Università degli Studi di Pavia
Facoltà di Ingegneria

DIPARTIMENTO DI INGEGNERIA INDUSTRIALE E DELL'INFORMAZIONE
PhD in Electronics, Computer Science and Electrical Engineering

PhD Thesis
XXX cycle

Workload characterization and autoscaling in Cloud environments

Candidate:
Momin I.M. Tabash
Matricola 432910

Supervisor:
Prof. Maria Carla Calzarossa

Co-supervisors:
Prof. Luisa Massari
Prof. Daniele Tessera

Academic Year 2016/2017

Acknowledgements

This thesis is the result of hard work and commitment to achieve one of my noble goals in life. First of all, my PhD journey was not a straight path: I have experienced a lot of changes in my personal attitudes between happiness and struggling. I felt quite lucky to work with a great team in the Performance Evaluation Lab. I should take this opportunity to express my sincere gratitude to my supervisors Prof. Maria Carla Calzarossa, Prof. Luisa Massari and Prof. Daniele Tessera for their excellent supervision, feedback and guidance. Without their efforts this work would have been impossible. Many thanks to them for giving me different opportunities to attend summer schools and international conferences and for their encouragement.

On a more personal level, I would like to thank my parents; my father Ismail and my mother Fathiya for their love and support. Last but not least, I will not forget my brothers and sisters for their continuous support demonstrated in many ways. Special thanks to everybody engaged in supporting me during my PhD journey.

Thank you all!

Abstract

Cloud computing is becoming a successful key factor in many types of business, because it enables an efficient model for resource provisioning. Even though resources can be provisioned on demand, they need to adapt quickly and in a seamless way to the workload intensity and characteristics and satisfy at the same time the desired performance levels. Autoscaling policies are devised for these purposes. In this thesis work, we apply a state-of-the-art reactive autoscaling policy to assess the effects of deploying the HTTP/2 server push mechanism in Cloud environments. A simulation environment based on the CloudSim simulation toolkit has been designed and developed to exploit a Web workload on a realistic Cloud infrastructure. Workload characterization based on measurements collected on a real Web server has been carried out to derive workload models to be used for workload description in the simulation experiments. These experiments have shown that the autoscaling mechanism is beneficial for Web servers even though pushing a large number of objects might lead to server overload.

List of Publications

1. M. C. Calzarossa, M. L. D. Vedova, L. Massari, D. Petcu, M. I. M. Tabash, and D. Tessera, "Workloads in the Clouds," in Principles of Performance and Reliability Modeling and Evaluation, Springer Series in Reliability Engineering, L. Fiondella and A. Puliafito, Eds. Springer, 2016, vol. 7084, pp. 552–550.

2. M. C. Calzarossa, L. Massari, M. I. M. Tabash, and D. Tessera, "Cloud autoscaling for HTTP/2 workloads," in Proc. of the 3rd International Conference on Cloud Computing Technologies and Applications (CloudTech'17), 2017.

After the paper "Cloud autoscaling for HTTP/2 workloads" has been presented at the CloudTech'17 conference, we have been invited to submit an extended version of the paper to a special issue "Cloud Computing, IoT, and Big Data: Technologies and Applications" of the journal "Concurrency and Computation: Practice and Experience" with guest editors: Mostapha Zbakh, Mohammed Bakhouya, Mohamed Essaaidi and Pierre Manneback".

Table of Contents

List of Figures

List of Tables

Chapter 1

INTRODUCTION

1.1 Motivation

Nowadays Cloud computing is becoming a successful key factor in many types of businesses because it enables an efficient model for resource provisioning and allows customers to significant reduce the upfront costs for acquiring hardware and software resources as well as the costs associated with the deployment, management and maintenance of these resources. In addition, Cloud computing enables providers to offer unlimited virtualized resources based on an on demand model.

Cloud computing introduces a good level of flexibility, by allowing customers to pay only for the actual use of resources according to a "pay per use" model. This model offers numerous advantages and allows customers to benefit from increased resource availability, elasticity and improved fault tolerance.

Cloud environments are very suitable for hosting applications and services whose workload intensity rapidly changes. Provisioned resources have to cope with workload fluctuations. Wherever these resources are not sufficient to handle workload peaks, the Quality of Service (QoS) is negatively affected. On the contrary, provisioning for peak workloads results in plenty of resources being underutilized.

Provisioning resources to the varying workload demands is a complex process. Therefore, for increasing the efficiency and reducing the cost of deploying services in Cloud environments, it is necessary to devise dynamic resource provisioning policies and automatically adjust the allocated resources to workload intensity and characterization. This objective can be achieved by autoscaling policies that add or release resources to satisfy the QoS constraints according to their actual load. Autoscaling policies enable Cloud customers to provision resources when the workload demand increases and deprovision unused resources when the demand decreases. Unexpected workload fluctuations should also be handled seamlessly. The workload is processed by the Virtual Machines. A Virtual Machine (VM) is a software artifact that executes other software as if it was running on a physical resource directly.

Typical autoscaling scenarios are illustrated in Figures 1.1 and 1.2. As can be seen, a Web application is deployed in a Cloud environment consisting of three pools of resources. The Web application provides the service to the Cloud users. Each pool of resources consists of different number of Virtual Machines (VMs). The autoscaling policy is configured and monitors the Web application performance. The resources requirements for this Web application are three VMs allocated in the pool of resources 1

and two VMs are allocated in each of the other two pools. New user arrival cause an increase in requests and an overutilization of the available allocated resources. Thus, the autoscaling policy decides to add some resources to each pool (see Figure 1.1). In particular, two VMs are provisioned to computing resources 1 and one VM is provisioned to each of the other two pools. On the contrary, the autoscaling policy deprovisions some resources from each pool when the number of users decreases (see Fig. 1.2). As can be seen, the autoscaling policy deprovisions seven VMs in total.

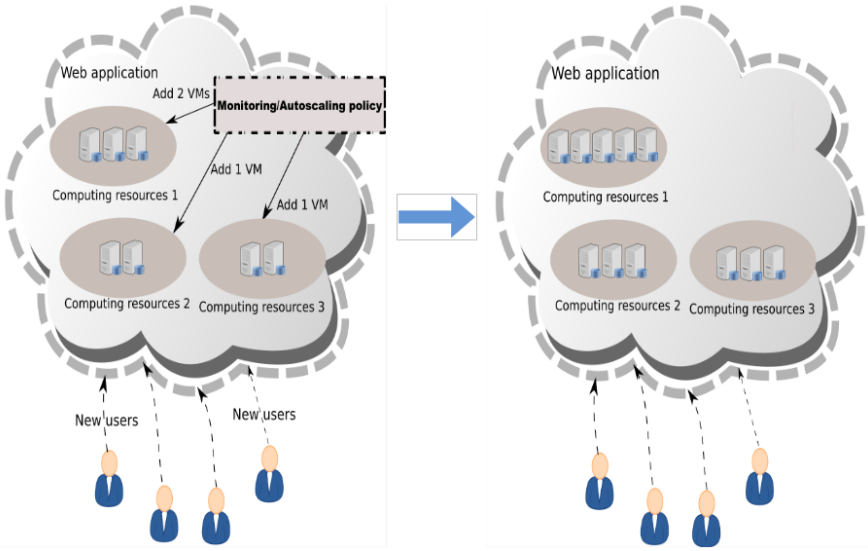


Figure 1.1: Scaling up the computing resources by adding new VMs

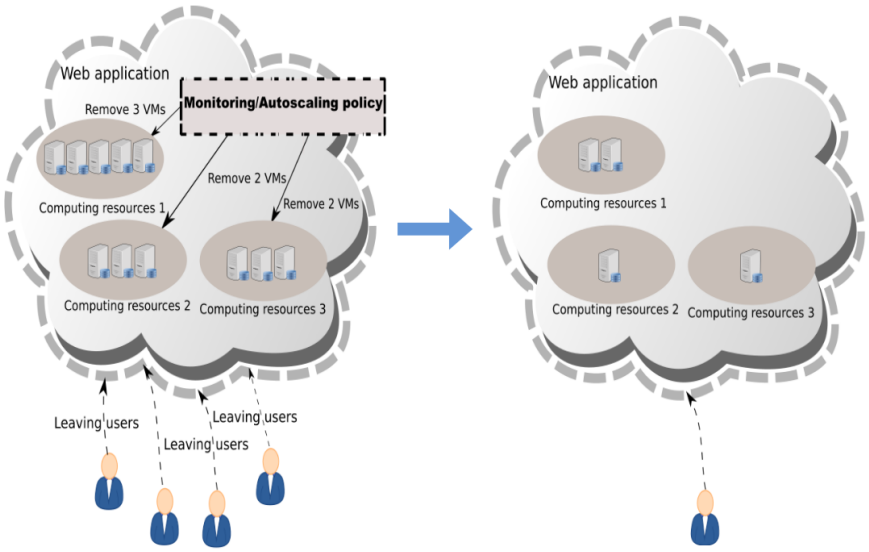


Figure 1.2: Scaling down the computing resources by removing VMs

1.2 Cloud computing

Cloud computing paradigm has recently received increased attention from academia and industry. According to a forecast from Cisco [1], more than 92% of the workloads will be processed in Cloud data centers by 2020. Gartner[1] predicts an increase in the infrastructure compute service space as the Cloud adoption becomes mainstream.
Resources can be provisioned in response to workload fluctuations avoiding resource under-utilization or over-utilization, while maintaining the desired of QoS.
Cloud providers must ensure the compliance of the Service Level Agreement (SLA). We recall that an SLA is a formal agreement between a Cloud provider and its customers, defining in quantitative terms the functional and non-functional aspects of the service being offered [2].
The five essential characteristics of Cloud computing identified by NIST [3] are summarized as follows:

- On demand provisioning of resources.
- Broad network access able to handle of user requests.
- Resource pooling between multiple Cloud customers with different resource requirements.
- Rapid elasticity, that is, the ability to vary resources according to the load.
- Transparent resource usage monitoring.

1.2.1 Cloud service models

According to NIST, three standard models are associated with Cloud computing, namely:

- **Software as a Service (SaaS)** represents a software distribution model in which customers use applications running on a Cloud provider infrastructure. Customers can access the services from different devices, but they do not need to manage or control the Cloud infrastructure.
- **Platform as a Service (PaaS)** represents a Cloud service model where a Cloud provider provides customers with a platform and the ability to run, manage and

[1] http://www.gartner.com/technology/home.jsp

develop applications and services. A customer has full control of the applications being deployed but does not manage or control the underlying infrastructure.

- **Infrastructure as a Service (IaaS)** represents a Cloud service model where resources (e.g., computing power, storage, networks) are provided as a service. A customer has full control of storage, applications and operating system but limited control of the networking components (e.g., firewalls). In detail, customers are provided with storage, networks and other resources where they can deploy and run software, components including an operating system and applications.

