situation at the local scope of neighborhoods. The randomness of system topology (as a result of using a gossip algorithm) facilitates the convergence of the management process towards efficiency in terms of energy and performance inside the data center.

We have evaluated our work through a cloud data center simulator that we developed in-house over PeerSim simulator tool [62]. PeerSim is a peer-to-peer network simulator written in Java, allowing the simulation of systems with varying cardinality. In our work, the evaluation focuses on the performance of our proposed strategy in terms of energy consumption, average CPU utilization, SLA violations due to overloading, number of virtual machine migrations and total sleep nodes inside the data center.

5.2 System Model

5.2.1 Model Description

Our system model is an Infrastructure as a Service (IaaS) environment, represented by a large-scale data center including N physical host nodes. Each host node in the data center is characterized by its CPU performance, its amount of RAM, disk storage and network

bandwidth. In our system model, there is no shared data structure or central controller. All physical host nodes are interconnected with high-speed LAN connection such as Gigabit Ethernet or Infiniband and the virtual machines can be migrated live between the physical host nodes. Each physical host node runs a controller module responsible for monitoring the host node's CPU utilization, and for resizing virtual machines according to their resource demands. In addition, it is responsible for managing underloading and overloading situation and also finding an optimal placement for its own migrated virtual machines.

5.2.2 Power Model

Instead of using an analytical model of power consumption by a server, in our simulation we utilize real data on power consumption captured from real physical host nodes [8].

5.2.3 SLA Violation Metrics

Experimental studies [92] show that live migration has a negative impact on the performance of applications running in a VM during migrations. On the other hand, the performance of applications running on a VM can be negatively affected if the physical node hosting that VM experiences a CPU utilization of 100%. Both cases, migration and 100% utilization, result in degrading an application's performance in our system model. In our experiment studies we employed generic and application-independent metrics for SLA violations. The first one is the percentage of total number of virtual machine migrations and the second one is the percentage of total time in which physical host nodes have experienced utilization of 100% [8]. In general, other system resources such as disk, memory, and network bandwidth should also be taken into account in the definition of quality of service requirements. Since the CPU is the main resource which is usually oversubscribed by cloud providers, in this thesis we only consider the CPU utilization of the system.

5.3 Proposed Dynamic VM Consolidation

Our dynamic VM consolidation strategy is split into three management algorithms: 1) overloading management, 2) underloading management, and 3) a virtual machine placement algorithm. The overloading and underloading management algorithms manage the respective host nodes to avoid local performance degradation in a an overloading situation, and global inefficient energy consumption in an underloading situation respectively. The former by migrating one or more VMs away from an overloaded host node to other nodes, the latter by migrating all virtual machines away from an underloaded host node, and switching the node to sleep mode. The VM placement algorithm finds the best possible physical host nodes to place migrated virtual machines due to overloading and underloading situations.

5.3.1 Overloading Management

In order to determine when a host is considered as overloaded, we utilize a single threshold-based strategy. If the CPU utilization of the host node exceeds the threshold value, it is considered as overloaded. In this situation a VM selection strategy selects one or more virtual machines to migrate, until the CPU utilization falls below the threshold value. In our work, we used a predefined threshold value and a VM selection strategy using a random policy to select virtual machines to migrate. The overloading management algorithm runs inside each physical host node locally and does not need to have any information from other host nodes in data center as input for its own decision making process.

5.3.2 Underloading Management

Underloading management can be done through a competitive strategy whereby a physical host node is considered as underloaded if it has a CPU utilization lower than the average CPU utilization of the other physical host nodes in data center. In this situation, each physical host node needs to know the CPU utilization of the other physical host nodes in data center to make its own decision.

5.3.3 Virtual Machine Placement

A physical host node has to migrate virtual machines away from itself to other physical host nodes whenever it is overloaded or underloaded. In underloading situations a physical host node has to migrate all the virtual machines away from itself to other physical host nodes, while in overloading situations the migration will continue until the CPU utilization of the host node falls below the threshold value. In our proposed strategy, virtual machine placement is done by the overloaded and underloaded physical host nodes themselves. In the VM placement schema a physical host node is selected to receive migrated virtual machines if it meets three conditions: 1) it has enough resources (CPU , memory, storage and network bandwidth) to place a new VM, 2) its CPU utilization dose not exceed the threshold value after placement, and 3) it shows a minimum increase in energy consumption among other physical host nodes after placement. Therefore, it is obvious that each overloaded or underloaded physical host node needs to have some information about the other host nodes in data center to make its own decisions. In the following, the general algorithm for dynamic virtual machine consolidation running at each physical host node shown as Algorithm 5.1.

5.4 A P2P Based Cloud Data Center

For communication reasons the data center is established on top of an unstructured P2P overlay using a simple gossip-based protocol. The gossip-based protocol builds a time-varying randomized P2P overlay in which each physical host node has just a partial system view, the so-called neighborhood. This property allows the system to

Algorithm 5.1: General Algorithm for Dynamic Virtual Machine Consolidation Running Inside Each Physical Host Node

```
if amIOverloaded (utilizationThreshold) then
  vmsToMigrate ← vmSelection (vmSelectionCriterion)
  VmPlacement()
end if
if amIUnderloaded () then
  vmsToMigrate ← getVmList()
  VmPlacement()
end if
```

scale with increasing number of physical host nodes as it does not rely on a central server. Physical host nodes can join and leave the network at any time without having any effect on the total performance of the system and also without the need to notify a central management unit. In the following subsections we will explain our gossip-based underloading detection as well as our VM placement schema. Also, we will explain how the system can tend to global efficiency when physical host nodes make local decisions within their own neighborhoods.

5.4.1 Underloading Detection Schema

Once a pysical host node is added to a data center, it initializes a local parameter *avgCpuUtilization* by its own CPU utilization. After the overloading management process, which can be done locally by each physical host node independently, all the physical host nodes periodically check whether they are underloaded or not. The process of the underloading detection schema for each physical host node is split into the following steps:

- Each physical host node triggers an underloading detection by exchanging *avgCpuUtilization* with other physical host nodes within the scope of the neighborhood and finds the average CPU utilization of physical host neighbor nodes.
- Each physical host node computes *avgCpuUtilization* being the average CPU utilization.
- Each physical host node compares its own current CPU utilization with *avgCpuUtilization*. If the current CPU utilization is lower than *avgCpuUtilization*, the host node is considered as being underloaded.

By running the underloading detection schema for several cycles the average utilization of the data center is disseminated inside the data center resulting in the global convergence of the underloading detection.

5.4.2 Virtual Machine Placement Schema

In our virtual machine placement schema, the VM placement optimization algorithm is triggered by an overloaded or underloaded host node itself within its own neighborhood. Running the VM placement algorithm within the scope of the random neighborhood facilitates the avoidance of a performance bottleneck in physical host nodes and also the convergence of the schema towards a global optimum close to a centralized approach. The process of the schema is split into the following steps:

- Once a physical host node considers itself as overloaded/underloaded, it initiates its VM placement optimization algorithm by checking whether it is currently receiving virtual machines from physical host neighbor nodes. If so, the algorithm is aborted. Otherwise, it attempts to acquire a lock for each member (including itself) of its neighborhood. Acquiring a lock on a neighbor node succeeds if the neighbor node is not already considered overloaded or underloaded, and also is not under virtual machine placement by yet another physical host node. If a neighbor node cannot be locked, it will not participate in the virtual machine placement process. The lock mechanism [22] allows us to enforce mutual exclusion for accessing physical host node resources in case of multiple ongoing consolidations.

- Next, the physical host node requests the CPU utilization, the threshold value and power state (Sleep/Wakeup) from the locked neighbors.

- Once the physical host node receives the information from locked neighbors, it executes an optimization algorithm to migrate own virtual machines to them. The output is a migration map which is run by the Virtual Machine Monitor (VMM) of the host node.

- After the migration all locks are released.

In real implementations of this scheme though, locks must be released automatically after some maximum time, to cope with nodes crashing while holding locks to any of their neighbors. Thus, locks are leased only.

5.4.3 Placement Optimization Algorithm

The proposed virtual machine placement schema can be adapted by any centralized optimization algorithm with a little modification. In our study, we modify a Power Aware Best Fit Decreasing (PABFD) strategy [6]. In this algorithm, we sort all the VMs (here, the VMs that must be migrated due to an overloading or underloading situation) in decreasing order of their current CPU utilization, and allocate each VM to a host node which has firstly enough resources (with consideration of the threshold value) to receive a VM, and secondly provides the least increase of the power consumption caused by the allocation. If the host nodes in the locked neighbor list do not have enough resources

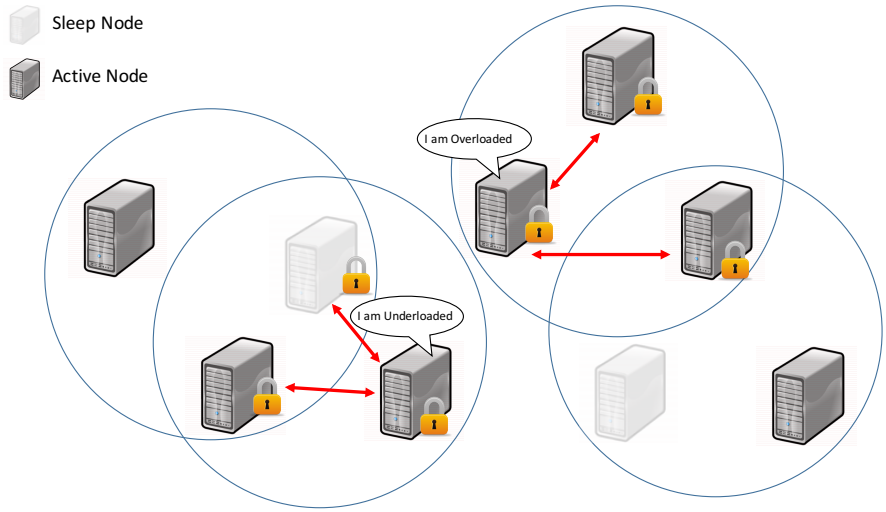


Figure 5.1: A schematic representation for the P2P based VM consolidation algorithm.

to receive all the virtual machines, the algorithm awakes sleeping neighbors by sending wakeup signals, locks them, and migrates the VMs to them. The pseudo-code of the algorithm is presented in Algorithm 5.2. The complexity of the algorithm is nm, where n is the number of physical host nodes and m is the number of VMs that have to be allocated. A schematic representation for the P2P based VM consolidation algorithm with $k = 3$ is given in Figure 5.1.

5.5 Simulation Setup

In order to evaluate the performance of our proposed approach, we carried out simulation runs using our cloud data center simulator that we have developed in-house over PeerSim simulator tool [62]. PeerSim is a peer-to-peer network simulator written in Java, allowing the simulation of systems with varying cardinality. Each peer in our simulation represents a physical host node, having its own simulated CPU performance counters, amounts of RAM, disk storage and network bandwidth. Each host node has a controller to manage overloading and underloading situations as well as placement of its own virtual machines which must be migrated. To run the consolidation algorithm within the scope of neighborhoods, the simulated P2P system is based on a simple gossip protocol in which the nodes are linked randomly to each-other to form a random graph having the specified degree k parameter. In our study, we simulated a P2P-based heterogeneous data center with 8000 physical host nodes. Half of them were simulated to be HP ProLiant ML110 G4 (Intel Xeon 3040, 2 cores x 1860 MHz, 4GB) and the others were simulated to be HP ProLiant ML110 G5 (Intel Xeon 3075, 2 cores x 2660 MHz, 4GB). The frequency of the host nodes' CPUs are mapped onto a MIPS rating. Table 5.1 shows the power consumption of each type of server at different load levels in Watts [8]. The characteristics

Algorithm 5.2: VM Placement Algorithm

```
Input = lockedMemberList, vmList Output = Migration map
{the vmList is a list of virtual machines that must be migrated away due to an
overloading or underloading situation}
vmList.sortDecreasingUtilization( )
for vm:vmList do
  minPower ← MAX
  allocatedHost ← NULL
  for host:lockedMemberList do
    if host.getState( ) is wakeUp then
      if host.utilization+vm.utilization≤host.threshold then
        power ← estimatePower(host, vm)
        if power < minPower then
          allocatedHost ← host
          minPower ← power
        end if
      end if
    end if
  end for
  migrationMap.add(vm, allocatedHost)
end for
if !vmList.isEmpty( ) then
  for host:lockedMemberList do
    if host.getState( ) is sleep then
      host.setState(wakeUp)
      for vm:vmList do
        if host.utilization+vm.utilization≤host.threshold then
          migrationMap.add(vm, host)
        end if
      end for
    end if
  end for
end if
return migrationMap
```

Server	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

Table 5.1: Power consumption in two selected servers in Watts

of the VM types correspond to Amazon EC2 instance types, with the only exception that all the VMs are single-core. The workload data of each VM is generated uniformly at random. The physical resources have been initially allocated to the random type of VMs based on a First Fit strategy. As a result, 46605 virtual machines have been deployed in our data center. The initial allocation is same for each simulation trial and the number of virtual machines is fixed during data center life time. In all the simulation trials, the CPU utilization's threshold has been empirically set to 0.9.