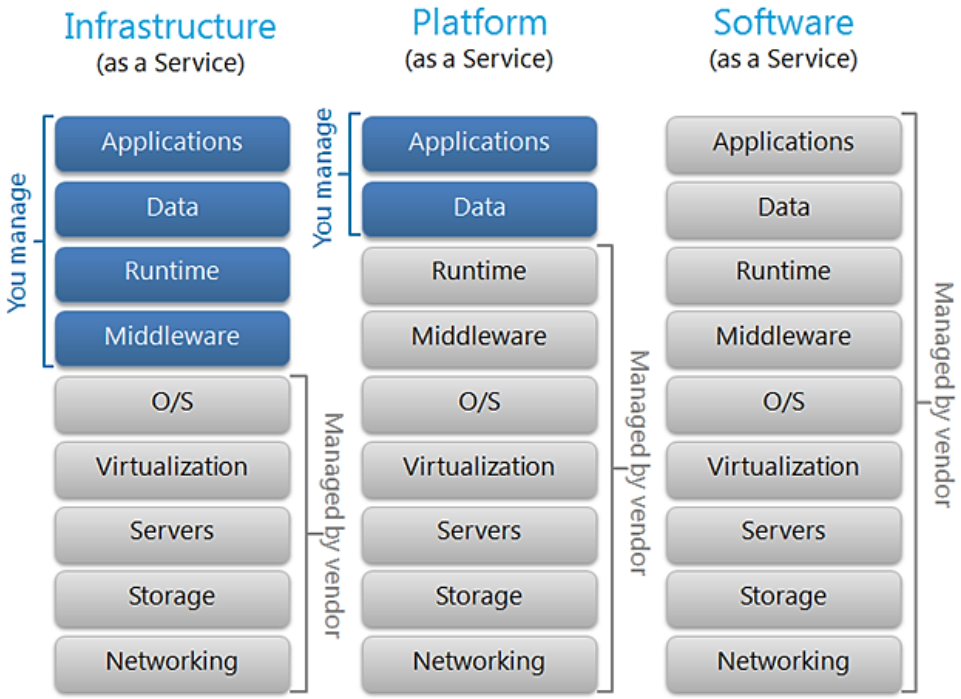


Figure 1.3: Layered architecture of three Cloud service models

Figure 1.3 shows a layered Cloud service model that consists of different components managed according to the privileges granted by each Cloud service model. Each model assigns certain responsibilities to the Cloud provider and allows customers to focus more on their business.

With the *SaaS model*, the customer consumes only applications running on the Cloud infrastructure, and accessible from various devices. The Cloud provider manages everything (i.e., infrastructure, load balancers and firewalls, operating systems and runtime environments). Browser-based interfaces can be used to access and customize services.

With the *PaaS model*, customers access an operating system and additional services that allow them to run their own applications. Customers are not aware of low level components that are managed directly by the provider.
With the *IaaS model*, customers can manage all the components of the Cloud infrastructure. In addition, they can deploy applications and install operating system images and additional software. In this model, customers have the responsibility to patch/update/maintain all software components.

1.2.2 Virtualization

Virtualization – one a basic technology in Cloud environments – introduces a layer of abstraction between physical hardware resources and operating systems. The technology behind virtualization – known as Virtual Machine Monitor (VMM) or hypervisor – separates the compute environments from the actual physical infrastructure. A Virtual Machine is a software artifact that executes other software as if it was running on a physical resource directly. Virtualization enables multiple virtual machines to share physical resources [4].
Traditionally, the operating systems are responsible for managing the allocation of physical resources (i.e., CPU, memory, disk and network bandwidth). On the contrary, hypervisors manage the execution of operating systems by booting, suspending or shutting down VMs as required. Some hypervisors also support replication and migration of Virtual Machines without interruption. Examples of the hypervisors are Xen[2], QEMU[3] and VMWare[4].

[2]https://www.xenproject.org/
[3]https://www.qemu.org/
[4]https://www.vmware.com/

1.3 Resource Provisioning

The resource allocation problem is a major issue in distributed computing. In Cloud environments where customers request a variety of services characterized by dynamically changing requirements, the problem is even more complex.

Traditional allocation solutions based on static resource provisioning are not suitable for Cloud environments since they lead to poor performance. In fact, over-provisioning resources to meet potential demand peaks can result in significant costs and unused capacities as depicted in Figure 1.4(a). In contrast, planning resources for the average load may lead to overload conditions as shown in Figure 1.4(b). To avoid these issues, Cloud resources must be dynamically adjusted in response to demand fluctuations as depicted Figure 1.4(c). This emphasizes the need of sophisticated resource provisioning mechanisms.

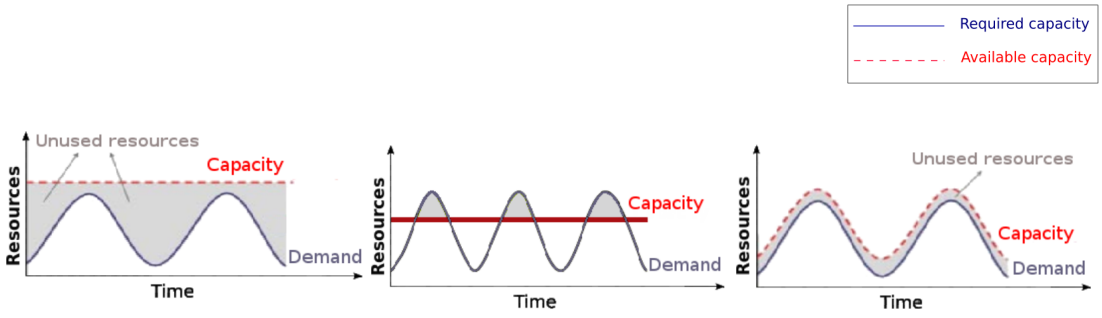


Figure 1.4: Effects of the static resource provisioning:(over-provisioning (a)) and (under-provisioning (b)) and (dynamic resource provisioning (c)). The solid curves represent the required capacity, while dashed curves represent the available capacity

Autoscaling has received significant attention to minimize the amount of allocated resources without violating QoS constraints. Autoscaling enables Cloud computing infrastructure to be elastic and highly available. Herbst et al. [5] explain the relation between scalability and elasticity. Scalability is the ability of a system to sustain increasing workloads by making use of additional resources. In contrast the elasticity, *"is the degree to which a system is able to adapt to workload changes by provisioning and de-provisioning resources in an autonomic manner, such that at each point in time the available resources match the current demand as closely as possible"*. In other words, the elasticity covers how quickly a system can respond to fluctuating resource demands.

Thus, autoscaling techniques enable elasticity.
Autoscaling automates the allocation/deallocation of the resources to match the application demands. Hence, the key feature of autoscaling is to allow dynamic provisioning of virtualized resources in response to the workload variability [5, 6].

1.4 Thesis contribution

The main contributions of this thesis work are summarized as follows:

- Development of detailed models of a Web workload (see Chapter 3);
- Design and development of a simulation environment based on the CloudSim simulation toolkit that enables workload generation and performance monitoring and the deployment of different autoscaling policies (see Chapter 5);
- Implementation of reactive autoscaling policies into the simulation environment (see Section 5.3);
- Testing of the developed environment for the analysis of the performance of the HTTP/2 server push mechanism deployed in Cloud environments (see Chapter 6).

1.5 Thesis organization

This thesis is organized as follows. In Chapter 2 the state of the art in the field of autoscaling policies and Cloud simulator tools is presented. Workload characterization and modeling are addressed in Chapter 3. The simulation environment that has been designed and developed is explained in Chapters 4 and 5. The setup of the experimental environment and the results of the tests performed to assess the benefits of autoscaling policies for the HTTP/2 server push mechanism are described in Chapter 6. Finally, some conclusions and open research issues are given in Chapter 7.

Chapter 2

STATE OF THE ART

This chapter presents the state of the art in the framework autoscaling policies for cloud environments and discusses open challenges as well as strengths and weaknesses of the various policies. In addition, the characteristics of the main Cloud simulators developed in the research environment are described.

2.1 Autoscaling techniques

Autoscaling policies are designed with particular goals, focusing on several application architectures or Cloud environment offering different capabilities (see, e.g., [7, 8]). Policies differ in their scaling indicators and in their final goals/evaluation criteria (e.g., prediction accuracy, SLA requirements). A policy can be designed to cope with load conditions (e.g., workload bursts, diurnal patterns), and focus on various aspects, such as workload prediction, adaptivity to dynamically varying workload characteristics and oscillation mitigations.

Autoscaling policies need to monitor performance indicators to determine whether and when scaling operations need to be triggered. In literature various studies [9, 10, 11, 12] classify autoscaling polices into *reactive policies* and *proactive*. Reactive policies react to the actual load and allocate new resources in response to the workload intensity. Proactive policies attempt to predict the load. These predictions are usually performed at fixed intervals or when big surges of traffic are expected. The main advantage of these policies is that they can predict unplanned load spikes. The accuracy of the predictions depends on the adopted approach. Two approaches, namely, Moving Window Average and Linear Regression, are often used for making predictions.
Moving Window Average is a simple method to smooth sequential data and is typically applied to time-based data, such as the computation of VMs utilization. The Moving Average aggregation will slide a window across the data and emit the average value of that window. For example, given the data [1, 2, 3, 4, 5, 6, 7], the calculation of a simple moving average with windows size of 5 is as follows:

- $(1 + 2 + 3 + 4 + 5)/5 = 3$
- $(2 + 3 + 4 + 5 + 6)/5 = 4$
- $(3 + 4 + 5 + 6 + 7)/5 = 5$

This method helps to detect workload fluctuations.

Linear Regression is often used to build a model of the relationships between a dependent variable (output) and independent variable or variables (input). If there is only one independent variable, the method is called simple linear regression, otherwise it is called multi linear regression [13].

The problem of scaling the Cloud resource has been extensively studied and many autoscaling techniques have been proposed by researchers, such as:

- Rule-based;
- Reinforcement Learning;
- Control Theory;
- Queuing Theory;
- Time series analysis.

Various autoscaling policies apply these techniques: a brief overview of these techniques is provided in what follows.

2.1.1 Rule based approach

One of the most common approaches applied to devise autoscaling policies is the rule based approach [14]. This approach is very simple to setup, manage, and implement, since scaling decisions are triggered according to some defined rules. These rules usually rely on measurements of performance metrics, such as CPU load, memory usage, I/O operations, throughput, response time. Two thresholds are ofter set, namely, an upper threshold for scaling up and a lower threshold for scaling down.
Stability in trigging actions is also required for avoiding oscillation, that is, opposite actions frequently performed in short time (i.e., allocating new resources and then deallocating resources or vice versa). For example, when the CPU utilization exceeds the upper threshold, one or more VMs will be allocated to the pool of resources to reduce the utilization. Suppose that this drops the utilization to less than the lower threshold, as a consequence one or more of the allocated VMs will be deallocated, thus causing oscillation. Therefore, it is effective to avoid frequent scaling actions. These actions have to consider the cooling time, that is, a common solution adopted to tackle oscillation and it is a minimum time has to be set between two opposite scaling actions. The given time avoids new scaling actions and enables the new added VMs to settle into

running mode. This prevents the Cloud from reacting too quickly and this situation should be prevented as it results in resource wastage and SLA violation.
Dutreilh et al. [15] have proposed a threshold-based technique. Two thresholds are set to dynamically adapt the resources of virtualized applications by allocating or deallocating VMs. The implementation of the technique relies on SLA metrics, such average response time to meet the Service Level Objectives. SLO is a key element of SLA between the Cloud provider and Cloud customer. SLOs are agreed upon as a means of measuring the performance of Cloud services itself. These metrics measures services, such as, average response time, accounting, security and they can be used to negotiate/monitor the Cloud customer requirements based on the agreed SLA. The proposed policy considers five parameters: an upper threshold, a lower threshold, a fixed amount of Virtual Machines to be allocated or deallocated, and two parameters for inertia durations, one for scaling up and the other for scaling down. In [16], the authors present a lightweight approach to enable cost-effective elasticity for Cloud applications. This approach operates a fine-grained scaling at the resource level (e.g., CPUs, memory, I/O) and a VM-level scaling. This policy enables monitoring service that monitors each running application in two levels. At first level, an entry monitor examines the requests arrival rate and response time over a finite interval, while the other level monitor examines the resources utilization. The policy applies an automatic reactive scaling mechanism and scaling actions are performed to meet the response time targets of each incoming request. An integrated and autonomic Cloud resource scaler has been developed by Hasan et al. [17]. This approach extends the typical two thresholds and add two levels of thresholds parameters in making scaling decisions. The first level, is set a parameter which is slightly below the higher threshold and the other level is set a parameter which is slightly above the lower threshold. The scaling actions are performed according to the CPU load and the response time over network. Chieu et al. [18] implemented a dynamic scaling algorithm for automated provisioning of Virtual Machines based on thresholds. A threshold on the number of active sessions is set in each Web server as scaling indicator. The scaling algorithm is implemented in a monitoring service and its scaling actions are performed based on the statistics of the scaling indicator. Al-Haidari et al. [19] analyze the impact of the utilization thresholds and the scaling size factor (i.e, number of instances to be added in each scaling action) on the performance of the Cloud services during the provisioning process. These two scaling factors are considered to cope with workload spikes and to improve the response

time.