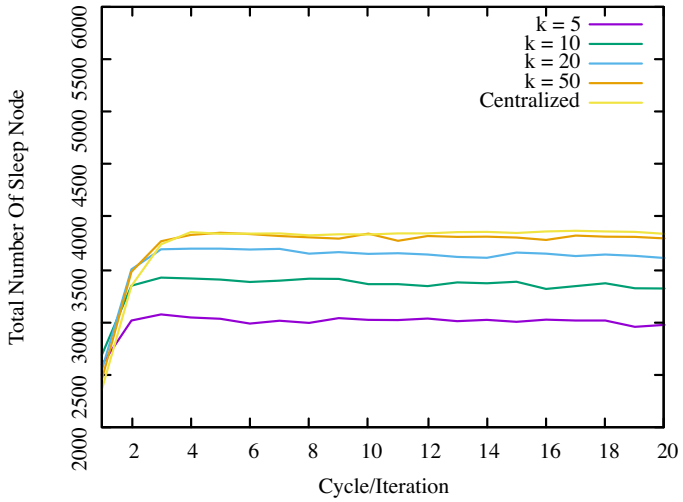
5.6 Experimental Results

The results were obtained by running the proposed gossip-based dynamic virtual machine consolidation for 20 cycles simulation (each cycle is equal to 300 seconds). Figure 5.2 compares the total number of sleep nodes and also the total consumed energy in data center at each cycle/iteration of the algorithm for different size of neighborhood k with a centralized approach.

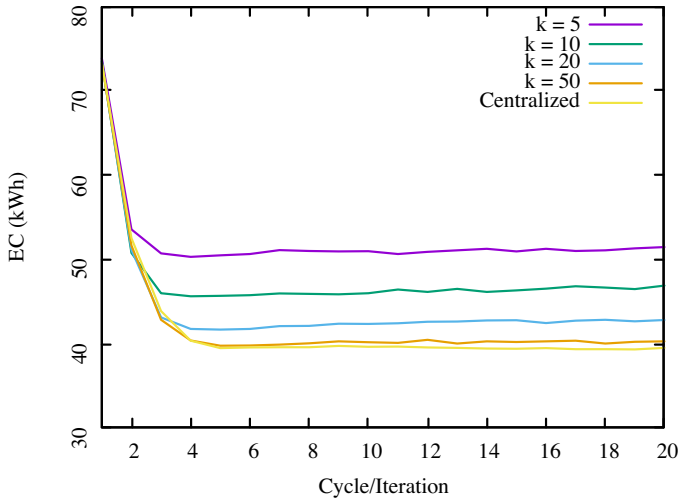
Figure 5.2(a) shows that the gossip-based dynamic virtual machine consolidation can maximize the total number of sleep nodes and consequently achieves a good energy efficiency (see Figure 5.2(b)) close to the centralized approach if the neighborhood size k is set to be at least 50. The reason that the system with small neighborhood size can not achieve a good energy efficiency is that, in overutilization situations, physical host nodes do not have a high chance to find proper active/wakeup neighbor nodes (physical host nodes which have enough idle resources) to place all their own migrated virtual machines resulting in either waking sleep neighbor nodes up or aborting virtual machine migrations.

In addition, in a small neighborhood size, power switching rate (switching between sleep and wakeup mode) is very high, because in this situation in one hand, many more cycles are needed until the overall average utilization of the physical host nodes to be disseminated across data center (note that, the overall average utilization is the criterion to detect underloading situation), resulting in switching more physical host nodes into the sleep mode, and on the other hand as stated before, a lots of switching into the wakeup mode is also required due to lack of enough active neighbor nodes to place migrated VMs. With this in mind, the Figure 5.3 highlights the power switching rate in each cycle for two different neighborhood size.

In the next experiment we compared the performance of the algorithm in terms of average CPU utilization (in the whole data center) in two different approaches: a gossip-based approach with different neighborhood size k, and a centralized approach. Figure 5.4 illustrates how the gossip based approach with neighborhood size $k = 50$ can

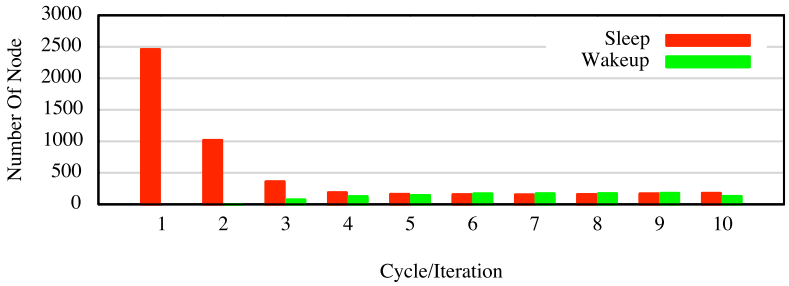


(a)

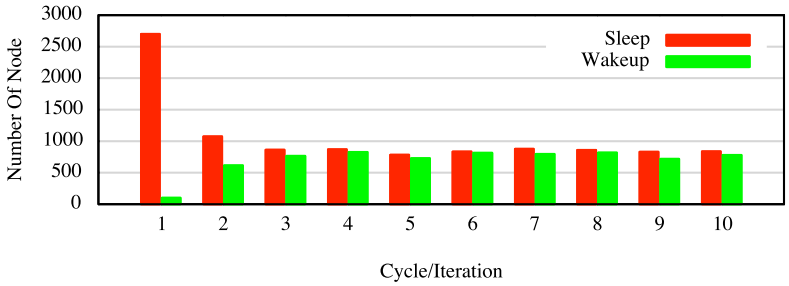


(b)

Figure 5.2: Total number of sleep nodes and total amount of energy consumed inside data center



(a) for k = 50



(b) for k = 5

Figure 5.3: Number of sleep and wake-up actions in each cycle

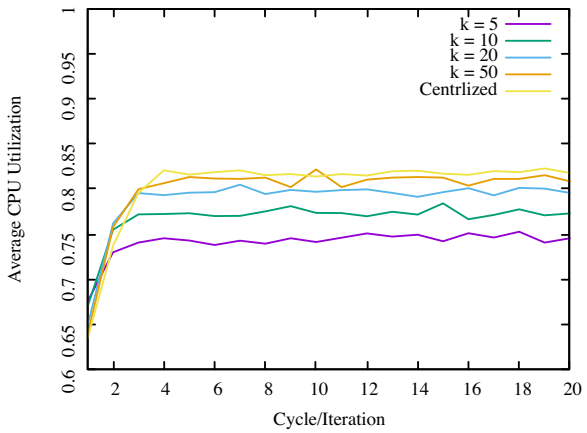
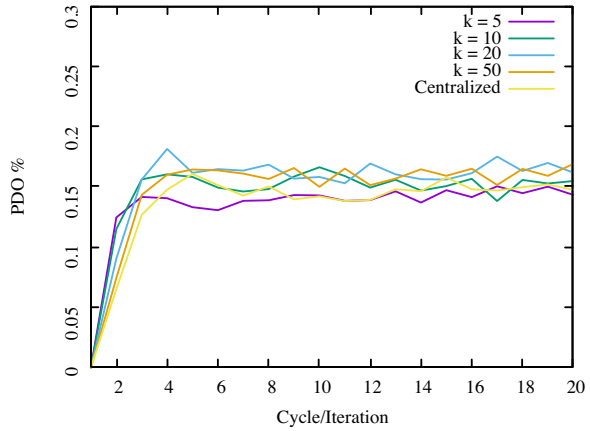


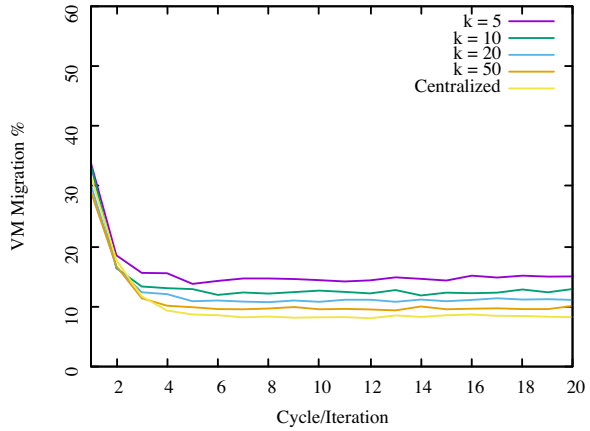
Figure 5.4: Average CPU utilization inside data center

increase average CPU utilization in the data center very close to the centralized approach. However, the average CPU utilization in the gossip based algorithm fluctuates over time, while the centralized approach shows more stability.

Finally, Figure 5.5 compares the Performance Degradation due to Overloading (PDO) and also the number of migrated virtual machines in percentage in both approaches. A



(a) Performance Degradation due to Overloading (PDO)



(b) Number of migrations

Figure 5.5: SLA violation metrics

physical host node with a higher utilization has a higher risk to experience 100% utilization due to a bursty demand of resources by applications running on the VMs. Therefore, it is obvious that overloading (experiencing 100% CPU utilization) is more likely in a data center with a high average CPU utilization. As stated before, a gossip-based VM consolidation approach with a larger neighborhood size can achieve a higher average CPU utilization in data center, therefore, as Figure 5.5(a) shows it can be less tolerated in Performance Degradation due to Overloading (PDO). In contrast, Figure 5.5(b) shows that the number of migrations will be decreased with increasing the size of neighborhoods. This is caused by the fact that decreasing the neighborhood size will increase the power switching rate and consequently increases the number of migrated virtual machines in data center.

CHAPTER 6

Self-Adaptive Capacity Controller in Overbooked Data Centers

Resource overbooking [83, 12, 29, 9] is a well known technique commonly used to mitigate the low resource utilization ratios reported by large datacenters, such as Google [73] or Amazon EC2 [48]. However, this utilization increment also increases the possibilities of having VMs interfering with each other, as more VMs are competing for the same resources which may lead to delays when accessing them. This is a well known effect known as *noisy neighbor problem* [89], which may heavily impact application performance.

On top of this problem, not all the VMs require the same Quality of Service (QoS), meaning they do not need to maintain the same throughput or response times over time. Furthermore, not all the VMs are equally affected by this interference [84]. For instance, a deadline constrained application (e.g., a computationally intensive task) does not need to maintain an average performance all the time as long as it completes the task in time. By contrast, an interactive application may require a certain minimum performance all the time, as given for instance by e-commerce systems.

In order to treat VMs with different QoS requirements differently, the authors of [83] presented a QoS differentiation (high and low QoS levels) solution by extending their previously developed overbooking framework [83]. This QoS differentiation creates different isolation levels between VMs by pinning them to specific cores inside the servers (note core pinning is a mechanism provided by KVM [37]), thus limiting the impact they may have on others. However, this QoS classification impacts the overall utilization when high QoS VMs are overprovisioned. In this chapter, we address research question III by presenting its corresponding contribution V, introduced in Chapter 1. We first present a background and theory development and then we will motivate the need of a self adaptive performance aware capacity controller in overbooked data centers in Section 6.1. In Section 6.2 we will present our solution based on a Fuzzy Q-learning (FQL) algorithm. Finally in Section 6.3 we will present our experimental setups and results.

6.1 Background

In [87] the authors, address low utilization ratios at cloud datacenters by presenting a framework which performs overbooking decisions based on long term risk estimation, i.e., the possibility of ending up in an overload situation leading to performance degradation. In [86] they extend this overbooking framework with an admission control that takes decisions based on fuzzy risk estimation to better account for uncertainty about future applications capacity needs, combined with a proportional-internal-derivative (PID) set of controllers [2] that adjusts the acceptable risk levels based on the deviation of the current utilization from the target [83].

Applications' tolerance to overbooking are different, not only among them but also over time (depending on their workload). This fact makes it difficult to assess the possible impact of potential overload situations. To overcome these difficulties, the authors of [84] firstly worked on a mitigation and recovery method for unexpected situations, where on the one hand the overbooking pressure is adjusted based on the behaviour of the currently running VMs, and on the other hand a short-term mitigation strategy based on Brownout [43] was used, ensuring graceful degradation (of some VMs) during load spikes. This approach may reduce overall datacenter utilization (by reducing the overbooking pressure) due to some VMs not being able to deal with the current overbooking level. However, the source of the problem may not be a too high utilization but rather VM interference. Therefore, in order to mitigate this VM interference problem, also known as noisy neighbor problem [89], the same authors studied the use of pinning mechanisms as a way to both provide isolation between VMs inside the same physical server, and perform QoS differentiation between VMs based on this isolation capacity [88]. Note that KVM core pinning is proposed in [37] as a feature needed to ensure reliable and repeatable performance.

In this work we use the KVM core pinning functionality to offer two different QoS levels (or isolation levels) by performing virtual cpu (vcpu) to physical cpu (pcpu) pinning using Algorithm 6.1, based on the QoS schema depicted in Figure 6.1 (left):

- **High QoS**: application vcpus are pinned to pcpus and get exclusive access to them – no other VM vcpus can be pinned to these pcpus.
- **Low QoS**: application vcpus are not pinned to any pcpus and can use any pcpus except the ones booked for high QoS applications.

This QoS classification however impacts the overall utilization when high QoS VMs are overprovisioned. To deal with this problem, a controller based on the high QoS applications performance, incrementally decreases their isolation level by sharing some of their cores, as presented in Figure 6.1 (right [88]). More specifically, at every control interval where a high QoS VM is behaving better than needed, an extra physical core allocated to it will be shared with the low QoS VMs. This process keeps on as long as the target performance of the high QoS VM is maintained. Otherwise, the maximum isolation level is established again to avoid any possible performance degradation by recovering the complete isolation. As applications needs are unknown, increasing the

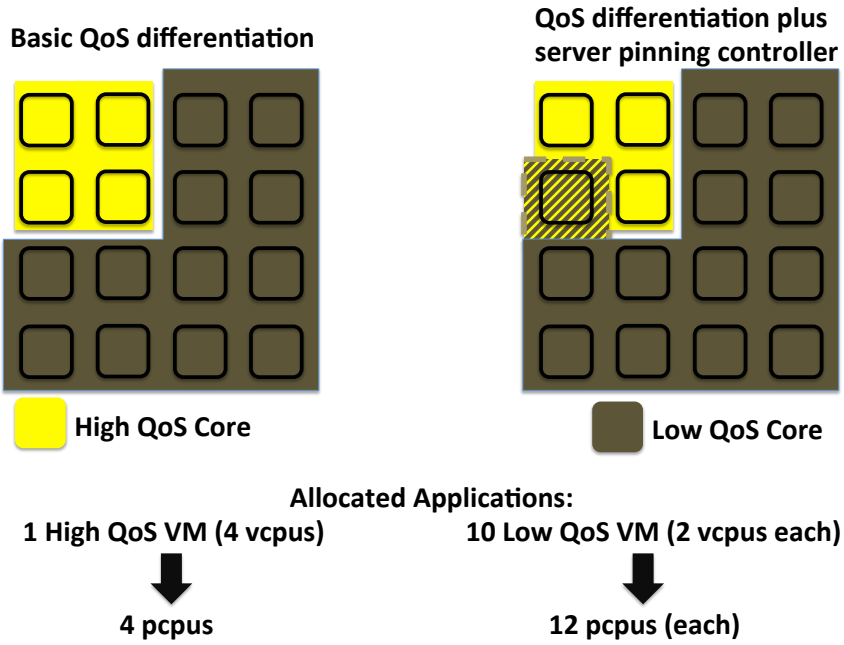


Figure 6.1: Virtual to physical core mapping using basic QoS differentiation (left) and advanced QoS differentiation by using a re-mapping controller (right).

Algorithm 6.1: Application Core Pinning

Configuration parameters: VMs, list of running VMs
H_QoS, list of high QoS pcpus
L_QoS, list of low QoS pcpus, including empty pcpus in case of no low QoS VMs (sorted by aggregated resource usage: CPU*Mem*IO)

```
1:  num ← number of vpcus of new_vm
2:  if new_vm requires HighQoS then
3:      cores ← first num pcpus of weighted list of L_QoS
4:      for each core ∈ cores do
5:          Pin vcpus of new_vm to core
6:          H_QoS ← H_QoS + core
7:          L_QoS ← L_QoS − core
8:      end for
9:  end if
10: for each vm with LowQoS req. ∈ VMs do
11:     //Repin all low QoS VMs
12:     Pin vm to all pcpus ∈ L_QoS
13: end for
```

isolation level when performance degradation is observed step by step instead of at once may impact the application during a longer period of time, which in turn may lead to a longer time to recover the desired/required performance. Note that this is performed at a per (high QoS) VM granularity level, i.e., there is a capacity controller per each high

QoS VM.

Although this controller presents good results regarding performance isolation of the high QoS applications, it is a really conservative approach that only works in a *reactive manner*. Therefore, it may have bigger impact in the overall utilization ratios than desired, specially with high fluctuating workloads. Furthermore, it may also affect the low QoS applications due to the abrupt (and more frequent) reductions in the amount of cores that they are entitled to use, as high QoS applications recover the isolation on all the cores at once.

To address this problem we propose a self-adaptive controller exploiting an online machine learning approach to efficiently learn decision policies in terms of isolation level, (i.e., the number of cores to share) with regard to the applications behavior. Our approach is based on a particular reinforcement learning technique called Fuzzy Q-Learning (FQL), previously expliand in Chapter 2, which is a combination of Q-Learning (QL) and fuzzy logic. In our proposed controller the QL algorithm, as a decision making engine, learns how to map the input states to the desired output decisions (here, the number of shared cores) to maximize the shared capacity over time, helping to increase utilization while maintaining the performance of the application in terms of the response time. As QL is a table-driven learning algorithm, using fuzzy logic can facilitate it to deal with large or continues state space resulting in handling high dimensionality. In addition, using fuzzy logic can enable knowledge encapsulation into the learning table, i.e. to merge a prior or expert knowledge into the problem.

6.2 Fuzzy Q-Learning Capacity Controller

An efficient capacity sharing between VMs co-located in the same server is achieved if enough capacity is provided to all the VMs (based on the QoS-level), and at the same time a high utilization ratio is achieved. On the one hand, utilization is increased by maximizing the amount of cores that high QoS VMs share with low QoS VMs, reducing the former's isolation level. On the other hand, the isolation level needs to be enough so that high QoS performance is not affected due to the capacity shared with lower QoS VMs. Consequently, this trade-off needs to be discovered and updated over time as it will be different not only for VMs mixture, but also for different workload patterns.

It is also important to make isolation level decision durable, i.e., to avoid constant and abrupt changes into the amount of cores being shared by the high QoS VMs as this will impact all the co-located VMs. On the one hand, it will impact the low QoS VMs due to the sudden fluctuations in the available capacity they are entitled to use. On the other hand, it will impact the other co-located high QoS VMs, since, even though high QoS VMs do not share cores among themselves, the low QoS VMs will have higher needs of the cores being shared by the rest high QoS applications, therefore having an impact in their performance, which in turn will lead to even more fluctuations in the number of cores they shared.

Due to the benefits of Fuzzy Q-Learning, we present an approach where the isolation level of the high QoS VMs, i.e., the amount of cores they share with the low QoS VMs,

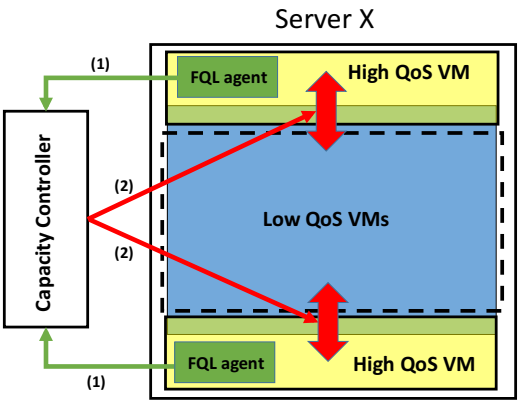


Figure 6.2: Architecture overview.

is managed by a **Capacity Controller** whose decisions are based on the **FQL agents at each high QoS VM**, as depicted in Figure 6.2.

Owing to the learning capacity, the FQL agents can predict the applications behavior once they gathered enough knowledge about them, and thanks to that they can proactively increase/decrease their exclusive access to resources before the performance degradation happens. What is more, as it tries to reduce the isolation level as much as possible, it enables the possibility of sharing more resources to the (low QoS) co-located VMs as it is aware about the next states of the application behavior in the future. For example, lets imagine a situation where a 8-cores high QoS VM needs more than 2 but less than 3 cores in isolation at the present. In this situation, the approach presented in [88] would increase the amount of shared cores one by one (i.e., 0, 1, 2, 3, 4, 5, and 6) up to the point where the number of cores shared is too high and the application has problems to keep up its performance. Then, the complete isolation would be recovered and the process would start again (from 0 to 6 shared cores). This behavior may impact the low QoS applications due to the abrupt changes in the number of cores they are allowed to use, as well as it would reduce the chances for higher utilization and/or accepting more VMs. By contrast, thanks to the learning behavior of the FQL agent, it realizes (after a few of iterations) that sharing 5 cores is the right option. Now imagine the application running on the VM suddenly demands more resources. In this situation the controller presented in [88] would again recover the complete isolation, while with the FQL approach the number of isolated core will proactively change to the new required isolation level, even before the application needs it. These are just simple examples, but more complex application behavior patterns can be discovered and learned by the FQL agents leading to a better resource sharing over time.

6.2.1 FQL Components

The FQL agents are associated to the VMs running the high QoS applications. The components of an FQL agent are described in the following:

- **State:** The combination of the current response time and the current isolation level in terms of number of shared cores describe each state inside a high QoS VM. These two characteristics have enough information to represent a predictable application behavior. Therefore, the input state vector to the FQL is defined as

$$x_t = [rt_t, nc_t]. \tag{6.1}$$

where rt_t and nc_t denote the current response time and the current number of isolated cores respectively.

- **Action:** Each element of the action set o denotes the number of cores needed in isolation, for transiting from current state to the next state.

$$o = [nc^1, nc^2, ..., nc^i]. \tag{6.2}$$

- **Reward Function:** The design of a reward function is the **key** to build reinforcement learning system. It maps each perceived state-action pair of the environment to a single number, a reward, indicating the intrinsic desirability of that state-action. The goal of the reinforcement learning is to maximize the cumulative reward over time. This is obtained if the learning agent seeks actions that result the highest q-value. In our system, the reinforcement learning tries to optimize the trade-off between sharing capacity and the performance over time, so the reward function can return a value determining a success rate, here **maximizing the number of shared cores** and **minimizing the response time**:

$$r_{t+1} = (1 - nc_t/tc) - (rt_t/tr), \tag{6.3}$$

where tc is the total number of physical cores assigned to a high QoS VM, and tr is the target response time. It is obvious that if the high QoS VM controller shares its cores as much as possible ($nc \Rightarrow 0$) while keeping the response time as far below as possible from the target response time ($rt \Rightarrow 0$) it will receive more reward. If the reward is close to zero this implies that the action is not effective and a negative reward is considered as punishment for the learning agent.

6.2.2 Let Reactive and Proactive be Friends

As FQL agents learn by trial and error interaction with the environment, it is likely to degrade the performance due to either random actions in an exploration mode or lacking enough learning knowledge in its q-table during the learning process, i.e., under new situations never experienced. The learning agents are making decisions in terms of the number of isolated cores for the high QoS VMs, where a response time higher than the target threshold is crucial in terms of SLA violations.

Our proposed controller can guarantee the performance for the high QoS VMs after the learning process but during the learning or under completely new situations there

is no guarantee. Although, combining the Q-learning with the fuzzy logic can speedup the learning process (as stated in the previous section), such performance degradation may not be acceptable for the high QoS users even for a short-term. To address this problem, our proposed learning controller calls a **reactive controller** to act on its behalf once the response time exceeds the set target threshold. In this situation the reactive controller establishes the maximum isolation level for the VM again. In addition, the learning controller receives a punishment proportional with the amount of exceedance (see Eq. (6.3) where $rt > tr$) once it calls the reactive controller. Therefore, the learning controller also learns how to act to decrease the number of calls over time.

6.3 Experimental Setup

The performance of our proposed capacity allocation controller for QoS assurance and differentiation based on FQL agents is evaluated next. The tests are conducted on two machines, one hosting applications and one generating the workload, connected through a 1 GB link. The first machine is a server consisting of a total of 32 cores (AMD Opteron™ 6272 at 2.1 GHz) and 56 GB of memory. KVM was used as a hypervisor and each application was deployed inside a VM. The second machine is a 4-core (Intel Core™ i5 processor at 3.4 GHz) desktop with 16 GB of memory. Note that as the capacity allocation controller works at server level, this is straightforwardly extensible to any number of servers.

6.3.1 Applications and Workload

We have emulated a representative cloud workload by mixing different types of VMs that can be grouped in two main classes (similarly to the boulders and sand scheme reported by Google in [73]). The first VM class is represented by (usually big) long-living VMs, running interactive applications that usually present some seasonality pattern in their use. For this application class we have used the *RUBiS* [75] and *RUBBoS* [74] cloud benchmarks, which are an auction website benchmark modeled after eBay and Slashdot, respectively. For the experiments we consider a fixed set of them in each run: 2 VMs requesting half of the server capacity (8 vcpus and 14 GB memory each). Note that as they are interactive applications, they are submitted with the high QoS requirement, as they can be more affected by other co-located VMs than the deadline oriented applications, similarly to the scenario presented in [73, 49]. As regards to the workload, a number of queries have been generated using information extracted from different days of the Wikipedia traces [66], and time-shifted one of them 12 hours, as shown in Figure 6.3, creating different trends, peaks, and daily usage patters. The client queries were generated using the `httpmon` tool[1].