2.1.2 Reinforcement learning

Reinforcement learning is an automatic decision-making approach defined as an interaction process between a learning agent (the autoscaling controller) and its environment (the target approaches). This technique gained popularity because of its ability to fast learn optimal elasticity policies at runtime without the need of prior knowledge of the application performance model. Q-Learning is a classic algorithm used by fuzzy autoscaling controllers for dynamic resource allocation [20]. A fuzzy reinforcement learning controller is proposed in [21]. This controller automatically scales up or down resources to meet performance requirements. In particular, workload and response time are monitored and the resource allocation is adapted to maintain the desired SLA, cost and response time. Asgari et al. [22] propose an automatic resource provisioning approach based on reinforcement learning for autoscaling resources according to a Markov Decision Process (MAP). The scaling decision is taken according to the four categories offered by MDP (i.e., conditions, operations, transmitted possibilities and rewards). Three factors are evaluated including SLA violation rate, scaling cost and number of scales. In [23] the authors propose two reinforcement learning techniques. In first one, the standard policies are evaluated by proper initialization of the learning functions. In second one, convergence speedups (i.e., increasing the rate of convergence in process of learning to solve problem) are applied for model-based reinforcement learning. In addition, a complete policy evaluation is introduced to detect changes at regular intervals into the learning phases. In [24] Barret et al. apply the Q-learning algorithm to determine an optimal autoscaling policy on a given platform. Their approach is based on agents learning in parallel on the same autoscaling task and sharing information regarding their experiences. This approach takes advantage of the inherent parallelism associated with distributed computational platforms. In addition, a state action space formalism is devised for learning optimal policies in the Cloud.

2.1.3 Control theory

Control theory deals with influencing the behavior of dynamic systems. The aim is to define the type of controller, i.e., reactive or proactive, that automatically adapts resources in response to the application demands. The controller replies according to the monitoring output and then sends the feedback to be compared with the refer-

ence values, e.g., associated with the SLA. The feedback of the input is the difference between actual and desired output level. Usually, a controller aligns the actual output to the reference. A control-theoretic elasticity management approach based on fuzzy control that enables qualitative specifications of elasticity rules is proposed in [25]. Response time or CPU load are the parameters used for evaluating the performance. Zhu and Agrawal [26] propose a multi-input-multi-output feedback control model-based dynamic framework for resource provisioning. This framework adopts reinforcement learning to tune parameters to guarantee the optimal application benefit within the time constraint. Padala et al. [27] introduce a resource control system that automatically reacts to the changes of a shared infrastructure to maintain application SLOs. This system is a combination of an online model estimator and a multi-input multi-output resource controller. The estimator collects performance metrics, while a controller allocates/deallocates the resources when needed. Kalyvianaki et al. [28] present an advanced controller that dynamically reacts to the workload changes without any prior knowledge of the application. This controller integrates a Kalman filter into feedback controllers that track the CPU utilization and dynamically update the allocations in order to meet the desired QoS. In [29] the authors apply techniques to augment the conventional closed-loop control framework by making automatic control robust for practically use in real environments. A self-trained proactive elasticity technique for Cloud-based services that automatically adjusts itself in response to the workload is presented in [30].

2.1.4 Time series analysis

Time series analysis is an approach applied for modeling and predicting the future behavior of a given phenomenon based on the historical data. Methods such as Moving Average, Auto Regression and Auto Regressive Integrated Moving Average are used to detect patterns and predict unforeseen values. Time series analysis is a key enabler of proactive autoscaling techniques. Typically time series analysis is utilized for workload or resource usage prediction. In [31] the authors applied a Fast Fourier transform in order to perform offline extraction of cyclic workload patterns. Different frameworks (e.g., CloudScale [32] and PRESS [33]) are used to perform long term cyclic patterns extraction and resource demands prediction.

Hu et al. [34] present a framework for predicting the incoming workloads and proactively provisioning the required resources by adding number of Virtual Machines to

reduce the latency and improve the QoS. The prediction is based on the historical workload and includes three modules (i.e., monitor, filter, predictor). A proactive autoscaling approach combined with different models (i.e., predictive model, cost model) is proposed in [35]. A workload prediction model based on time series and machine learning techniques is developed for predicting unforeseen workload patterns. In [36] Khan et al. propose a model that allows proactive analysis of workload patterns and estimation of the autoscaling operations. Three main entities of the autoscaling operations, such as, health monitor, a user-defined launch configuration that captures the parameters necessary to create and terminate resources and a load balancing component (i.e., load balancer, queue) are identified. The interactions between these entities are quantified as variables and coordinated by an autoscaling component. The model enables the analysis of the relations between the autoscaling operations and workload patterns. In [37] three models predict the workload based on the analysis of monitored data using time series analysis. In addition, new trigger strategy for automatic scaling mechanism is proposed to reduce the delay. The trigger strategy is based on a pattern matching model (i.e., a method matching the sequence with some historic patterns, and based on the string matching algorithm and Euclidean distance).

2.2 Cloud simulation tools

Simulation tools are widely used to simulate the behavior of Cloud environments and evaluate their performance. In addition, these tools open up the possibility of evaluating different scenarios in a controlled environment. The simulation experiments are repeatable. Different workload characteristics can be considered for testing resource provisioning policies.

Many simulation tools have been developed for Cloud environments, such as CloudSim, iCanCloud, GreenCloud, MDCSim, EMUSIM and CloudAnalyst. CloudSim [38] is one of the most popular simulation toolkits is one of the most popular simulation toolkits. It is a generalized, extensible simulation framework, developed in Java that enables contributions from other developers. In addition, it provides great flexibility to create simulation scenarios. In this thesis work, CloudSim toolkit has been extended (see Chapter 5) to create new features and functions to cope with our simulation scenarios. In Chapter 4, more details and explanations of the CloudSim will be provided.

In what follows, a brief review of the other Cloud simulation tools is presented.

The iCanCloud simulator [39] has been designed and built to conduct large experiments and provide a flexible and fully customizable global hypervisor for integrating any Cloud brokering policy. In addition, it contains a user-friendly GUI for configuring and lunching simulations and provides in-depth simulation of physical layer entities such as cache, allocation policies for memory and file system models.

GreenCloud [40] is an extension of the NS2[1] network simulator which focuses on simulating the communications between processes running in the Cloud at the packet level. Both NS2 and GreenCloud are written in C++ and OTcl[2]. GreenCloud provides plugins that allow the use of physical layer traces which make experiments more realistic. GreenCloud extracts, aggregates, and obtains information about the consumed energy in the data centers. For instance, a packet loss probability in the optical fiber depending on the transmission range can be obtained via simulation of signal propagation dynamics. GreenCloud is specially designed to simulate and test power consumption of the datacenter components (e.g., server, switches, links).

MDCSim [41] is a commercial event driven simulator for the design and analysis of large scale, multi-tier data centers. It manipulates different features of the data center (i.e, number of tiers, scheduling algorithms, communication mechanisms), does not support

[1] https://www.isi.edu/nsnam/ns/

[2] http://nile.wpi.edu/NS/otcl.html

GUI. In addition, it supports estimation of power consumption.

EMUSIM [42] is an integrated architecture to anticipate and predict service behavior on Cloud platforms. It is based on both simulation and emulation. The EMUSIM is built on top of architecture and operation details of Automated Emulation Framework[3] and CloudSim toolkit. This architecture automatically extracts information from application behavior via emulation and then uses this information to generate the corresponding simulation model. This model is then used to build a simulated scenario.

CloudAnalyst [43] is a discrete event simulator built on top of CloudSim toolkit. This tool allows users to model various scenarios where data centers and users are in different geographic locations. In addition, it supports visual modeling and simulation of large scale applications and allows description of application workloads, based on information of geographic location of users generating traffic, location of data centers, number of users and data centers, and number of resources in each data center. This information enables CloudAnalyst to generate information about response time and processing time of requests. The simulation results consist of different metrics, such as response time of the requests, processing cost. CloudAnalyst provides some features that can be summarized as follows:

- easy to use GUI;
- flexibility in customization and configuration of the simulation experiments;
- repeatability of simulation experiments;
- graphical output.

[3] http://emuframework.sourceforge.net/

2.2.1 Comparison of Cloud simulation tools

A limited number of Cloud simulation tools is available for public. Table 2.1 presents a summary and a comparison of the various Cloud simulators. The table lists the main characteristics of the simulators, such as programming language, distribution.

Simulator	Platform	Programming language	Networking	Simulator type	Distribution
CloudSim	CloudSim	Java	Limited	Event driven	Open Source
iCanCloud	OMNeT/MPI	C++	FULL	Event driven	Open Source
GreenCloud	NS2	C++/OTcL	FULL	Packet Level	Open Source
EMUSIM	AEF	Java	Limited	Event driven	Open Source
CloudAnalyst	CloudSim	Java	Limited	Event driven	Open Source
MDCSim	AEF/CloudSim	C++/Java	Limited	Event driven	Commercial

Table 2.1: Comparison of different Cloud simulators

As can be seen, all the tools are open source but MDCSim is commercial. As shown, GreenCloud is a packet level simulator. This means that whenever a data message has to be transmitted between the simulator entities, a packet structure with its protocol headers is allocated in the memory and is processed by the associated protocol. CloudSim, iCanCloud, EMUSIM, CloudAnalyst and MDCSim are event-based simulators. They are not processing small simulation objects, such as packets individually, but they make interactions between objects. This method reduces simulation time considerably and improves scalability. As can be seen, all tools support either full or limited networking. GreenCloud and iCanCloud offer full support to communication model by completely implementing TCP/IP protocol reference model. This allows capturing the dynamics of widely used communication protocols such as IP, TCP and UDP. CloudSim, EMUSIM, CloudAnalyst and MDCSim implement limited communication support. For example, in CloudSim, the transmission delay and bandwidth are accounted. A network package is provided with CloudSim, that maintains a data center topology in the form of a directed graph, where the bandwidth and delay parameters are assigned to edges. Similarly, simplified communication support is implemented in MDCSim.

Chapter 3

WORKLOAD CHARACTERIZATION

Workload characterization plays a key role in many performance studies. The term workload refers to all inputs received and processed by a given technological infrastructure. Understanding the properties and the behavior of the workloads is important for different performance issues. In particular, Cloud services being deployed nowadays are characterized by dynamic changes in load intensity.
Workload characterization is the basis for devising and evaluating efficient resource provisioning policies. The evaluation of the Quality of Service (QoS) perceived by users also requires a good understanding of the workload properties.
This chapter discusses the basic concepts of workload in the Clouds, the workload characterization and the specific issues related to generation of Web workloads. In addition, the technologies applied to develop a workload model will be also explained.