The second VM class consists of different types of relatively short living, non-interactive VMs. Firstly, in order to increase the uncertainty in the system, and thus creating a more realistic aggregated workload, we created different VMs that run shell

[1] https://github.com/cloud-control/httpmon

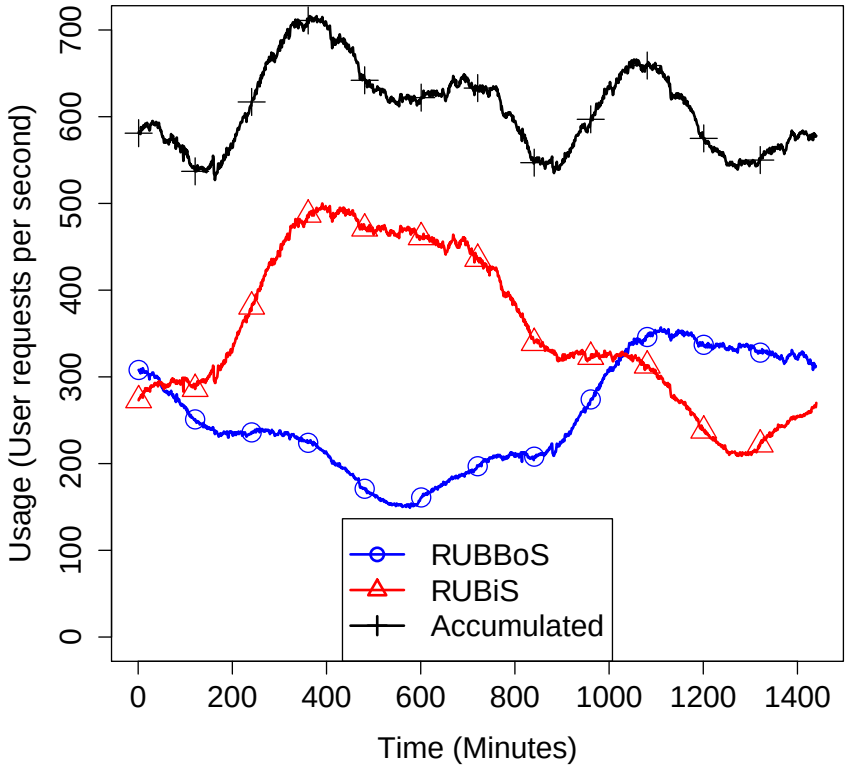


Figure 6.3: Workloads for RUBiS VMs.

scripts that consume random amounts of CPU and memory over time. This creates a set of VMs with highly heterogeneous and time-varying resource requirements, with both bursty and steady behavior in their resource consumption. On the other hand, to have a measurable performance of this class of VMs, we have also created different types of VMs that continuously solve random sudokus[2] and report their corresponding throughput achieved over time. Additionally, these sudoku VMs can be differentiated by their two main behaviors:

- *SudokuBE*: solves a certain amount of sudokus before a given deadline. This type of VM behaves in a best effort manner, trying to solve as many sudokus as possible and therefore using as much CPU time as available.

- *SudokuT*: keeps a certain throughput (number of solved sudokus) over time. If during one time period the target is not achieved, the sudokus queue up, and need to be solved in the next period (i.e., open-loop). We mix two different SudokuT types, with a target of 11 and 22 solved sudokus per second, respectively.

[2]http://norvig.com/sudoku.html

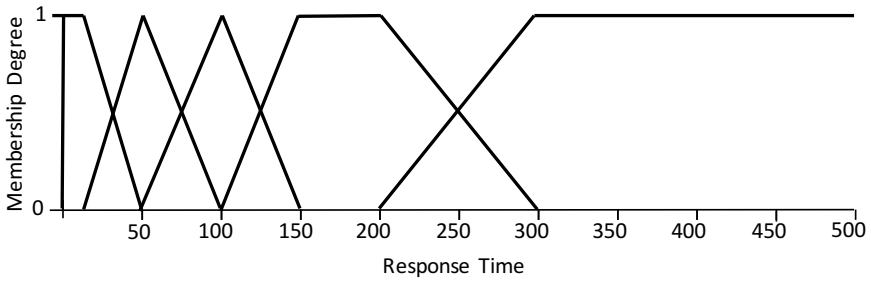
The arrival pattern of these second type of VMs is generated using a Poisson distribution with $\lambda = 20$ seconds. This leads to a final workload consisting of 2 high QoS VMs represented by the web servers, and a varying number of low QoS VMs consisting on a mix of the different Sudoku and shell scripts VMs. Note the resulting overall workload stresses the CPU usage most, therefore we evaluated this capacity dimension in the next section. Besides, the submission ratio is high enough to keep the system always saturated, stressing the controller reaction to an extreme case with high chances of VMs interference.

6.3.2 FQL Settings

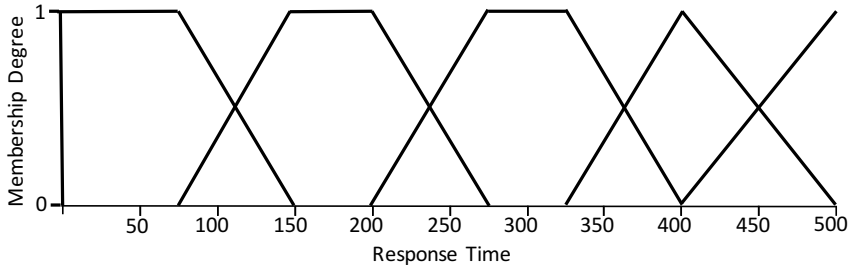
In the FQL setup, the discount factor γ has been set to 0.5. This value for the discount factor allows the learning agent to consider the current and long-term rewards equally. This would be a proper value for our case study where we would like to be a little bit conservative in terms of taking actions while the future is also important for us. The learning rate β changes over time. Whenever the controller receives a reward the learning rate is set to 0.1 and whenever it receives a punishment it is set to 1.0. This guarantees that "bad" news travels faster than "good" news enabling the learning agent response to the performance violation rapidly. We applied trapezoidal and triangular shaped membership functions as shown in Figure 6.4 for the response time of the RUBiS and RUBBoS VMs and also applied triangular shaped membership functions for the number of cores. Note that the design of the fuzzy sets for the input parameters is based on our prior knowledge about the workload of RUBiS and RUBBoS VMs, presenting a more conservative approach for RUBBoS as its behavior is less linear with regards to the number of users. It is worth noting that, the fuzzy sets can also be tuned at run time with an alternative approach [19]. As Figure 6.4 shows we have 5 fuzzy sets for the response time and 9 fuzzy sets for the number of cores which provides (9×5) 45 rules in our Fuzzy Inference System (FIS). Note the rules are considered as the states for the Q-learning. The benefit of using FIS is obvious here as much less states are generated for the Q-learning algorithm. As a consequence, the learning process can be accelerated and the space complexity is decreased – the space complexity in Q-learning is always $O(S \times A)$, where S is the number of states and A is the number of actions. The value of ϵ in our EE policy has been set to 0.9 (as stated before, it is typically chosen close to and below 1), meaning that the agent acts randomly with the probability of 0.1 and enforces a learned action with the probability of 0.9. It helps our system combats overfitting. The FQL iteration for perceiving the next state and receiving the reward for the previous action has been set to 30 seconds. Furthermore, the q-table has been initialized by zero in order to start with zero learning knowledge.

6.4 Performance Evaluation

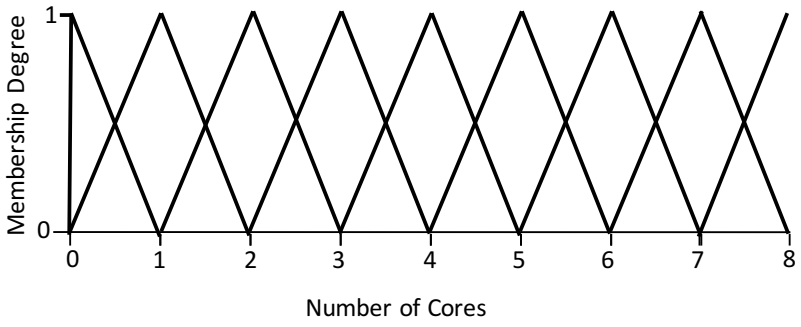
To evaluate the performance improvement achieved thanks to our Fuzzy Q-Learning (FQL) based capacity management controller, a comparison with the performance achieved



(a)

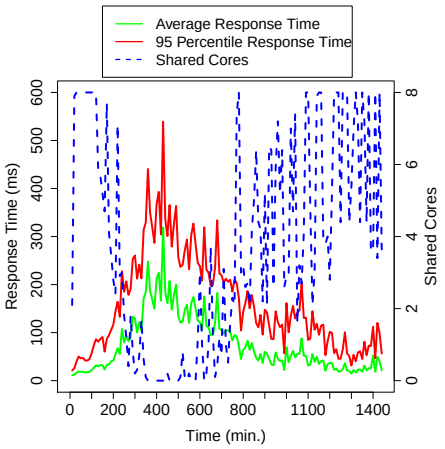


(b)

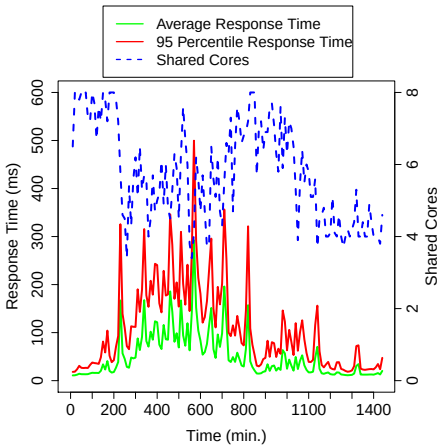


(c)

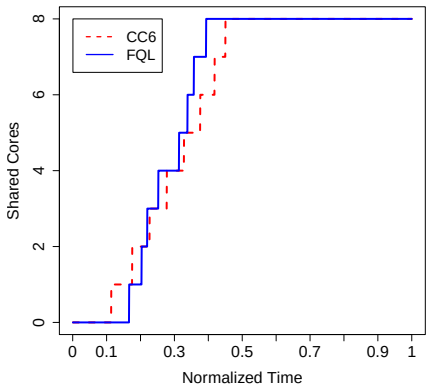
Figure 6.4: Fuzzy sets. (a) For the response time of the RUBBoS VMs, (b) For the response time of the RUBiS VMs, (c) For the number of cores of both RUBBoS and RUBiS VMs.



(a) CC6



(b) FQL



(c) Shared Cores Summary (1st day)

Figure 6.5: RUBiS performance (1st day).

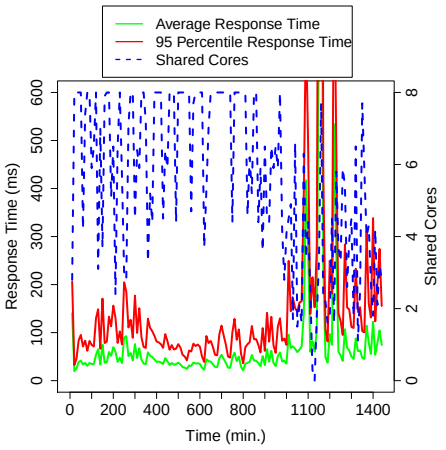
by the controller implemented in [88], labeled as CC6, is presented next. As explained at Section 6.1, the controller proposed in [88], unlike our current proposal, behaves in a conservative, reactive manner, progressively increasing the amount of cores being shared if the performance is maintained, and recovering the complete isolation at once when there is problems to maintain it. On the one hand, we measure the average and 95-percentile **response time** for the RUBiS and RUBBoS VMs in order to ensure that high QoS applications performance is maintained. On the other hand, we evaluate the **overall utilization** and number of **shared cores** over time, as well as the number of sudoku **VMs concurrently running**. The **throughput** achieved by the sudoku VMs is also measured to ensure that low QoS applications performance is acceptable, i.e., they keep a specific throughput over time and/or meet their deadlines. Finally, in order to conclude whether there is an evidence of statistically significant improvement in relation

with any of the previously described metrics, we use the Wilcoxon statistical test [96], a non-parametric statistical hypothesis test that compares two related samples to assess whether their population mean ranks differ.

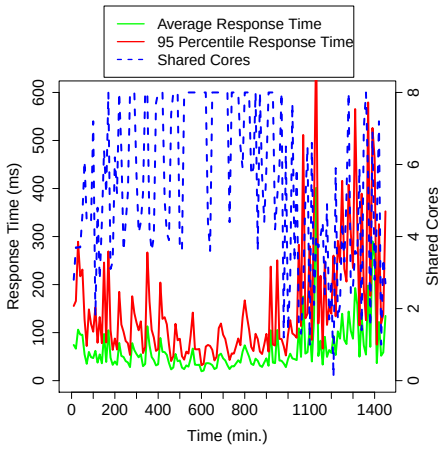
Figure 6.5 shows the performance comparison between CC6 and FQL as regards to the RUBiS application performance. As Figures 6.5(a) and 6.5(b) depict, there is no remarkable difference at the response time since the RUBiS VM presents a linear performance regarding the number of request, which facilitates to keep the response times within the desired limits. However, it may be highlighted that the trend for FQL is slightly flatter due to sharing a more stable number of cores over time. In fact, the Wilcoxon statistical test concludes that the reduction on the response time (both average and 95th) is statistically significant (p-value $<$ 5.0e-07). It can be seen at Figure 6.5(c) that the amount of cores shared over time is pretty similar for both approaches (FQL shares slightly more cores), sharing all the cores around half of the time for both approaches. However, as higlighted by Figures 6.5(a) and 6.5(b), the number of cores shared over time quickly and largely fluctuates for CC6, while remains more stable for our approach.

Figure 6.6 and Figure 6.7 show the same comparison between CC6 and FQL but for the RUBBoS application, which presents a less linear behavior with respect to the number of requests per second. In this case, as the performance fluctuates more, we show its evolution over three days, to also highlight the learning of the FQL approach. Although the performance is reasonable good for the first day for both approaches (Figure 6.6(a) and Figure 6.7(a)), with slights improvements for the FQL, the performance is improved day by day for the FQL approach. In Figure 6.6(b) and Figure 6.7(b) the differences between them are noticeable, with an overall reduction of the response times, specially towards the end coinciding with the RUBBoS workload peak. Not only the overall response time is reduced, but also the fluctuations, both in number and size. These differences are even more remarkable at the third day (Figure 6.6(c) and Figure 6.7(c)), where thanks to the learning, the FQL approach remarkably reduces the response times, as well as presents a more constant behavior compared to CC6. Again, the response time reduction is statistically significant, not only for the third day but also for the complete experiment (Wilcoxon p-value: 2.2e-16) As in Figure 6.5, it can be clearly seen a much more stable number of cores shared over time for the FQL approach.

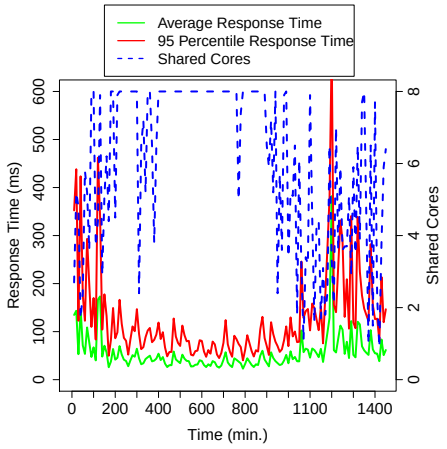
Regarding the learning process over time, Figure 6.7 clearly highlights the learning of the FQL-based controller, taking advantage of the gathered knowledge about the RUBBoS daily patterns, and making response time fluctuations much lower over time. It must also be noted that, due to the transitions between proactive and reactive mechanisms, the performance is acceptable during the three days. The FQL calls the reactive controller to act on its behalf when response time is not good and there is not enough learning knowledge yet gathered. That is the reason for the short peaks during the first days. Note these peaks also happen in CC6 approach. Due to the high load, the actuation needs to be done in a proactive way, otherwise even by not sharing any of the cores (as CC6 does), the VM needs some time to recover the already queued requests. We can see at Figure 6.7(c) that, even though the response time was already good during the second day, it is even improved during the third day by sharing a slightly less amount of



(a) 1st Day



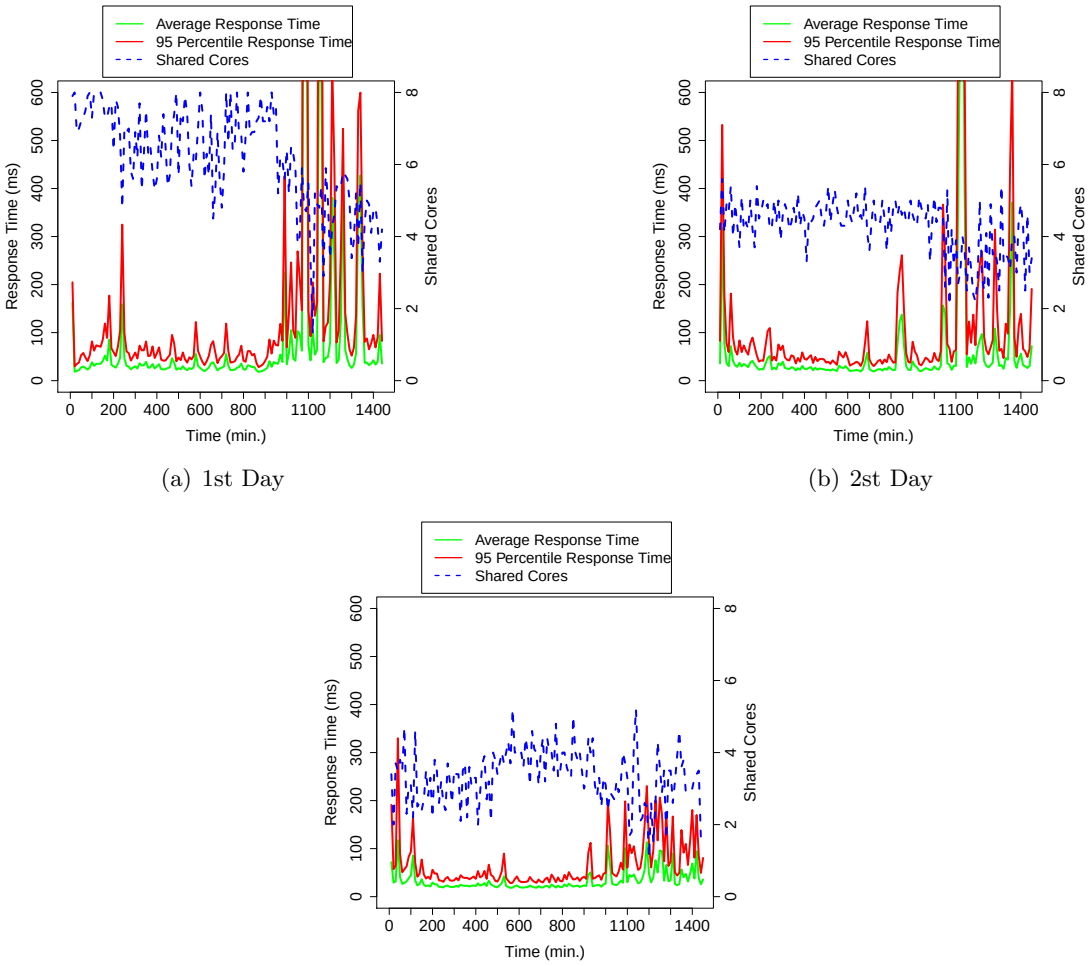
(b) 2st Day



(c) 3st Day

Figure 6.6: RUBBoS Performance (CC6).

cores, in favour of a more constant and predictable behavior. If we analyze together the sharing behavior during the third day, for both FQL agents, i.e., for RUBiS and RUBBoS applications (see Figure 6.7(c) and Figure 6.8(b)) and compared then with the behavior obtained for CC6 (see Figure 6.6(c) and Figure 6.8(a)), we can appreciate that thanks to the RUBBoS FQL agent sharing a less amount of cores, the RUBBoS application present a better behavior than before. What is more, as both FQL agents become aware of the system evolution over time, the RUBiS FQL agent is able to shared a larger amount of cores (remarkable different compared with CC6 for RUBiS during the third day – Figure 6.8(a)) as the RUBiS VM is more easy to control – more linear behavior regarding the number of request per second than the RUBBoS VM. Consequently, the controllers learned that it was better to be more aggressive in the sharing policy for RUBiS and more conservative for RUBBoS, as performance deviations in the former can be fixed



(a) 1st Day

(b) 2st Day

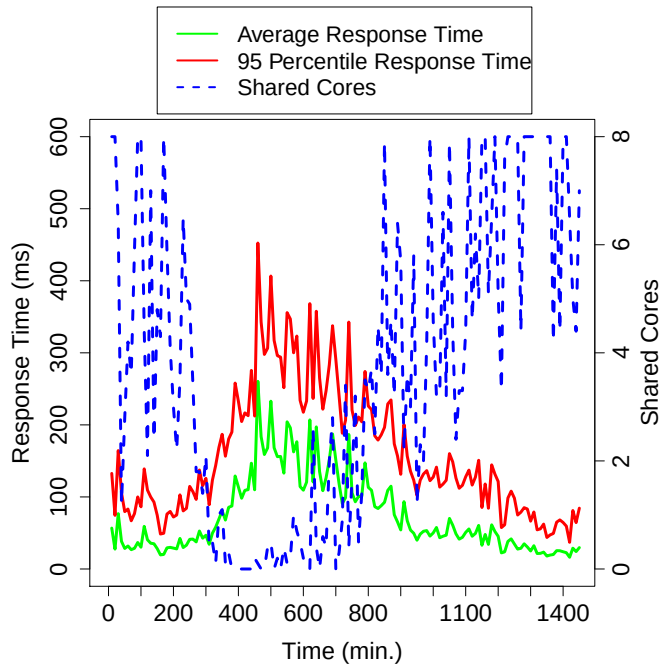
(c) 3st Day

Figure 6.7: RUBBoS Performance (FQL).

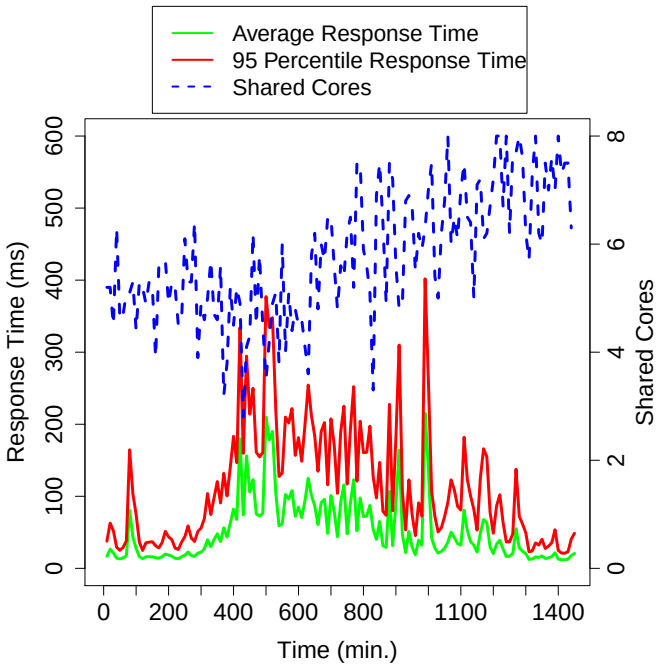
quickly just by removing a few shared cores.

All in all, a much more stable sharing for both FQL agents, with less and smaller fluctuations, is achieved. Additionally, as depicted in Figure 6.9, the FQL approach is also able to share more cores over time, specially reducing the amount of time where no cores were shared for the RUBBoS VM (from 30% of the time to just 10%), enabling the option to accept more VMs and therefore taking better advantage of the available resources.

In addition to the learning process highlighted at the RUBBoS response times, Figure 6.10 shows the differences between the amount of cores shared by the capacity controller for the complete three days, for both RUBiS and RUBBoS, highlighting the FQL agents learning over time. For RUBiS, as the performance was good all the time, there are no large variations among days. However, for RUBBoS we can see the controller



(a) CC6



(b) FQL

Figure 6.8: RUBiS Performance (3rd day).

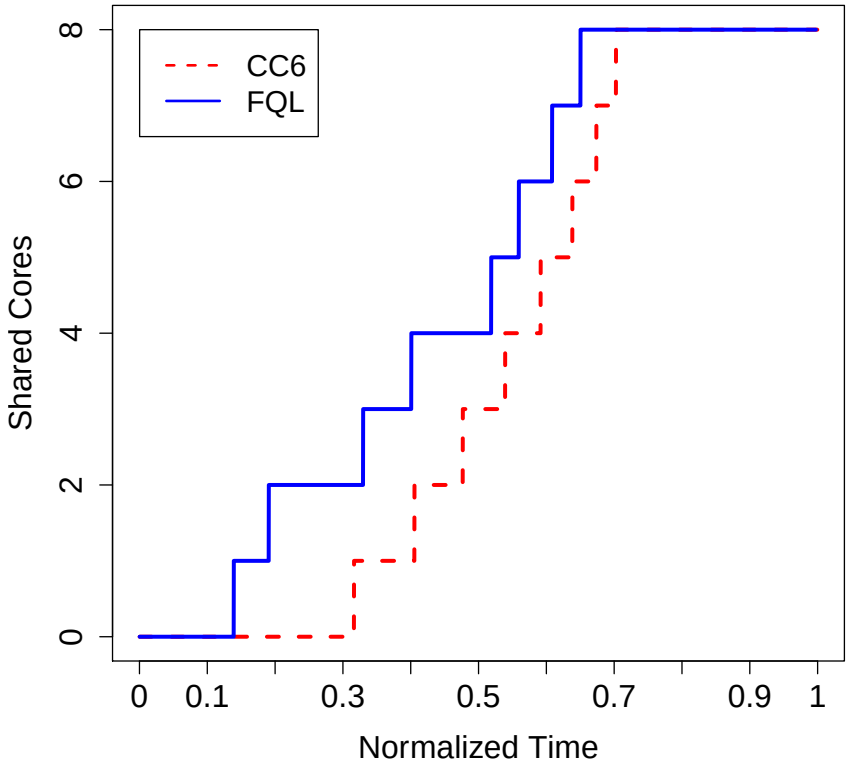
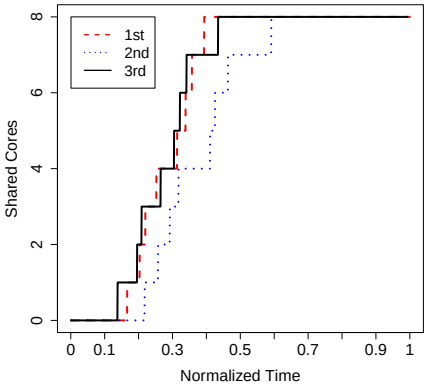


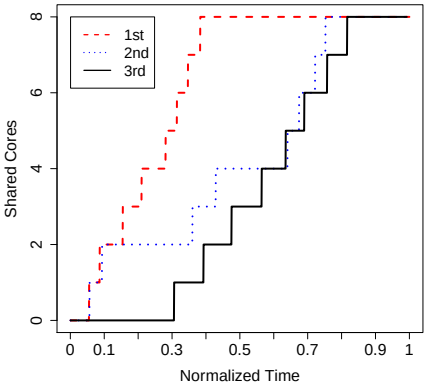
Figure 6.9: Shared Cores: RUBBoS (3 days).

clearly shares fewer cores for the last day (mainly due to the peak during the last part of the day) based on the knowledge acquired during the previous days, leading to an improved performance as shown in Figure 6.7(c). The aggregated number of cores shared by both FQL agents can be seen in Figure 6.10(c), where the boxplot clearly highlights the smaller variation in the overall number of cores shared over time, as well as the higher average. Additionally, the evolution over time is depicted at Figure 6.11, where the smaller fluctuations over time are also clearly highlighted. Once again, there is statistical proof of both FQL agents sharing more cores than the CC6 approach, with Wilcoxon p-values of 1.98e-07 and 2.2e-16 for RUBiS and RUBBoS, respectively. Additionally, a 35% reduction on the standard deviation of the amount of cores shared over time was achieved by our FQL controller.