3.1 Workloads in the Clouds

Cloud workloads are composed by a collection of various applications and services characterized by its own performance and resource requirements and constraints usually specified in the form of Service Level Agreements [44]. The complete lifecycle of the workloads in the Clouds includes different aspects that refer to the characterization at the design time (i.e., workload categories, structures and patterns), the matching at the deployment time (i.e., resource requirements and scheduling) and the conditions of the execution time (i.e., failure analysis and prediction). The workload dynamics affect Cloud performance. In fact, workload can suddenly grow or shrink as a consequence of the user interactions. In particular, the use of virtualization and shared resources could lead to performance degradation. The degradation is often due to interference and resource contention arising from the co-location of heterogeneous workloads on the same physical infrastructure and overheads caused by the resource management policies being adopted.

3.1.1 Workload categories

The term workload refers to all inputs of a system, e.g, applications, services, transactions, data transfers. These inputs are submitted by the users and processed by a Cloud infrastructure. These inputs usually correspond to the online interactions between users and Web application and to jobs to be processed in batch mode.
The behavioral characteristics of the Cloud workloads can be specified in terms of qual-

itative and quantitative attributes. These attributes are analyzed to identify workload categories that focus on aspects, such as:

- Processing model;
- Architectural structure;
- Resource requirements;
- Non-functional requirements.

These dimensions play an important role in the formulation of the Cloud management strategies and in assessment of the expected service level.
The *processing model* addresses two types of workload categories, namely online (i.e., interactive) and offline (i.e., batch or background). These workload categories are characterized by different behaviors and performance requirements. Moreover, they have a different impact on management policies (e.g., resource scheduling, VM placement, VM migration). An *interactive workload* is typically composed of short lived processing tasks submitted by a variable number of concurrent users. On the contrary, a *batch workload* is composed of compute intensive long lived tasks.
The *architectural structure* is another dimension that takes account of the processing and data flows characterizing each individual Cloud application. In particular, these flows are described by the number and types of services or tasks being instantiated by a Cloud application and their mutual dependencies.
The workloads also are classified according to the amount of resources used, namely:

- Compute or I/O intensive;
- Elastic or bandwidth sensitive.

In general, network bandwidth is a critical resource for online interactive workloads, while batch workloads are often characterized by intensive storage and computing requirements. In addition, the resource requirements of some workloads are stable, that is, evenly distributed across their execution, whereas other workloads, such as those associated with the online services, exhibit specific temporal patterns, e.g., periodic, bursting, growing, on/off.
The dimension describing the *non-functional requirements* refers to SLA constraints, such as performance, dependability and security. In particular, reliability is very important in Cloud environments especially when deploying business-critical or safety-critical

applications. Reliability denotes the probability that workloads can successfully complete in a given time frame. The presence of failures decreases the reliability. Failures are due to various types of events, e.g., software bugs, exceptions, overflows and timeouts. In detail, for data intensive workloads, a sudden increase in the rate at which data are submitted for processing can lead to failures, thus making the service unavailable. Moreover, failures are often correlated, that is, they often occur between dependent or co-located services or applications.

3.1.2 Workload monitoring and profiling

Monitoring and profiling are the basis for measuring the qualitative and quantitative attributes of the workloads. In general, monitoring keeps track of all the activities performed in the Cloud by the workloads being processed and of the status of the resources either allocated and available. Profiling focuses on describing how workload exploits the Cloud resources. These activities play a critical role when addressing different problems, such as capacity planning and resource management, performance tuning, billing, security and troubleshooting, SLA verification. To tackle specific monitoring issues, various approaches have been devised (e.g., measurement sources and accuracy, sampling granularity, intrusiveness and scalability).
Workload attributes can be monitored at runtime to describe the resource usage. Two perspectives can be adopted to collect measurements, namely, *Cloud provider* and *Cloud user* perspective. The main targets of Cloud monitoring are:

- Client;
- Virtual Machine;
- Physical machine.

In details, individual VMs and resource usage of physical machines can be measured by the hypervisor. This service is performed by the Cloud provider. On the contrary, Cloud users have limited privileges for monitoring their workloads. They could use logging and profiling facilities. Because of the VM isolation – a virtualization technology that hides the characteristics and performance of the underlying physical machines and the VM management policies – Cloud users use profiling facilities made available by the providers (see, e.g., [45, 46]).

The application logs are exploited to correlate the resource usage with workload intensity and characteristics.

Monitoring tools collect measurements about resource usage by deploying distributed software agents. Generally, monitoring approaches rely on system tools and interfaces (e.g., `vmstat`, `iostat`, `netstat`) or on proprietary solutions. Alhamazani et al. [47] have studied various monitoring techniques that monitor different application components (e.g., Web server, compute service, storage service and network). In addition, they consider various QoS parameters, including CPU utilization, bandwidth, throughput and response time to be monitored. The QoS statistics can assist the Cloud providers to maintain the SLA and customer satisfaction. Moreover, depending on the monitoring capabilities of the virtualization technologies, ad-hoc scripts can be used for sampling low level quantitative attributes, such as CPU waiting times, number of virtual memory swaps, TLB flushes and interrupts [48]. VM scheduling and provisioning events, (e.g., number and types of allocated VMs) can also be collected by monitoring agents [49].

The granularity and level of details of the measurements have to be chosen with the aim of limiting the monitoring intrusiveness. Measurements are usually stored into *tracelogs*, that is, collections of time stamped records with various types of information (e.g., resource demands, scheduling events, application specific data).

Resource usage of individual workload activities can be measured by *Profiling*. In detail, profiling can be exploited by Cloud users for optimal dynamic resource provisioning and by Cloud providers for tuning VMs placement and scheduling policies [50]. Profiling has to cope with new challenges due to interference among co-located VMs. Indeed, the sharing of hardware resources could result in unpredictable behaviors of hardware components, such as cache, CPU pipelines and physical I/O devices [51]. Typical solutions for collecting profiling measurements are based on dynamic instrumentation and sampling hardware performance counters. An alternative approach is based on measuring at the *hypervisor level* the overall behavior of the VMs hosting the target applications.

Monitoring and profiling are essential aspects of Cloud computing. However, no portable and interoperable tools are available. There are many open source and commercial tools addressing specific targets and platforms [52, 53]. Examples of *open source monitoring tools* are: `Nagios`[1], that is part of the OpenStack suite[2], `Ganglia`[3], `Collectl`[4] and

[1] http://nagios.sourceforge.net

[2] http://www.openstack.org

[3] http://ganglia.sourceforge.net

[4] http://collectl.sourgefourge.net

`MonALISA`[5]. Cloud providers offer several commercial tools such as, `Amazon CloudWatch`, `Microsoft Azure Watch`, `IBM Tivoli Monitoring`, `Rackspace`, `RightScale`, `Cloudify`, `Aneka`. While these monitoring facilities are designed to be deployed in Cloud environments, external monitoring services like `CloudHarmony`[6], `CloudSleuth`[7], `CloudClimate`[8] and `Up.time`[9], focus on monitoring applications and infrastructures from multiple locations on the Internet.

The development of a common framework of workload monitoring is an open issue, that might be seen as an obstacle for users to deploy their applications [54]. Some recent studies have introduced the concept of *Monitoring as a Service* (MaaS) [55, 56] to improve the scalability and effectiveness of monitoring service consolidation and isolation.

3.1.3 Workload scheduling

Workload scheduling is a challenging issue in Cloud environments. In particular, the mapping between jobs/tasks and VMs is an open research aspect. The problem of finding an *optimal mapping* is *NP-complete* and therefore it is hard to deal with exact methods when the number of VMs and tasks is large. For this reason, (meta-)heuristics are currently used to find sub-optimal solutions. Meta-heuristics based on methods, such as neural networks, evolutionary algorithms or set-of-rules, are proved to be efficient in solving optimization problems related to scheduling.

The scheduling problem addresses different objectives, such as to minimize makespan, data transfer, energy consumption and economic cost to satisfy SLAs. Simple approaches take into consideration one objective at a time. More sophisticated approaches are aimed at combining multiple objectives into a single aggregate objective function (see, e.g., [57]) or considering multi-objective algorithms (see, e.g., [58, 59]).

A recent survey summarizes the evolutionary approaches for scheduling in Cloud environments [60]. The different viewpoints for scheduling and the corresponding objectives are identified as follows:

- Scheduling for user QoS whose main objectives are the makespan and cost minimization, as well as application performance and reliability;

[5] http://monalisa.caltech.edu
[6] http://cloudharmony.com
[7] http://cloudsleuth.net
[8] http://www.cloudclimate.com
[9] http://www.suptimesoftware.com

- Scheduling for provider efficiency whose main objectives are load balancing, maximization of the resource usage and energy savings;

- Scheduling for negotiation whose main objectives are to satisfy both user and provider goals.

Job scheduling and *resource scaling* are often interrelated [61]. Several frameworks have been recently introduced to address resource scalability. For example, SmartScale [62] is an automated scaling framework that uses a combination of vertical and horizontal approaches to optimize both resource usage and reconfiguration overheads. Scaling mechanisms are also encountered in [63]. In this work, different scalability patterns are considered and a performance monitoring approach that allows automatic scalability is proposed. Autoscaling is often interrelated with *load balancing* strategies. Even though physical machines are often the main target of these strategies, effective load balancing and resource allocation policies take into account the concurrent execution of different application types, i.e., interactive, batch, and the mix of applications with different resource requirements [64, 65, 66].

3.2 Web workloads

Web servers are the basic components of the Web. They host Websites and allow users to access Web pages by means of HTTP protocol. In this thesis work, the term Web server indicates the software components, whose primary function is to process the HTTP requests received from the clients (i.e., users) and generate HTTP responses to deliver Web pages to the clients. One important feature of the Web servers is the ability to collect detailed information about their traffic, i.e. HTTP requests and responses. This information is stored to log files.

For example, the Apache Web server[10] provides very comprehensive and flexible logging capabilities. The two main types of log files are the access log [11] and error log [12]. Both types of files contain information about the HTTP requests received by the Web server and the corresponding HTTP responses. In particular, the error log stores information about the requests that caused errors, such as requests for non existing files. This log is very useful for troubleshooting. The types of information and the format of the access

[10] `https://httpd.apache.org/`

[11] `https://httpd.apache.org/docs/1.3/logs.html#accesslog`

[12] `https://httpd.apache.org/docs/1.3/logs.html#errorlog`

log is configurable by means of the CustomLog directive [13] available in the Apache configuration file.

The information stored in the logs is structured in records, one record per HTTP transaction. Two standard formats are available. The Common Log Format [14] includes the the IP address of the client that issued the HTTP request, the date-time of the request, the request line, the status code of the response message and the size of the requested resource (see Figure 3.1).

On the contrary, the Combined Log Format [15] contains few additional fields, namely, the User-agent and Referrer fields. The User-agent refers to the software agent used by the client to issue the request, while the Referrer denotes the page from which the request comes from. Figure 3.2 shows a record stored according to the Combined Log Format.

```
119.153.159.149 - - [03/Apr/2016:06:32:26 +0200] "GET /site/en/
   home.html HTTP/1.1" 200 14781
```

Figure 3.1: Example of a record of access log stored according to the Common Log Format.

```
119.153.159.149 - - [03/Apr/2016:06:32:26 +0200] "GET /site/en/
   home.html HTTP/1.1" 200 14781 "https://www.google.com.pk/" "
   Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like
   Gecko) Chrome/49.0.2623.110 Safari/537.36"
```

Figure 3.2: Example of a record stored according to the Combined Log Format

The Apache Web server supports an additional log format, namely, the Forensic Log that stores two records, one for the HTTP request message and one for the HTTP response message, thus allowing a deeper level of detail. This format is not commonly

[13] https://httpd.apache.org/docs/1.3/mod/mod_log_config.html#customlog

[14] https://httpd.apache.org/docs/1.3/logs.html#common

[15] https://httpd.apache.org/docs/1.3/logs.html#combined

used because it can cause overload on the server and the size of the corresponding log file tends to increase very quickly.