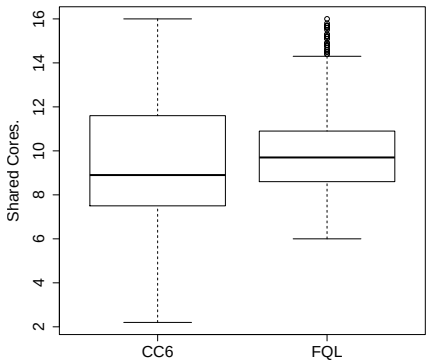
As shown in previous figures, the self-learning and self-adaptive FQL agents are able to share more cores and in a more stable way over time even without knowledge about the status of the infrastructure (i.e., the server). Consequently, the FQL-based capacity controller presents a more efficient sharing mechanism, where more cores are shared, and in turn more VMs can be accepted. This is highlighted in Figures 6.12(a) and 6.12(b),



(a) RUBiS (FQL)



(b) RUBBoS (FQL)



(c) Aggregated cores shared

Figure 6.10: Cores Sharing: Learning over time.

where a histogram for the amount of sudoku VMs concurrently running is depicted. The histogram for FQL is shifted to the right compared to the CC6 approach. Only the first bars are larger for CC6, which means there are fewer concurrently running sudoku VMs during longer period of time. By contrast, all the bars for the higher amount of concurrently running sudoku VMs (from 7 onwards) are larger for FQL, which means that most of the time there are more sudoku VMs concurrently running when using FQL. In addition, a summary boxplot is presented in Figure 6.12(c), highlighting a 5% improvement in the number of concurrently running VMs within an already overbooked system. Thanks to that increment in the number of running VMs, the overall utilization also rises as shown by Figure 6.13, where the average utilization along the 3 days experiments is depicted.

Finally, Figure 6.14 shows the average performance per sudoku VM type (Sudoku10,

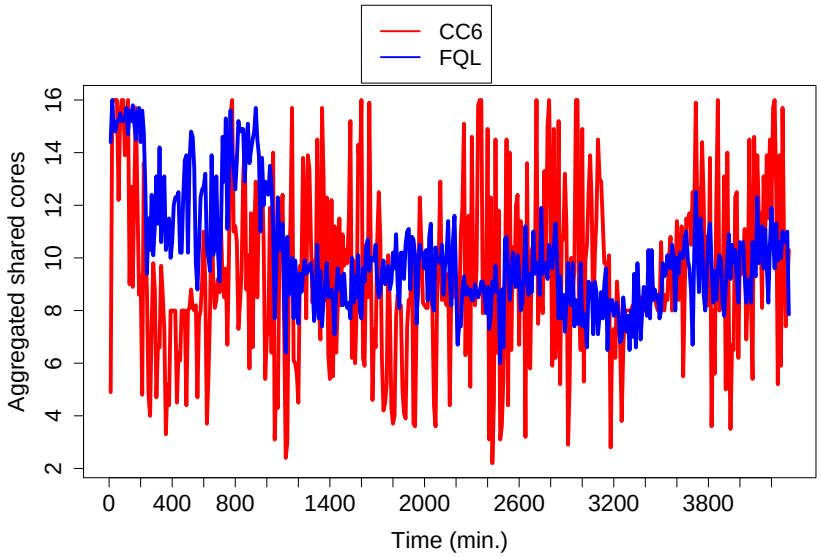
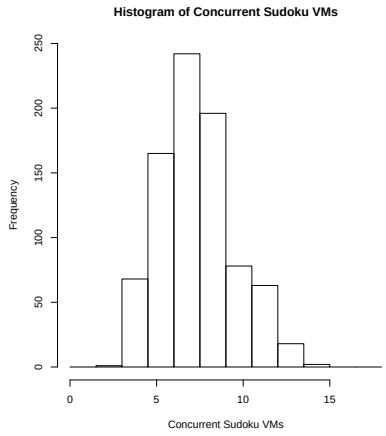
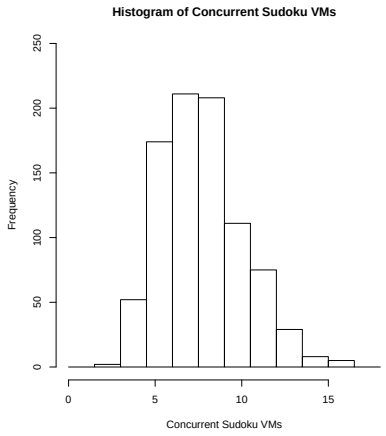


Figure 6.11: Cores sharing over time.

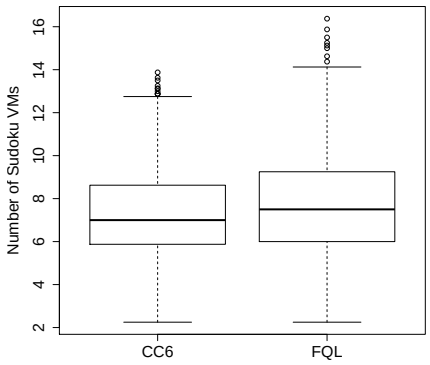
Sudoku20 and SudokuBE) over time. As it can be seen, both Sudoku10 and Sudoku20 fulfill the 11 and 22 solved sudokus per second throughput requirement during all the experiment. Regarding SudokuBE, the number of sudokus solved per second need to be (on average) over 35 sudokus/second to meet application deadlines. This is clearly achieved during the complete experiment, with average performance well above 35 sudokus per second. Moreover, there is statistical proof based on the Wilcoxon test that the average performance of SudokuBE is improved compared to CC6 (p-value: 0.00039). These results highlight the fact that even if more VMs are concurrently running, their performance is not affected at all due to the larger amount of cores being shared between high and low QoS VMs over time, as well as due to fewer and less abrupt fluctuations in the transitions from sharing to not sharing cores, i.e., thanks to the more stable sharing policy.



(a) Histogram CC6



(b) Histogram FQL



(c) Number of concurrent sudoku VMs over-time

Figure 6.12: Sudoku VMs: number of VMs and performance.

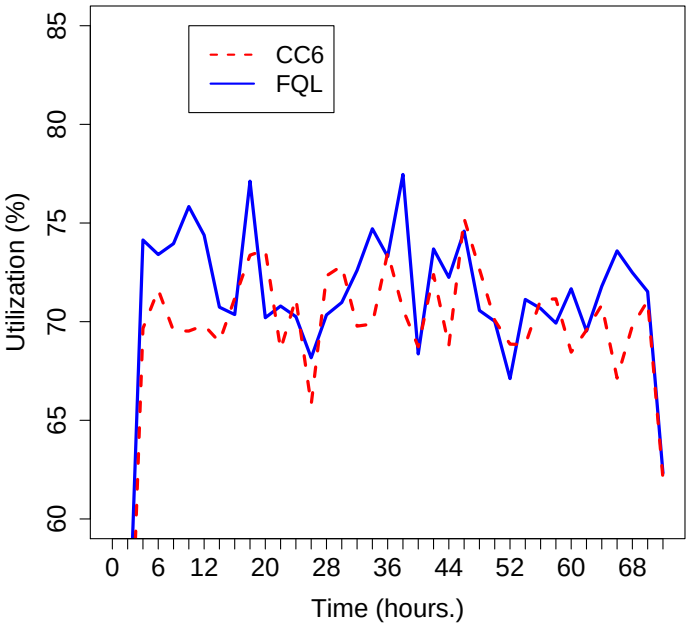


Figure 6.13: Overall utilization.

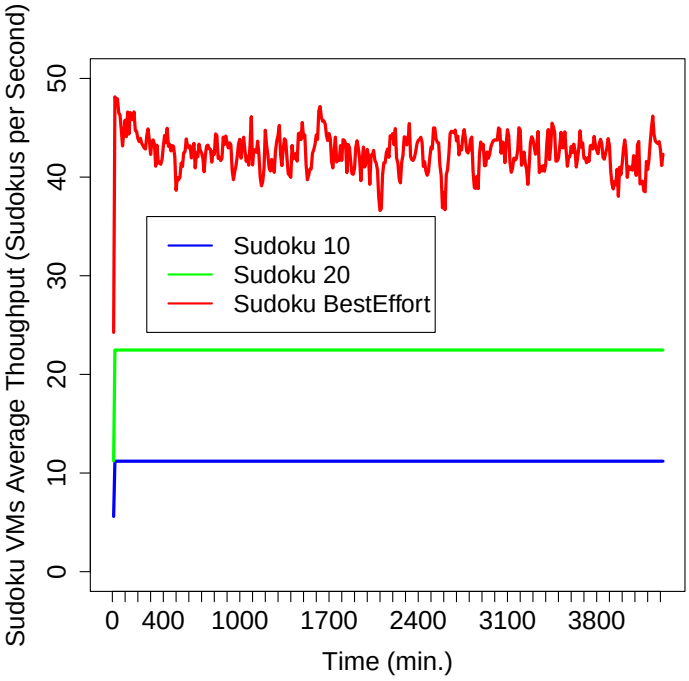


Figure 6.14: Sudoku performance (FQL).

CHAPTER 7

Conclusion

In this chapter, we present the main conclusion of our research by highlighting the significance of this thesis in terms of summarizing the contributions and their implications for the advancement of Energy Efficiency and/or Quality of Service in cloud environments (Section 7.1). The limitations of the thesis are discussed in Section 7.2. Finally, Section 7.3 states the ongoing trends and open topics in related research areas for future research to build upon the contributions presented in this work.

7.1 Summary

This thesis addresses, the problem of resource utilization in Infrastructure as a Service (IaaS) which can effect in one hand on the global energy consumption of the data centers and the quality of service of the running applications on the other hand. In this thesis, we first focus on dynamic VM consolidation strategies which is one of the key mechanisms to design an energy-efficient dynamic cloud resource management system and exploit the application of machine learning algorithms and fuzzy systems, in special reinforcement learning algorithms to tackle the problem of energy efficient resource management in cloud data centers. Then we focus on the scalability problem of dynamic VM consolidation strategies and propose a multi-agent learning and a gossip-based approach for tackling this problem. Finally we address the problem of VMs interfering in overbooked data centers. We propose a fuzzy reinforcement learning algorithm to solve this problem resulting in more efficient resource utilization with less VMs interfering in cloud data centers.

In the remaining of this section, we provide more details in order to summarize the contributions that previously elaborated in Chapter 4 to Chapter 6.

7.1.1 Intelligent Decision Making in Distributed Dynamic Virtual Machine Consolidation

We first propose a novel and intelligent host overloading detection algorithm for the problem of dynamic VM consolidation in cloud data centers by using a Reinforcement Learning algorithm. The proposed algorithm can learn how dynamically tune the CPU threshold value in each physical host node independently in the process of host overloading detection towards energy-performance trade-off optimization in cloud data center. Then we propose a Reinforcement Learning based VM selection policy in the process of dynamic VM consolidation. The proposed algorithm is able to integrate multiple VM selection criteria to benefit from all advantages and possible synergetic contributions of them in long-term learning. In fact, our approach learns how to find an optimal strategy to applying multiple criteria for selecting VMs during a dynamic VM consolidation procedure towards improving the energy-performance trade-off. Finally, we propose a cooperative multi-agent Reinforcement Learning with a Fuzzy Q-learning implementation to solve the problem of physical host nodes management for distributed dynamic virtual machine consolidation in cloud data centers. In our system model, a data center is considered as a multi-agent learning environment, in which learning agents represent decision making engines situated inside each physical host node. In fact, each learning agent, in cooperation with other agents, learns how to make decisions about its corresponding physical host node to be overloaded, and which virtual machine must be selected to migrate such that the global goal of the system (energy-performance trade-off optimization) can be achieved.

7.1.2 Dynamic Virtual Machine Consolidation for Large Scale Cloud Data Centers

We propose a fully decentralized dynamic VM consolidation strategy on top of an unstructured P2P network of physical host nodes to be used in huge data centers. Dynamic virtual machine consolidation can be used by physical host nodes only within the scope of their neighborhoods, allowing the system to be scaled by increasing the number of physical host nodes and virtual machines inside the data center, without any centralized global system knowledge. In our system model the physical host nodes can leave the data center (e.g. because of failure) or be added to the data center (e.g. in order to increase resources) without having any effect on the global performance of the virtual machine consolidation process. Our results illustrate that our proposed strategy can achieve global efficiency in terms of energy and quality of service very close to the optimal centralized approach if the neighborhood size is large enough.

7.1.3 Self-Adaptive Capacity Controller in Overbooked Data Centers

VMs performance can be affected by other co-located VMs, the problem being known as the noisy neighbor. This problem is specially aggravated in overbooked systems as more VMs are competing for the same physical resources. In order to alleviate this

problem, some isolation techniques may be used, such as cgroups or KVM core pining. By using them, different isolation levels can be created even inside a single server, therefore enabling the possibility of providing QoS differentiation by pinning some VMs to specific cores exclusively, while others will share a group of them. These isolation levels may lead to low utilization ratios if high QoS VMs are overprovisioned.

To solve this problem, the extra capacity allocated to the high QoS VMs can be used by other lower QoS VMs, as long as the high QoS VMs are not affected. In this paper we present a Fuzzy Q-Learning based capacity controller that dynamically decides how much capacity each high QoS VM can share with low QoS VMs, i.e., how much the isolation level between them is reduced. Thanks to the mix between reactive and proactive actions, as well as due to the learning over time, a more stable performance is achieved, where more low QoS VMs can be processed without affecting the high QoS VMs performance, consequently making a more efficient use of the available resources and increasing the overall utilization.

7.2 Limitations of this Thesis

Although some significant results of our proposed solutions have been demonstrated, it is important to pinpoint the limitations of this thesis. In this section, we state a number of notable limitations, which highlight observations that are out of scope in our considerations and remain for future research.

- In the problem of dynamic VM consolidation, we just take into account CPU in the definition of QoS requirements. It relies on the fact that CPU is the main resource which usually oversubscribed by cloud providers. This assumption does not mean other resources such as memory, storage, and network bandwidth do not have any impact on the QoS. We believe taking other resources into account can result better performance in the process of dynamic VM consolidation but increase the complexity of the system design.

- As for the proposed solution addressing the problem of host overloading detection for dynamic VM consolidation, the current design of Fuzzy Q-learning (FQL) only supports a set of discreet values for CPU threshold. Although, such design can decrease the complexity of the model but it may lead to lose the accuracy of the system and consequently can cause making inaccurate decisions in some circumstances. Making the FQL's output continues is not a big deal, the fuzzy part of FQL gives this flexibility that the output can be generated in a continues way with a little change in the FQL architecture.

- The proposed Cooperative Multi-Agent Learning solution for the problem of dynamic VM consolidation has been limited to use in data canters which are homogeneous (all the physical host nodes have the same dynamics behavior). This is because, in the proposed cooperative strategy the agents (here physical host nodes) are supposed to use the shared learning knowledge by other agents obtained in the

same environment/situation. We are completely aware that having a cooperative mutli-agent learning solution for dynamic VM consolidation that would be adapted by heterogeneous data centers provides more attraction, however, we believe it still needs more studies and investigation.

- In the scope of this thesis, wherever we addressing the problem of dynamic VM consolidation, the assumption is that the number of Virtual Machines (VM) are fixed in the running data center and all of the VMs are interactive. While, in reality, the number of virtual machines is fluctuating due to either having some batch jobs (batch VMs) which finish their execution in a specific period of time and release the allocated VMs or accepting some new VMs by the admission control. We believe, our proposed strategies for the problem of dynamic VM consolidation are robust with respect to the dynamic change of number of virtual machines in data canter if the fluctuation rate is low.

- As for the proposed fully decentralized solution addressing the problem of scalibilty in dynamic VM consolidation, although the current design of gossip based algorithm can obtain the level of energy efficiency very close to the centralized approach, we didn't take into account any increase in network related energy consumption due to increase in passing messages as required by a P2P protocol.

- The self-adaptive capacity controller proposed in this thesis just ensure the CPU isolation and does not take into account the memory and I/O isolation as memory and I/O were less stressed than the CPU due to the workload characteristics. However, it is important that all hardware capacity dimensions were considered in designing a capacity controller which is applied in an unknown environment with unknown workload characteristics.

- Apart from the challenges addressed throughout the designing process, there are still several technical challenges that should be carefully considered while focusing on Reinforcement Learning (RL) algorithms:(i) RL might need to observe a huge number of episodes (state and action pairs) and state transitions to converge to optimal policies. The sample complexity depends in part on how noisy the immediate rewards and state transitions are and, to a much greater degree, on the compactness of the value function (Q function in Q-learning terminology) representation. (ii) In Fuzzy Q-learning (FQL) structure, the fuzzy system is able to overcome the curse of dimensionality. In addition, it allows us to encapsulate expert knowledge into the learning table resulting in speeding up the learning process. However, the expert knowledge is not always available or sometimes is poor. In this situation, it is necessary to use some techniques [40] to automatically configure the fuzzy sets without relying on previous knowledge about system behaviour. (iii) Generally, in RL methods a certain number of exploration of actions (believed to be sub-optimal) is required to facilitate better learning of policies. As with poor initial policies, exploratory actions can be costly in live systems, therefore, the designer of the system must take care that such costs are not prohibitive. As an example, we

proposed a strategy to tackle this challenge in designing our self-adaptive capacity controller in Section 6.2.2.

7.3 Future Work

As discussed in the previous section, it can be observed that some important issues are out of the scope of our proposed solutions in this thesis. These issues imply the following open research directions.

- RL has a long history, but it has recently been blended with Deep Learning techniques to great impact in applications such as playing video games [61]. We believe that resource management is a complex task specially when we increase the number of input signals to have better decision makings (where the learner needs to keep much more historical knowledge in its memory to act better in the future). In this situation, different "raw" and noisy signals simply can be incorporated as inputs to the deep neural network, and the resultant strategy can be used in such stochastic environment.
- Our proposed cooperative multi-agent learning based dynamic VM consolidation can be modified and integrated with our proposed gossip-based fully decentralized approach. This combination can provide a fully decentralized cooperative multi-agent learning dynamic VM consolidation in which the physical host nodes learn to act locally to fulfill the QoS requirements for the running applications and cooperate to jointly decrease energy consumption on top of a P2P protocol in data center.
- As for the proposed self-adaptive capacity controller in overbooked data center, the current work can be extended with the use of cgroups [37] to ensure not only CPU isolation but also I/O and memory isolation.

Bibliography

[1] AMD-V. `http://www.amd.com/en-gb/solutions/servers/virtualization`, 2012.

[2] Karl J. Åström and Richard M. Murray. *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton University Press, 2008.

[3] Donato Barbagallo, Elisabetta Di Nitto, Daniel J Dubois, and Raffaela Mirandola. A bio-inspired algorithm for energy optimization in a self-organizing data center. In *Self-Organizing Architectures*, pages 127–151. Springer, 2010.

[4] Enda Barrett, Enda Howley, and Jim Duggan. Applying reinforcement learning towards automating resource allocation and application scalability in the cloud. *Concurrency and Computation: Practice and Experience*, 25(12):1656–1674, 2013.

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

[6] Anton Beloglazov, Jemal Abawajy, and Rajkumar Buyya. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future generation computer systems*, 28(5):755–768, 2012.

[7] Anton Beloglazov and Rajkumar Buyya. Adaptive threshold-based approach for energy-efficient consolidation of virtual machines in cloud data centers. In *MGC@ Middleware*, page 4, 2010.

[8] 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.

[9] Anton Beloglazov and Rajkumar Buyya. Managing overloaded hosts for dynamic consolidation of virtual machines in cloud data centers under quality of service constraints. *IEEE TPDS*, 24(7):1366–1379, 2013.

[10] James C Bezdek, Robert Ehrlich, and William Full. Fcm: The fuzzy c-means clustering algorithm. *Computers & Geosciences*, 10(2-3):191–203, 1984.

[11] N. Bobroff, A. Kochut, and K. Beaty. Dynamic placement of virtual machines for managing sla violations. In *10th IFIP/IEEE Intl. Symposium on Integrated Network Management (IM)*, pages 119–128, 2007.

[12] David Breitgand, Zvi Dubitzky, Amir Epstein, Oshrit Feder, Alex Glikson, Inbar Shapira, and Giovanni Toffetti. An adaptive utilization accelerator for virtualized environments. In *IEEE Intl. Conference on Cloud Engineering (IC2E)*, pages 165–174, 2014.

[13] Simone Brienza, Sena Efsun Cebeci, Seyed Saeid Masoumzadeh, Helmut Hlavacs, Öznur Özkasap, and Giuseppe Anastasi. A survey on energy efficiency in p2p systems: File distribution, content streaming, and epidemics. *ACM Computing Surveys (CSUR)*, 48(3):36, 2016.

[14] 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.

[15] Susanta Nanda Tzi-cker Chiueh and Stony Brook. A survey on virtualization technologies. *Rpe Report*, 142, 2005.

[16] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live Migration of Virtual Machines. In *Conference on Symposium on Networked Systems Design and Implementation (NSDI)*, pages 273–286. USENIX Association, 2005.

[17] Vahid Dastjerdi. *QoS-Aware and Semantic-Based Service Coordination for Multi-Cloud Environments*. PhD thesis, University of Melbourne, 2013.

[18] Christina Delimitrou and Christos Kozyrakis. Quasar: Resource-Efficient and QoS-Aware Cluster Management. In *Intl. Conference on Architectural Support for Programming Languages and Operating Systems*, 2014.

[19] Meng Joo Er and Chang Deng. Online tuning of fuzzy inference systems using dynamic fuzzy q-learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 34(3):1478–1489, 2004.

[20] Xiaobo Fan, Wolf-Dietrich Weber, and Luiz Andre Barroso. Power provisioning for a warehouse-sized computer. In *ACM SIGARCH Computer Architecture News*, volume 35, pages 13–23. ACM, 2007.

[21] Eugen Feller. *Autonomic and energy-efficient management of large-scale virtualized data centers*. PhD thesis, Université Rennes 1, 2012.

[22] Eugen Feller, Christine Morin, and Armel Esnault. A case for fully decentralized dynamic vm consolidation in clouds. In *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on*, pages 26–33. IEEE, 2012.

[23] Eugen Feller, Louis Rilling, and Christine Morin. Energy-aware ant colony based workload placement in clouds. In *Proceedings of the 2011 IEEE/ACM 12th International Conference on Grid Computing*, pages 26–33. IEEE Computer Society, 2011.

[24] Md Hasanul Ferdaus, M Manzur Murshed, Rodrigo N Calheiros, and Rajkumar Buyya. Virtual machine consolidation in cloud data centers using aco metaheuristic. In *Euro-Par*, pages 306–317, 2014.

[25] Ian Foster, Yong Zhao, Ioan Raicu, and Shiyong Lu. Cloud Computing and Grid Computing 360-Degree Compared. In *Grid Computing Environments Workshop (GCE)*, pages 1–10. IEEE, 2008.

[26] Armando Fox, Rean Griffith, Anthony Joseph, Randy Katz, Andrew Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, and Ion Stoica. Above The Clouds: A Berkeley View of Cloud Computing. *University of California (UCB/EECS)*, 28:13, 2009.

[27] Gartner. Worldwide Public Cloud Services Market to Total $131 Billion. `http://www.gartner.com/newsroom/id/2352816/`, 2013.

[28] Gartner. Worldwide Public Cloud Services Market Is Forecast to Reach $204 Billion in 2016. `http://www.gartner.com/newsroom/id/3188817/`, 2016.

[29] Rahul Ghosh and Vijay K. Naik. Biting Off Safely More Than You Can Chew: Predictive Analytics for Resource Over-Commit in IaaS Cloud. In *5th Intl. Conference on Cloud Computing*, pages 25–32, 2012.

[30] P. Y. Glorennec and L. Jouffe. Fuzzy q-learning. In *Intl. Conference on Fuzzy Systems*, volume 2, pages 659–662, 1997.

[31] Daniel Gmach, Jerry Rolia, Ludmila Cherkasova, Guillaume Belrose, Tom Turicchi, and Alfons Kemper. An integrated approach to resource pool management: Policies, efficiency and quality metrics. In *Dependable Systems and Networks With FTCS and DCC, 2008. DSN 2008. IEEE International Conference on*, pages 326–335. IEEE, 2008.

[32] Jianhua Gu, Jinhua Hu, Tianhai Zhao, and Guofei Sun. A new resource scheduling strategy based on genetic algorithm in cloud computing environment. *Journal of Computers*, 7(1):42–52, 2012.

[33] Simon S Haykin. *Neural networks: a comprehensive foundation*. Tsinghua University Press, 2001.

[34] Michael R Hines and Kartik Gopalan. Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning. In *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, pages 51–60. ACM, 2009.