3.3 Workload characterization methodology

Workload models are abstractions of the real workloads that capture their main characteristics and reproduce the users behaviors. Various interesting works that model the user behavior and derive representative workload models have been proposed in the literature (see, e.g., [67, 68, 69, 70, 71]).
A Web workload consists of the HTTP requests issued by the clients toward Web or proxy servers to download a page. This workload has been extensively studied and characterized in [72, 73]. The main aspects considered refer to the page and the traffic properties, the access patterns, and the user behavior.
In this work, we focus on workload characterization based on experimental approaches, that is on the analysis of measurements collected on real infrastructures while the real workload is being processed.

3.3.1 Selection of characterizing parameters

The definition of the basic workload components is the initial steps of any workload characterization study. For example, the HTTP requests or a user-session can be selected as workload component. A user-session consists of a set of requests, where the time between consecutive requests, i.e., interarrival time, is smaller than a predefined value. The workload is described by the intensity, e.g., request arrival rate. Additional parameters are derived from log files or from performance monitors. A Web page consists of an HTML file (i.e., primary page) and several embedded objects. Each request can hence be described by its type (i.e., primary Web page or embedded object), size, and interarrival time.
Once measurements have been collected, and the parameters describing the workload have been selected, the properties of the workload and the behavior of the users need to be uncovered as to build the corresponding models. In what follows, the main analysis techniques are explained.

3.3.2 Exploratory data analysis

Data analysis, that is, the process of preparing, transforming, evaluating and modeling data to discover useful information, includes several steps. These steps range from converting the data to a format appropriate for analysis (e.g., cleaning the dataset), applying statistical and visualization techniques and interpreting the results. Statistical methods are applied to discover the characteristics of the data. In particular, descriptive statistics provide a quantitative description of the data. For example, measures of dispersion (e.g., mean, range, variance, coefficient of variation, skewness, median and percentiles) are important to describe the properties of each workload parameters. Moreover, parametric statistics allow for exploring the strength of the relations between the parameters. Pearson correlation coefficient ρ_{xy} provides a quantitative measure of the extent to which parameter x is positively or negatively related to parameter y, that is:

$$\rho_{xy} = \frac{\sigma_{xy}}{\sigma_x \sigma_y},$$

where σ_{xy} denotes the covariance and σ_x and σ_y the standard deviations of the parameters x and y, respectively.

Visualization techniques are applied to summarize the distributions of the parameters and identify potential outliers. The term *outlier* refers to workload components characterized by an "atypical" value of one or more parameters. Outliers help in identifying anomalies, unexpected behaviors or even errors in the measurements. Outliers removal is crucial because of their potential effects on the workload models.

3.3.3 Multivariate Analysis Techniques

A further step of the workload characterization methodology deals with the analysis of the components in the multidimensional space of their parameters. This step allows for deriving models that capture and summarize the overall properties of the workload. For this purpose, clustering techniques are applied. Clustering is an unsupervised process that subdivides a set of observations (workload components) into homogeneous groups (i.e., clusters) [74]. Clustering groups components with similar properties, whereas the components across groups are quite distinct. The centroids of the clusters are often used as representatives of the groups.

A very popular nonhierarchical clustering algorithm is the *k-means* [75]. This algorithm

iteratively partitions the data into k clusters by assigning each observation to the cluster C_j that minimizes the objective function, that is:

$$\sum_{j=1}^{k} \sum_{i \in C_j} \|x_i - c_j\|^2$$

where x_i and c_j denote the i^{th} observation and of the centroid of the cluster j, described by the m parameters. $\|\cdot\|$ refers to the Euclidean distance. This algorithm minimizes the distances within a cluster, while the distances among clusters are maximized.
Since the number of observations and the number of their characterizing parameters are often quite large, to reduce the data dimensionality Principal Component Analysis (PCA) [76] is applied. This analysis linearly transforms the potentially correlated parameters into a set of uncorrelated parameters, that is, the principal components.
In this thesis work, workload characterization is applied to derive models of Web workloads to be used to drive the simulation experiments with a realistic workload.

Chapter 4

SIMULATION ENVIRONMENT

Cloud environments are very complex. Therefore, testing and evaluating autoscaling policies on real Cloud infrastructure requires large efforts. Moreover, it is very hard to perform repeatable and controllable experiment on a real infrastructure. Thanks to simulation, it is possible to overcome these issues and analyze different policies under different configurations before their actual implementation and deployment into a production environment. This chapter presents the CloudSim simulation toolkit and describes its main components and features.

4.1 CloudSim simulation toolkit

The CloudSim simulation toolkit is a Java event-based Cloud simulator, developed and maintained by the Cloud Computing and Distributed Systems Laboratory of the University of Melbourne [38]. It provides a generalized and extensible simulation framework that enables modeling, simulation and experimentation of the Cloud infrastructures and services. CloudSim offers many different functionalities:

- modeling and simulation of large scale Cloud computing environments;
- simulation of network connections among the simulated system components;
- virtualization engine that aids in creation and management of multiple, independent and co-hosted virtualized services;
- space-shared and time-shared allocation of processing cores to virtual services.

Several entities, such as Data Centers, Hosts and VMs, communicate with the core of CloudSim. These entities are also called components. Within a single Host, one or more Virtual Machines can be allocated. A Virtual Machine is characterized by a pre-assigned amount of resources, such as amount of memory, storage, bandwidth and processing power. The Host component represents a physical machine, that is, a server. Each Host has pre-assigned characteristics, such as memory, storage, number of cores. In addition, management of Virtual Machines (e.g., creation and destruction) is performed by the Host. The Data Center component is responsible to handle the incoming requests and schedule them on the Virtual Machines for processing. The simulation is driven by events occurring at ordered timestamps. These events have a specific structure, defined in the "SimEvent.Java", that consists of the following fields:

- **Event Type:** specifies the type of event;

- **Event Start Time:** defines the start time of the event;
- **Event Waiting Time:** defines the time spent in the waiting queue;
- **Event Source Entity:** defines the entity requesting the event for processing;
- **Event Destination Entity:** defines the entity that will actually execute/process the event;
- **Event Tag:** defines the type of behavior to be simulated. Tags have predefined event values associated with them;
- **Event associated data to be processed:** this field is optional and depends on the type of event tag associated with the event.

Different scheduling policies are available in the CloudSim toolkit. For exmaple, *VmAllocationPolicy* is an abstract class that represents the provisioning policy of Hosts to Virtual Machines in a Datacenter. This policy allocates Hosts for placing VMs. Once a VM is allocated to an Host with sufficient resources, there are two options available for scheduling VMs. The time-shared policy is such that the CPU cores can be shared across multiple VMs at the same time, whereas with the space-shared policy dedicated CPU cores are allocated to each individual VM. Figure 4.1 depicts the scheduling policy model (i.e., space-shared, time-shared). The policies can be seen at the Host and VM layers.

The Cloudlet scheduler is responsible for managing the actual workload on a VM. Several Cloudlets run simultaneously on a given VM. Time-shared and space-shared are implemented by the CloudletScheduler and allow processor sharing at the VM and Cloudlet levels. The processing resources of a VM are allocated by the scheduling policy implemented by the VmScheduler, which is the class representation of the scheduling policy for the VMs.

VmShedulerTimeShared and VmShedulerSpaceShared are implemented by VmScheduler. Once the VmAllocationPolicy has found a Host with the necessary resources, such as CPU cores, memory, network bandwidth and storage, the VM allocation is performed and the processing shared policy is allocated by the VmScheduler and CloudletScheduler. Figure 4.2 explains the mechanism of sharing the CPU cores by VMs and Cloudlets. At a certain time, VM 0 and VM 1 share the two cores and the Cloudlets are being processed at the same time on both cores.

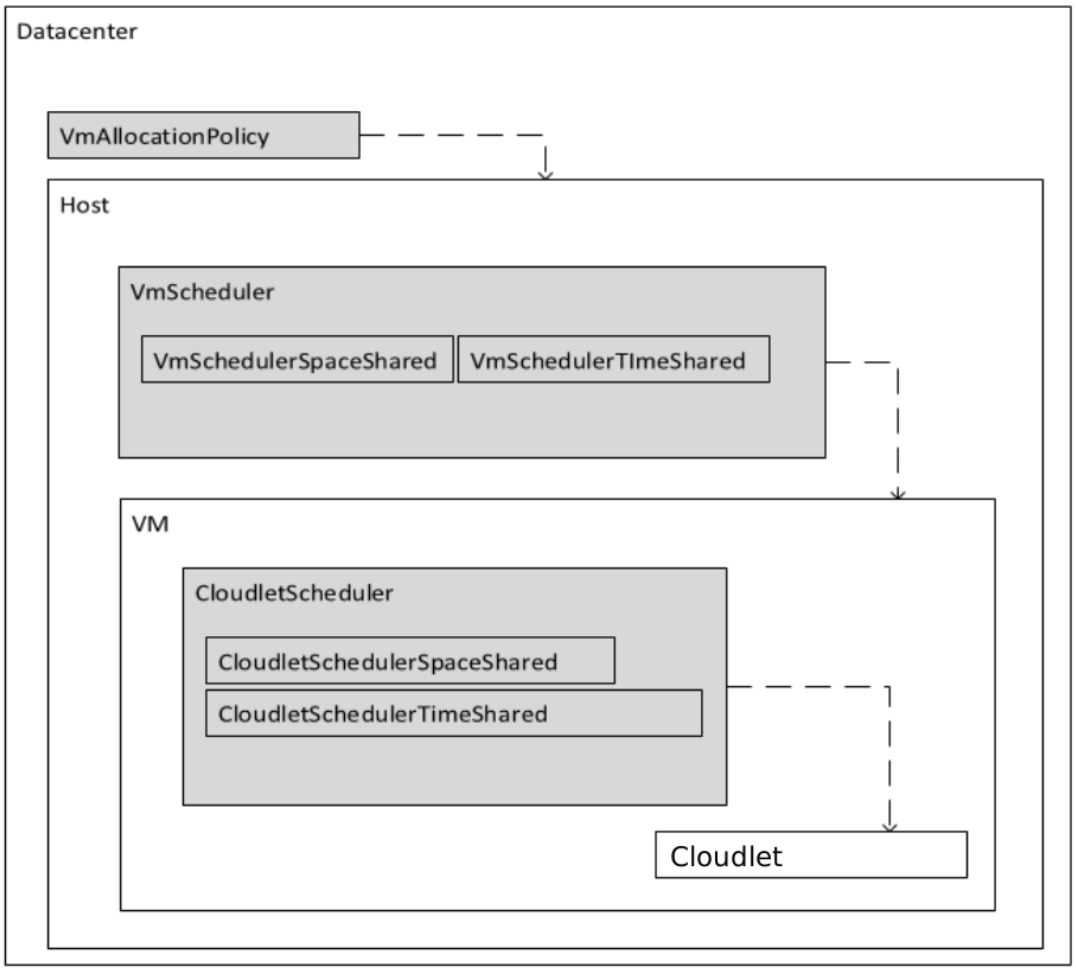


Figure 4.1: Scheduling policy model

Figure 4.3 depicts the space-shared policy for VMs and Cloudlets. This policy dedicates the two cores at certain time to one VM and each core processes only one Cloudlet at a time.

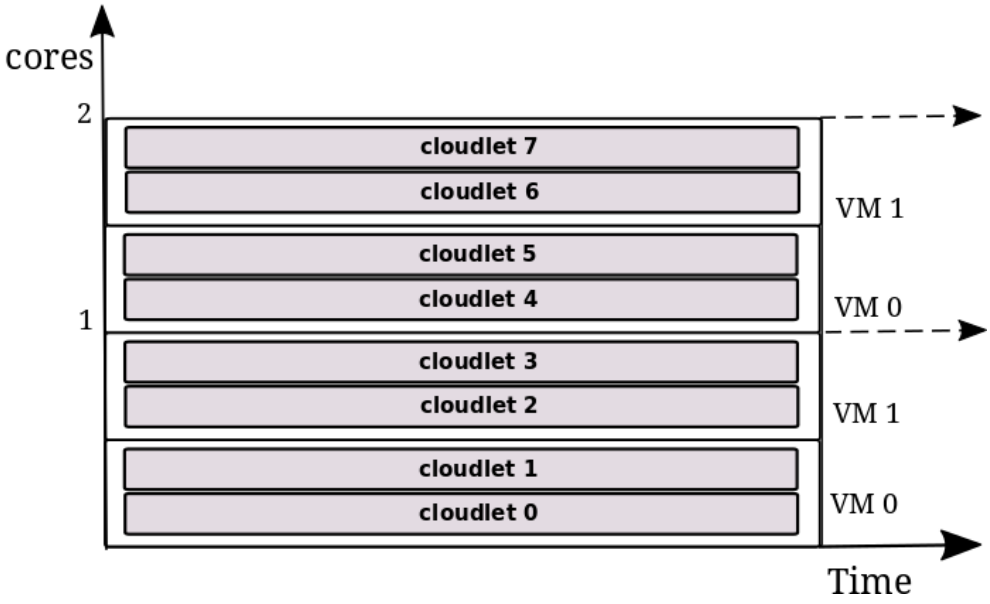


Figure 4.2: Time-shared policy for VMs and Cloudlets

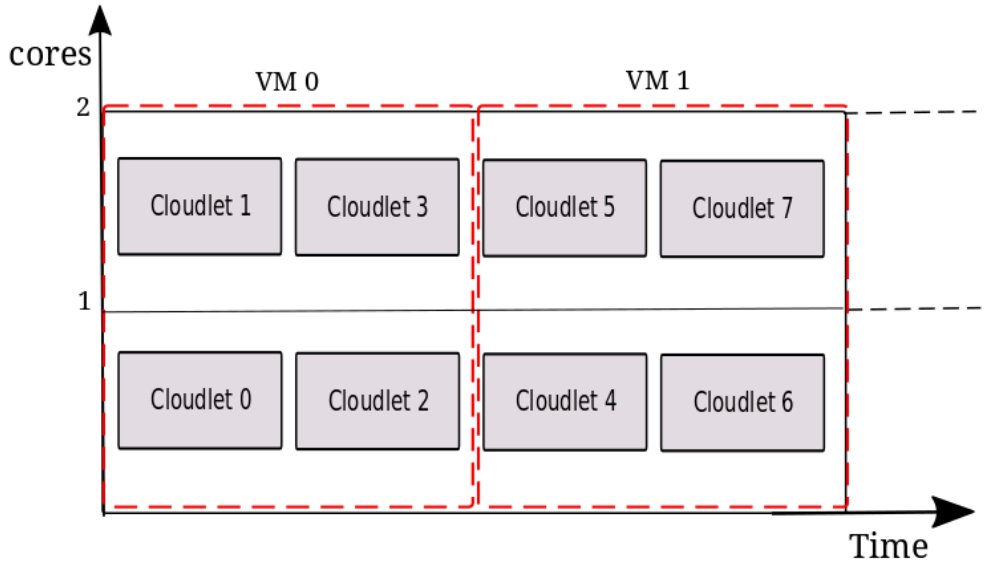


Figure 4.3: Space-shared policy for VMs and Cloudlets

Figure 4.4 shows the multi-layered architecture of the CloudSim simulation toolkit. As can be seen, the architecture consists of three layers, namely: User code, CloudSim and CloudSim core simulation engine. The top layer is the User code used to define

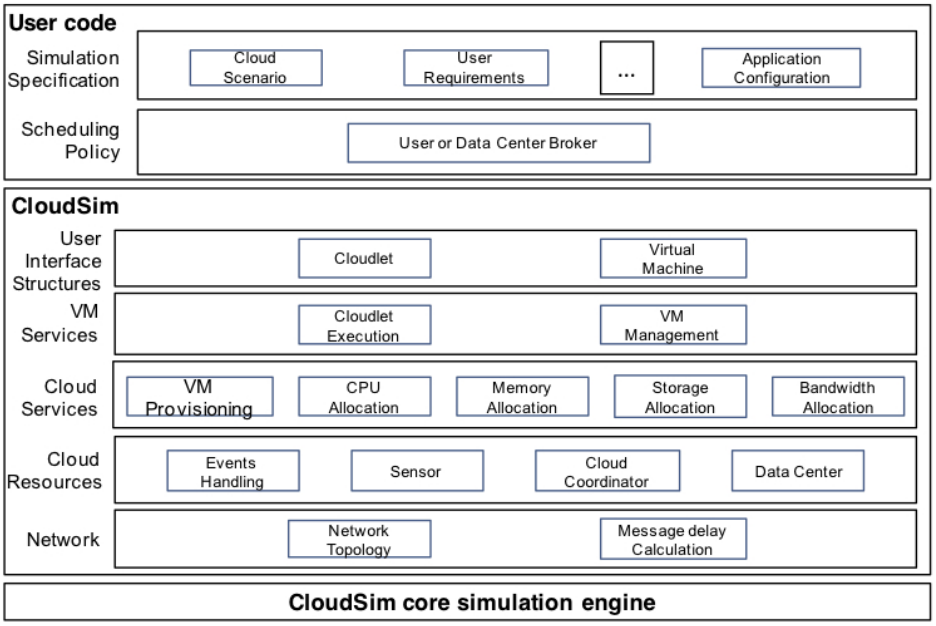


Figure 4.4: Layered architecture of the CloudSim simulation toolkit

the basic entities of the Hosts, applications and VMs. In particular, the user specifies the configurations of Virtual Machines to be allocated and their characteristics and the requirements and characteristics of the workload. In addition, the user specifies the policies for allocation or deallocation of the VMs. Scheduling the workload and managing of the resource are performed by the broker.

The CloudSim layer consists of various modules and components that enable building a Cloud simulation environment. The modules of this layer deals with resource provisioning and VM allocation to the Hosts and with managing the VMs and the execution of the application tasks (Cloudlets).

The lower layer corresponds to the core of the simulation engine that deals with the communication between components, the processing of the events and the management of the simulation clock. This layer implements a discrete event simulation framework to coordinate the various components.

The main extensions designed and developed within this thesis work refer to the core layer. In particular, the functionalities deal with:

- definition of workload configuration and generation of the workload requests;
- implementation of new scheduling and provisioning policies;
- modeling of different Cloud scenarios.

As a result, the extensions of the basic functionalities of the CloudSim toolkit make it possible to perform different tests based on specific scenarios and configurations, in terms of workload scheduling, resource provisioning, VM allocation and performance monitoring.

4.2 Basic CloudSim entities

This section briefly describes the main classes of the CloudSim toolkit and the characteristics used to model and simulate a Cloud infrastructure. Figure 4.5 shows the most important classes of the CloudSim toolkit. The *CloudSim* class includes the core of the

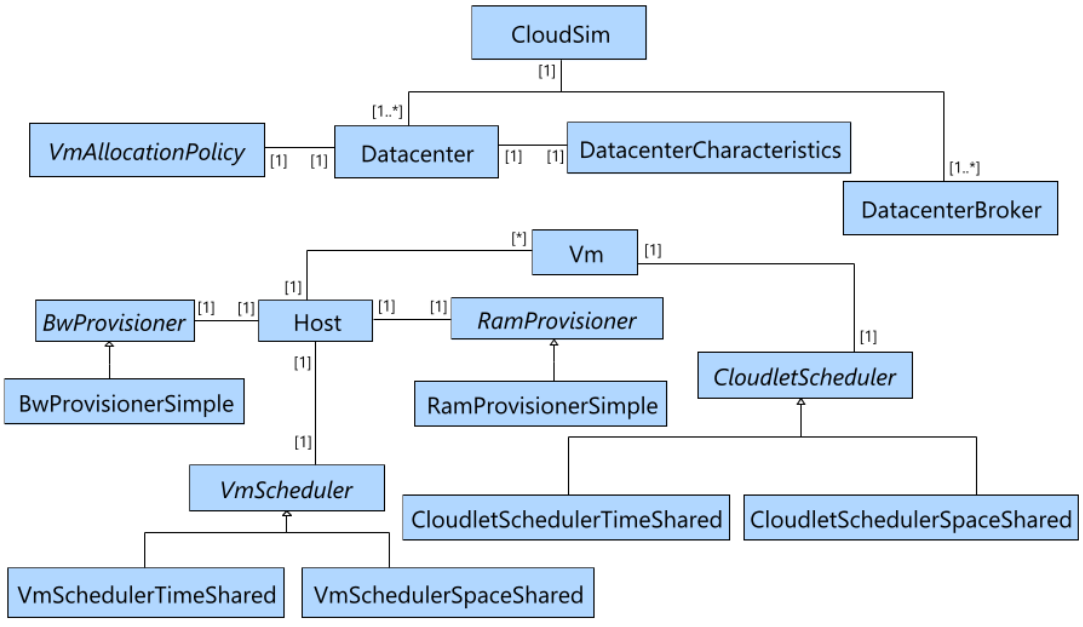


Figure 4.5: CloudSim UML class diagram

simulation since it manages the queue of the events, namely, future and deferred events. An event is characterized by a time and after its creation it is scheduled into the queue of future events where all events are stored according to their future occurrence time. Once an event has occurred, it is moved to the deferred queue where it is processed.

The *CloudSim* class is responsible for organizing and executing all the events of the simulation. The events are registered by the $CloudInformationService(CIS)$, a component instantiated in the *CloudSim* class that keeps track of all the Cloud resources and their IDs.

The main entities acting during the simulation, such as the Data Center and the Broker can schedule new events and send messages to other events. For this purpose, they must extend the *SimEntity* abstract class. The *Datacenter* class represents the Data Center, that is, the hardware infrastructure offered by Cloud providers. This class encapsulates a set of compute Hosts that can either be homogeneous or heterogeneous with respect to their hardware configurations (i.e., memory, number of cores, bandwidth and storage). Furthermore, every Datacenter component instantiates a generalized application provisioning component that implements a set of policies for allocating bandwidth, memory, and storage devices to Hosts and VMs.

The $Host$ class models a physical resource, such as a compute or storage device in the Data Center. This class encapsulates important information, such as the amount of memory and storage, the list and types of processing cores, an allocation policy for sharing the processing power among Virtual Machines, and policies for provisioning memory and bandwidth to the Virtual Machines. It is possible to change these policies, by extending the respective abstract classes, namely, $RamProvisioner$, $BandwidthProvisioner$ and $VmScheduler$.

The VM class models the Virtual Machines, managed and hosted by a Cloud $Host$ component. To create a Virtual Machine it is necessary to specify its characteristics, that is, the number of instructions it can process per second (expressed in MIPS), its image size (expressed in Mbytes), its RAM (expressed in Mbytes), the bandwidth it can use (expressed in Mbps), the Virtual Machine Manager, the broker responsible for assigning the Cloudlet, and provisioning policy. In addition, it is necessary to specify the policy for the Cloudlet scheduling. New policies for scheduling the Cloudlets can be created by extending the *CloudletScheduler* abstract class.

The $ProcessingElement$ (PE) class represents a CPU core of a physical machine. The CPU unit is defined in terms of Millions Instructions Per Second (MIPS) rating. The Data Center acts on behalf of the user. The Cloudlet entity is the smallest component of an application service. It is executed as a task on a given VM. The Cloudlets correspond to the workload to be processed by the VMs.