[35] Helmut Hlavacs and Thomas Treutner. Genetic algorithms for energy efficient virtualized data centers. In *Network and service management (cnsm), 2012 8th international conference and 2012 workshop on systems virtualiztion management (svm)*, pages 422–429. IEEE, 2012.

[36] Rachel Householder, Scott Arnold, and Robert Green. On cloud-based oversubscription. *International Journal of Engineering Trends and Technology (IJETT)*, 8(8):425–431, 2014.

[37] IBM Knowledge Centre. Kernel Virtual Machine (KVM): Best practices for KVM. Web page at `http://www-01.ibm.com/support/knowledgecenter/linuxonibm/liaat/liaatbestpractices_pdf.pdf?lang=en`.

[38] Pooyan Jamshidi, Amir M Sharifloo, Claus Pahl, Andreas Metzger, and Giovani Estrada. Self-learning cloud controllers: Fuzzy q-learning for knowledge evolution. In *Cloud and Autonomic Computing (ICCAC), 2015 International Conference on*, pages 208–211. IEEE, 2015.

[39] Márk Jelasity. Gossip. In *Self-organising software*, pages 139–162. Springer, 2011.

[40] Chia-Feng Juang. Combination of online clustering and q-value based ga for reinforcement fuzzy system design. *IEEE Transactions on Fuzzy Systems*, 13(3):289–302, 2005.

[41] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[42] Maria Kihl, Erik Elmroth, Johan Tordsson, Karl-Erik Årzén, and Anders Robertsson. The Challenge of Cloud Control. In *Feedback Computing*, 2013.

[43] Cristian Klein, Martina Maggio, Karl-Erik Årzén, and Francisco Hernández-Rodríguez. Brownout: Building more robust cloud applications. In *36th Intl. Conference on Software Engineering*, 2014.

[44] Geoff Koch. Discovering multi-core: Extending the benefits of mooreâĂŹs law. *Technology*, 1, 2005.

[45] Jonathan G Koomey et al. Estimating total power consumption by servers in the us and the world, 2007.

[46] Ewnetu Bayuh Lakew. *Autonomous Cloud Resource Provisioning: Accounting, Allocation, and Performance Control*. PhD thesis, Umea University, Department of Computing Science, 2015.

[47] Jacob Leverich and Christos Kozyrakis. Reconciling high server utilization and sub-millisecond quality-of-service. In *9th European Conference on Computer Systems (EuroSys)*, 2014.

[48] Huan Liu. A measurement study of server utilization in public clouds. In *IEEE 9th Intl. Conference on Dependable, Autonomic and Secure Computing (DASC)*, pages 435–442, 2011.

[49] David Lo, Liqun Cheng, Rama Govindaraju, Parthasarathy Ranganathan, and Christos Kozyrakis. Heracles: Improving resource efficiency at scale. In *Annual International Symposium on Computer Architecture (ISCA)*, pages 450–462, 2015.

[50] Zaigham Mahmood. Cloud Computing: Characteristics and Deployment Approaches. In *International Conference on Computer and Information Technology (CIT)*, pages 121–126. IEEE, 2011.

[51] Dan Marinescu and Reinhold Kroger. State of the art in autonomic computing and virtualization. *Distributed Systems Lab, Wiesbaden University of Applied Sciences*, pages 1–24, 2007.

[52] Moreno Marzolla, Ozalp Babaoglu, and Fabio Panzieri. Server consolidation in clouds through gossiping. In *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2011 IEEE International Symposium on a*, pages 1–6. IEEE, 2011.

[53] Seyed Saeid Masoumzadeh and Helmut Hlavacs. Integrating vm selection criteria in distributed dynamic vm consolidation using fuzzy q-learning. In *Network and Service Management (CNSM), 2013 9th International Conference on*, pages 332–338. IEEE, 2013.

[54] Seyed Saeid Masoumzadeh and Helmut Hlavacs. An intelligent and adaptive threshold-based schema for energy and performance efficient dynamic vm consolidation. In *European Conference on Energy Efficiency in Large Scale Distributed Systems*, pages 85–97. Springer, 2013.

[55] Seyed Saeid Masoumzadeh and Helmut Hlavacs. A cooperative multi agent learning approach to manage physical host nodes for dynamic consolidation of virtual machines. In *Network Cloud Computing and Applications (NCCA), 2015 IEEE Fourth Symposium on*, pages 43–50. IEEE, 2015.

[56] Seyed Saeid Masoumzadeh and Helmut Hlavacs. Dynamic virtual machine consolidation: A multi agent learning approach. In *Autonomic Computing (ICAC), 2015 IEEE International Conference on*, pages 161–162. IEEE, 2015.

[57] Seyed Saeid Masoumzadeh and Helmut Hlavacs. A gossip-based dynamic virtual machine consolidation strategy for large-scale cloud data centers. In *Proceedings of the Third International Workshop on Adaptive Resource Management and Scheduling for Cloud Computing*, pages 28–34. ACM, 2016.

[58] Seyed Saeid Masoumzadeh, Helmut Hlavacs, and Luis Tomás. A self-adaptive performance-aware capacity controller in overbooked datacenters. In *Cloud and Autonomic Computing (ICCAC), 2016 International Conference on*, pages 12–23. IEEE, 2016.

[59] Peter Mell and Timothy Grance. The NIST Definition of Cloud Computing. *National Institute of Standards and Technology Special Publication*, pages 800–145, 2014.

[60] Lauri Minas and Brad Ellison. *Energy efficiency for information technology: How to reduce power consumption in servers and data centers*. Intel Press, 2009.

[61] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[62] Alberto Montresor and Márk Jelasity. PeerSim: A scalable P2P simulator. In *Proc. of the 9th Int. Conference on Peer-to-Peer (P2P'09)*, pages 99–100, Seattle, WA, September 2009.

[63] Alireza F Naeeni. Advanced multi-agent fuzzy reinforcement learning. Master's thesis, Dalarna University, Dalarna, 2004.

[64] Ripal Nathuji, Aman Kansal, and Alireza Ghaffarkhah. Q-clouds: Managing performance interference effects for qos-aware clouds. In *Eurosys*, 2010.

[65] Ripal Nathuji and Karsten Schwan. Virtualpower: coordinated power management in virtualized enterprise systems. In *ACM SIGOPS Operating Systems Review*, volume 41, pages 265–278. ACM, 2007.

[66] Page view statistics for Wikimedia projects. Web page at `http://dumps.wikimedia.org/other/pagecounts-raw/`, Visited 2013-03-13.

[67] Liviu Panait and Sean Luke. Cooperative multi-agent learning: The state of the art. *Autonomous agents and multi-agent systems*, 11(3):387–434, 2005.

[68] KyoungSoo Park and Vivek S Pai. Comon: a mostly-scalable monitoring system for planetlab. *ACM SIGOPS Operating Systems Review*, 40(1):65–74, 2006.

[69] Dana Petcu. Multi-Cloud: Expectations and Current Approaches. In *International Workshop on Multi-cloud Applications and Federated Clouds*, pages 1–6. ACM, 2013.

[70] Dana Petcu. Consuming Resources and Services From Multiple Clouds. *Journal of Grid Computing*, pages 1–25, 2014.

[71] Rackspace. Rackspace Hosting Reports Second Quarter 2012 Results. `https://blog.rackspace.com/rackspace-hosting-reports-second-quarter-2012-results`, 2012.

[72] Jia Rao, Xiangping Bu, Cheng-Zhong Xu, Leyi Wang, and George Yin. Vconf: a reinforcement learning approach to virtual machines auto-configuration. In *Proceedings of the 6th international conference on Autonomic computing*, pages 137–146. ACM, 2009.

[73] Charles Reiss, Alexey Tumanov, Gregory R. Ganger, Randy H. Katz, and Michael A. Kozuch. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *3rd ACM Symposium on Cloud Computing (SoCC)*, 2012.

[74] RUBBoS: Bulletin Board Benchmark. Web page at `http://jmob.ow2.org/rubbos.html`, Visited 2013-11-4.

[75] RUBiS: Rice University Bidding System. Web page at `http://rubis.ow2.org/`, Visited 2013-11-4.

[76] Laura Savu. Cloud Computing: Deployment Models, Delivery Models, Risks and Research Challenges. In *International Conference on Computer and Management (CAMAN)*, 2011.

[77] Michael D Schroeder and Jerome H Saltzer. A hardware architecture for implementing protection rings. *Communications of the ACM*, 15(3):157–170, 1972.

[78] Jyothi Sekhar, Getzi Jeba, and S Durga. A Survey on Energy Efficient Server Consolidation through VM Live Migration. *International Journal of Advances in Engineering and Technology*, 5(1):515–525, 2012.

[79] Richard L Sites, Anton Chernoff, Matthew B Kirk, Maurice P Marks, and Scott G Robinson. Binary translation. *Communications of the ACM*, 36(2):69–81, 1993.

[80] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[81] Gerald Tesauro. Reinforcement learning in autonomic computing: A manifesto and case studies. *IEEE Internet Computing*, 11(1), 2007.

[82] Michel Tokic and Günther Palm. Value-difference based exploration: adaptive control between epsilon-greedy and softmax. In *KI 2011: Advances in Artificial Intelligence*, pages 335–346. Springer, 2011.

[83] L. Tomás and J. Tordsson. An autonomic approach to risk-aware data center overbooking. *IEEE Transactions on Cloud Computing*, 2(3):292–305, 2014.

[84] Luis Tomás, Cristian Klein, Johan Tordsson, and Francisco Hernández-Rodríguez. The straw that broke the camel's back: safe cloud overbooking with application brownout. In *International Conference on Cloud and Autonomic Computing Conference (ICCAC)*, 2014.

[85] Luis Tomás, Seyed Saeid Masoumzadeh, and Helmut Hlavacs. Self-adaptive capacity controller: A reinforcement learning approach. In *Autonomic Computing (ICAC), 2016 IEEE International Conference on*, pages 233–234. IEEE, 2016.

[86] Luis Tomás and Johan Tordsson. Cloudy with a chance of load spikes: Admission control with fuzzy risk assessments. In *Proc. of 6th IEEE/ACM Intl. Conference on Utility and Cloud Computing*, pages 155–162, 2013.

[87] Luis Tomás and Johan Tordsson. Improving Cloud Infrastructure Utilization through Overbooking. In *Cloud and Autonomic Computing Conference (CAC)*. ACM, 2013.

[88] Luis Tomás and Johan Tordsson. Cloud Service Differentiation in Overbooked Data Centers. In *IEEE Conference on Utility and Cloud Computing (UCC)*, 2014.

[89] Luis Tomás, Carlos Vázquez, Johan Tordsson, and Ginés Moreno. Reducing noisy-neighbor impact with a fuzzy affinity-aware scheduler. In *International Conference on Cloud and Autonomic Computing (ICCAC)*, pages 33–44, 2015.

[90] Rich Uhlig, Gil Neiger, Dion Rodgers, Amy L Santoni, Fernando CM Martins, Andrew V Anderson, Steven M Bennett, Alain Kagi, Felix H Leung, and Larry Smith. Intel virtualization technology. *Computer*, 38(5):48–56, 2005.

[91] Bhuvan Urgaonkar, Prashant Shenoy, and Timothy Roscoe. Resource overbooking and application profiling in shared hosting platforms. In *OSDI*, pages 239–254, 2002.

[92] William Voorsluys, James Broberg, Srikumar Venugopal, and Rajkumar Buyya. Cost of virtual machine live migration in clouds: A performance evaluation. *CloudCom*, 9:254–265, 2009.

[93] Wenting Wang, Haopeng Chen, and Xi Chen. An Availability-Aware Virtual Machine Placement Approach for Dynamic Scaling of Cloud Applications. In *International Conference on Autonomic and Trusted Computing (UIC/ATC)*, pages 509–516. IEEE, 2012.

[94] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[95] Joshua White and Adam Pilbeam. A survey of virtualization technologies with performance testing. *arXiv preprint arXiv:1010.3233*, 2010.

[96] Frank Wilcoxon. Individual Comparisons by Ranking Methods. *Biometrics Bulletin*, 1(6):80–83, 1945.

[97] Timothy Wood, Prashant Shenoy, Arun Venkataramani, and Mazin Yousif. Sandpiper: Black-box and gray-box resource management for virtual machines. *Computer Networks*, 53(17):2923–2938, 2009.

[98] Jing Xu, Ming Zhao, Jose Fortes, Robert Carpenter, and Mazin Yousif. On the use of fuzzy modeling in virtualized data center management. In *Autonomic Computing, 2007. ICAC'07. Fourth International Conference on*, pages 25–25. IEEE, 2007.

[99] Yongming Yang, Yantao Tian, and Hao Mei. Cooperative Q Learning Based on Blackboard Architecture. In *2007 International Conference on Computational Intelligence and Security Workshops (CISW 2007)*, pages 224–227. IEEE, December 2007.

[100] Xiao Zhang, Eric Tune, Robert Hagmann, Rohit Jnagal, Vrigo Gokhale, and John Wilkes. CPI2: CPU performance isolation for shared compute clusters. In *SIGOPS European Conference on Computer Systems (EuroSys)*, pages 379–391, 2013.