The *Cloudlet* class models the application services to be deployed. The characteris-

tics of an application can be defined in term of its resource requirements, bandwidth, memory and I/O operations. A Cloudlet can be seen as a process to be executed on a Virtual Machine. The Cloudlet class includes the specifications of the cores that can be used to execute the Cloudlet and the utilization models of CPU, RAM and bandwidth. For creating new utilization models, it is necessary to implement the interface $UtilizationModel$. These models represent a key aspect to consider for modifying the behavior of the workload. When a Cloudlet is submitted to a Virtual Machine, its execution time is estimated to schedule the event corresponding to the completion of its processing. Since the VM capacity changes in response to the workload dynamics, it is necessary to re-compute the expected completion time and update the remaining number of instructions to be processed.

The class $DatacenterBroker$ models a broker responsible for mediating negotiations between Cloud application requirments and Cloud providers. The broker acts on behalf of Cloud providers. It discovers suitable Cloud service providers by querying the Cloud Information Service (CIS) and undertakes on-line negotiations for allocation of resources/services that can meet the application QoS requirements. Within the simulation, the broker is responsible for VM creation and deletion and to keep track of the list of the Cloudlets submitted and executed on the VMs.

The abstract class $SimEntity$ is able to handle events and send events to other entities. Subclasses of the SimEntity are: Datacenter, DatacenterBroker, CloudInformationService, CloudSimShutdown. Since the CloudSim is an event-driven simulator, it is important to exchange these events between the simulation entities. Therefore, these entities must extend the SimEntity class and override the contained methods to create, process, and delete entities.

The $BwProvisioner$ is an abstract class that models the policy of bandwidth provisioning to VMs. The main role of this component is to allocate the network bandwidth to a set of competing VMs deployed on the data center. The class allows VMs to reserve as much bandwidth as required, even though this is constrained by the total available Host bandwidth.

The $CloudletScheduler$ abstract class implements the policies that determine the share of processing power of the Cloudlets in a Virtual Machine. Two types of provisioning policies are implemented: space-shared and time-shared.

The $DatacenterCharacteristics$ class contains configuration information of data center resources.

The *RAMProvisioner* abstract class represents the policy for allocating memory to the VMs. The execution and deployment of VMs on a given Host is feasible only after the RamProvisioner component has checked that the Host has the required amount of free memory.

The *VMMAllocationPolicy* abstract class represents the provisioning policy of Hosts to Virtual Machines in a data center. The main functionality of this class is to select an available Host in the data center that meets the memory, storage, and availability requirements for the VM deployment.

The *VMsScheduler* class is a component implemented by the Host component that models the policies (i.e., space-shared, time-shared) required for allocating processor cores to VMs.

Figure 4.6 depicts CloudSim high level modeling. All the resources of the Cloud environment have to register themselves in the Cloud Information Service entity that is responsible to respond to the queries for the available resources.

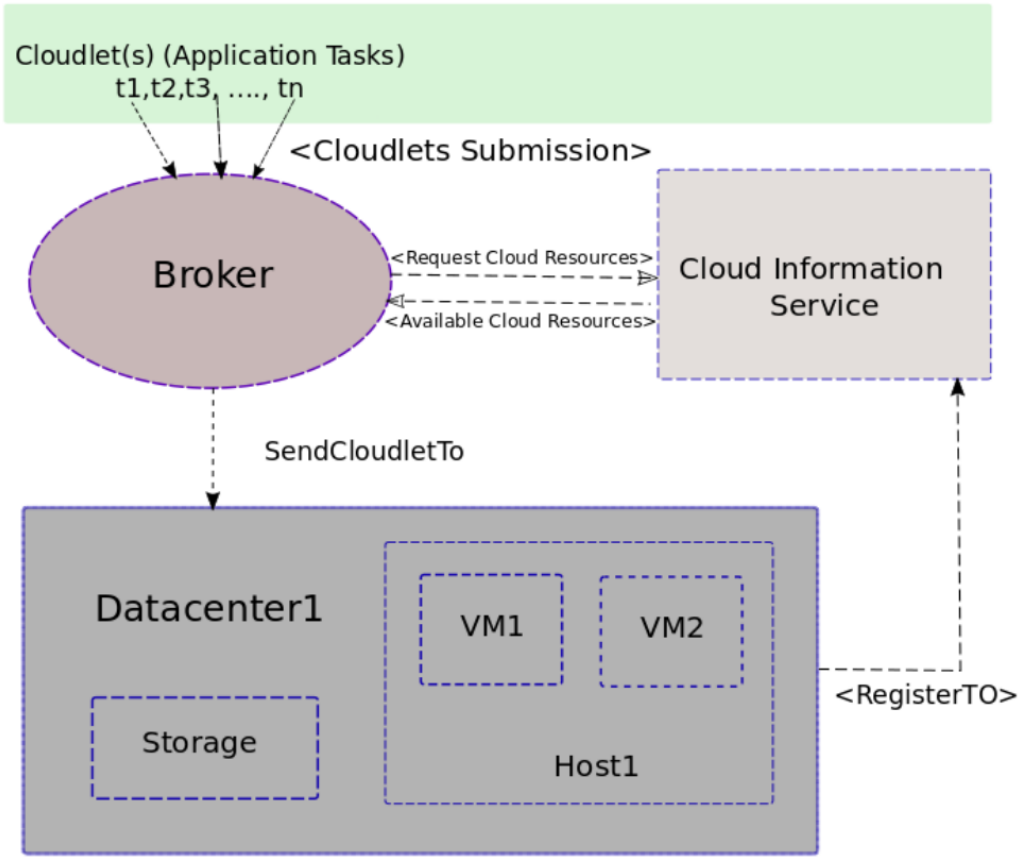


Figure 4.6: CloudSim high level modeling

The Cloudlets are received by the broker which checks the available resources by querying the CIS. Then, the broker schedules the Cloudlets on the available VMs and keeps track of their processing status.
A typical simulation experiment follows the life cycle shown in Figure 4.7.

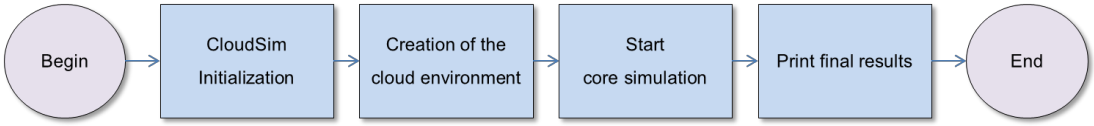


Figure 4.7: Simulation life cycle

Each simulation scenario is defined by specifying the appropriate configurations. When the simulator starts, all the simulation entities are created and instantiated according to the specified configurations, modeling a full Cloud simulation environment.

Chapter 5

CLOUDSIM TOOLKIT EXTENSIONS

To generate the workload dynamics according to the devised workload models and test the performance of autoscaling policies, a simulation environment has to cope with different Cloud scenarios. For this purpose, we develop a framework built on top of the CloudSim simulation toolkit. This chapter describes the extensions of the CloudSim toolkit designed and implemented in this thesis work.

5.1 Introduction

To setup a simulated Cloud environment, it is necessary to define the required components and integrate them into the simulator. These components – that interact at simulation time with the core of the CloudSim toolkit – are used to generate a Web workload according to the devised workload models. The generated workloads are scheduled and processed on the allocated VMs according to the scheduling policies (i.e., Round Robin, Weighted Round Robin) specified at the broker. In addition, measurements are collected on the computation process by the monitoring component. The autoscaling component implements two policies (i.e., discrete sampling, average load) that consider the load of the VMs and dynamically scale the computation resources. Figure 5.1 summarizes the architecture of the simulation environment designed and developed within this thesis work and outlines the interactions among the components and with the core of the CloudSim toolkit.

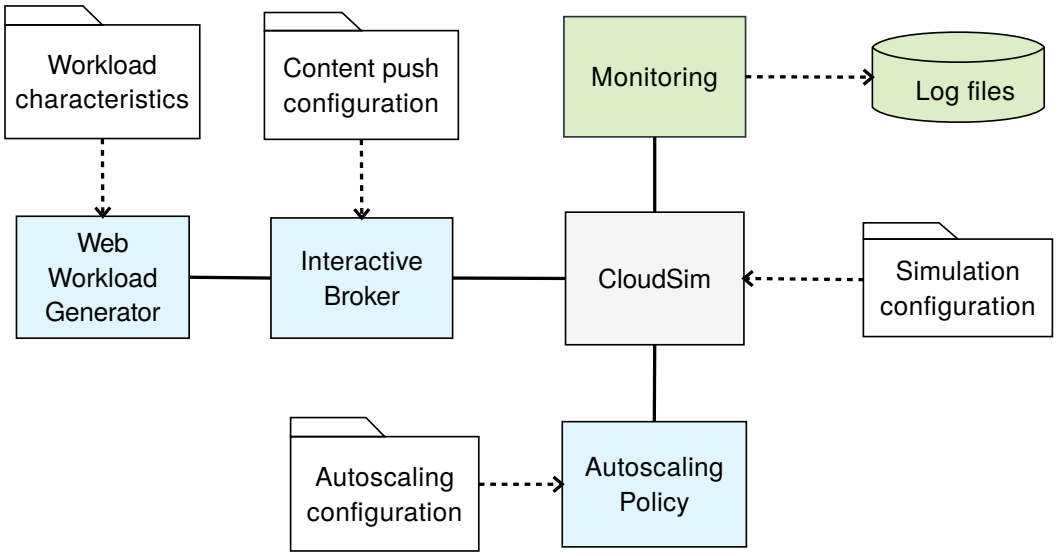


Figure 5.1: Architecture of the simulation environment

As can be seen, the extensions deal with a Web Workload Generator, an Interactive Broker, an Autoscaling Policy, and a Monitoring component. Configuration parameters are specified to customize the behavior of the various components. In details, the workload characteristics refer to the workload models identified from the analysis of measurements. The Workload Generator generates the workload according to these specifications. The produced workload is processed on the allocated VMs. Scheduling the workload for processing is performed by the broker. The broker is also configured in terms of the content push feature and the scheduling policy. The server push mechanisms enable Web servers to speculatively deliver content to the client without waiting for an explicit client request. The percentage of the content to be pushed by the Web server and the scheduling policy (i.e., Round Robin, Weighted Round Robin) are specified. The autoscaling policy is configured to monitor the load of the allocated VMs, and it takes decisions on allocation or deallocation of the VMs based on the monitoring data. The components of the simulation environment are interconnected and integrated and ready to generate, schedule and process the workload on the pool of resources.

5.2 Web workload generator

As already explained in Chapter 3, workload characterization is very important to understand the properties and the behavior of the workloads for different performance issues. For example, to devise and evaluate new resource provisioning policies and evaluate the QoS perceived by users – it requires a good understanding of the workload properties. The user behavior can be modeled by means of probability distributions. A probability distribution is a statistical function that describes the probability that a random variable equals some values within a given range. The range is defined by the minimum and maximum possible values. The distributions are characterized by factors such as mean, standard deviation, skewness and kurtosis.

Web workload generator component is responsible for generating the Web workload according to the probability distributions characterizing the attributes of the derived workload models. The workload generation process takes advantage of the Stochastic Simulation in Java (SSJ) software library[1]. The generated workload consists of a set of users, characterized by their arrival times and their browsing behavior.Indeed, each

[1] http://simul.iro.umontreal.ca/ssj

user can generate one or more sessions. A session is the set of the requests issued by a user given that the time between consecutive requests is smaller than a predefined value. Each request is described by a type (i.e., primary Web page, embedded object), a size and an arrival time. A Web page consists of an HTML file (i.e., primary page) and several embedded objects, Figure 5.2 shows the multi layer description of the workload adopted in this work.

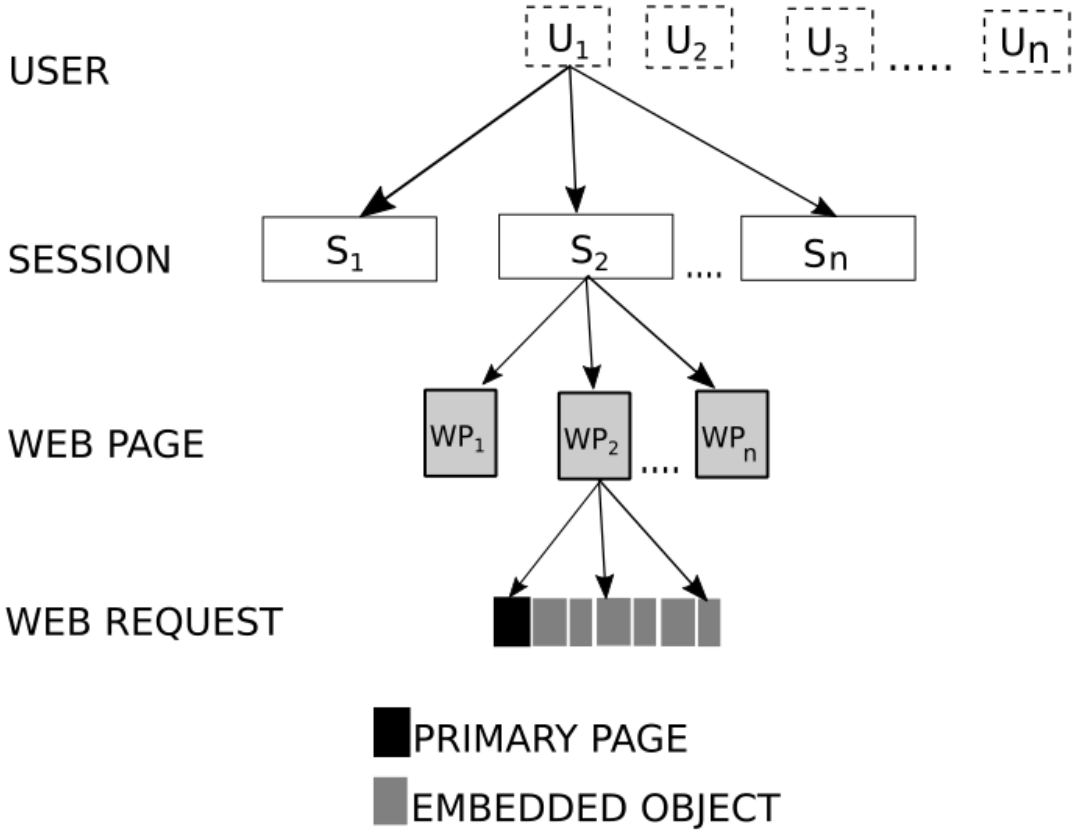


Figure 5.2: Multi layer workload description

The workload generator supports the generation of multiclass workloads. Each class describes the characteristics and properties of the users. As an example, the configurations of a class is shown in Figure 5.3. The user characteristics are specified in terms of the number of requested pages and the number of objects per page, the interarrival time between pages, the interarrival time between the page and its first embedded object, the interarrival time between embedded objects, the size of the page and size of the embedded objects. All these parameters are specified by the distributions identified by the corresponding models.

For example, the number of requested pages is defined by a uniform distribution in the range [0,10]. Similarly, the number of objects per page is defined by a uniform distribution in the range [1,35].

```
user.class.0.number.of.pages: UniformDist 1 10
user.class.0.number.of.objects.per.page: UniformDist 1 35
user.class.0.page.interarrivals: UniformDist 1 5
user.class.0.page2firstObj.interrarrival: DiscreteDistribution 0
   1 2 p 26471 11196 1477
user.class.0.object.interarrivals: MultiLevelUniform 0 1 2 37835
   1211 483
user.class.0.page.size:  MixedDist UniformDist 15529 15587 |
   UniformDist 23122 23597 | 11041 8574
user.class.0.object.size: MixedDist UniformDist 1 100 |
   UniformDist 550152 589949 | 50 50
```

Figure 5.3: Configuration of a user class

5.3 Autoscaling policy

As already discussed, autoscaling policies deal with dynamically scaling the amount of provisioned resources by taking into account the workload dynamics. In this thesis work, two reactive autoscaling policies have been studied and implemented. These policies react to the workload changes and base their decisions on the load conditions. Scaling actions are triggered by some predefined thresholds.

The austoscaling component developed and integrated within the CloudSim toolkit implements two policies, namely, a discrete sampling policy, and an average load policy. This component receives the measured indicators of the VM performance from the monitoring component.

5.3.1 Discrete sampling policy

The discrete sampling policy considers the status (idle, busy) of the CPU of all allocated VMs at specific time steps and computes their average utilization. A new VM is allocated when the average utilization of all VMs is above a given threshold for the selected time steps. Similarly, a VM is deallocated when the average utilization of all VMs is below a given threshold for the selected time steps.

Figure 5.4 shows a simple example of the computation of the average VM utilization. In the example, we consider four time steps and five VMs and 40% and 80% for the

scaling thresholds. As can be seen, at time step $T1$, three VMs are busy and two are idle, so the average utilization is 60%. At time step $T2$, two VMs are busy and three VMs are idle so the average utilization is 40%. At time step $T3$ and $T4$, the average utilization is equal to 80% and 20%, respectively. The number of steps and the

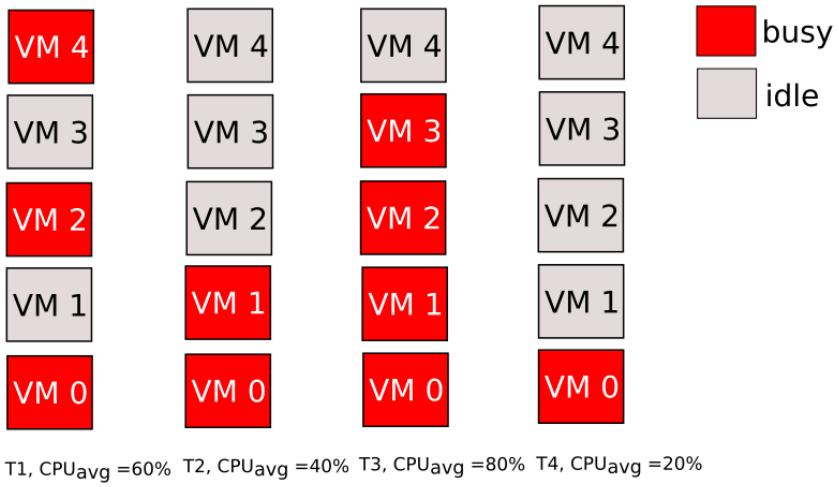


Figure 5.4: Discrete sampling policy over four time steps with five allocated VMs and lower and upper thresholds equal to 40% and 80%

thresholds are parameters specified in the custom configuration files. In this example, the computed average utilization of all VMs at the four time steps is neither above nor below the specified thresholds. Hence, no scaling actions are triggered.

5.3.2 Average load policy

The average load policy considers the status (busy, idle) of the CPU of all VMs at a fine-granularity sliding time window and computes the average utilization of each VM. The policy triggers the scaling actions according to one of these three conditions:

- the utilization of at least one VM must be above or below the threshold for the selected time window;
- the utilization of a given number/percentage of VMs must be above or below the threshold for the selected time window;
- the average utilization of all VMs must be above or below the threshold for the selected time window.

The average utilization of each VM relies on the values of the VM load provided by the monitoring component.
Figure 5.5 shows an example of application of this policy. The thresholds are set to 80% and 40% and some VMs are allocated. The scaling actions are taken according to the first condition and the average load is computed each 5 time units. For instance, at time window 1, the average load of the allocated VMs is computed at T5. Suppose that the average load of one VM is 80%, while the utilizations of the other VMs is neither below nor above the thresholds. Therefore, a new VM is allocated according to the first condition. Similarly, at time window 2, the average load of allocated VMs is

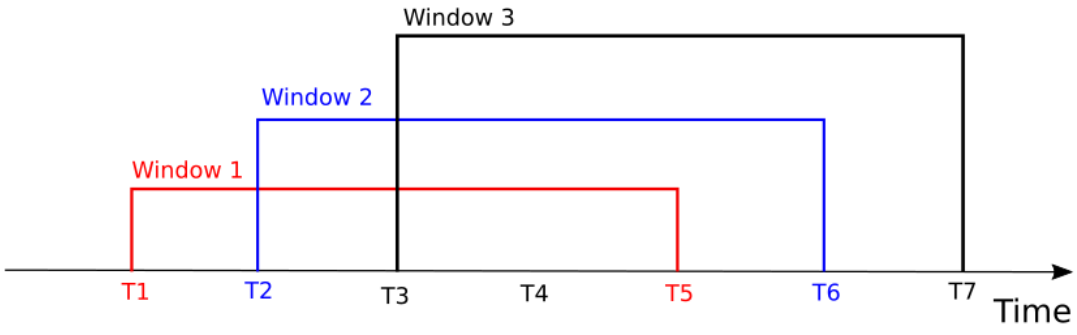


Figure 5.5: Average load policy with a time window of five time units

computed at T6. For example, we suppose that the average load of the allocated VMs

is neither below nor above the thresholds, so the policy does not trigger any scaling action. Finally, at time window 3, the average load is computed at T7. For example, the average load of one VM is lower than 40%, while the average load of the other VMs is neither below nor above the thresholds. So, this condition triggers the policy for deallocating one of the VMs.
These autoscaling policies have been developed and implemented into CloudSim toolkit and their computation mechanism is used in the simulation scenarios for taking scaling actions in response to the workload dynamics.

5.3.3 Autoscaling policy configuration

As already discussed, the discrete sampling policy and the average load policy trigger scaling actions for allocating or deallocating the VMs according to their load. These policies are invoked regularly over time. Different configurations can be set to cope with various simulation scenarios. The main configuration parameters can be summarized as follows:

- minimum and maximum number of VMs to be allocated;
- number of VMs to start with;
- cooling time, that is, the minimum time between to consecutive scaling actions;
- number of time steps (or time window) to be used for computing the VM load;
- number (or percentage) of VMs to be allocated/deallocated when a scaling action is taken.

5.4 Broker level extensions

As already explained in Sect. 4.2, the data center broker allows for managing the Cloud resources. This component is responsible for the coordination between different simulation components, i.e., monitoring, data center, Cloud Information Service. In addition, it plays as key role in managing the simulation flow, starting from the user requests scheduled according to one of the policies and assigning them to the VMs for processing. Moreover, the broker is responsible of the allocation and deallocation of the VMs.

5.4.1 User-based scheduling policy

The user-based scheduling policy is designed and developed for handling and scheduling the incoming user requests on the allocated VMs to avoid the overload of the VMs and to maximize the usage the allocated resources. The requests scheduling is performed according to one of the scheduling policies i.e., Round Robin, Weighted Round Robin. The broker invokes the specified policy for scheduling the user sessions on one of the VMs. The VM is selected according to one of the following approaches:

- **Round Robin:** this policy handles requests in a rotational order which all requests use a given resource without considering the actual load of the VMs.
- **Weighted Round Robin:** this policy assigns to each VM a weight that determines the fraction of requests to be sent to a given VM. The lower the weight, the larger the proportion of requests a VM receives.

The request scheduling is performed at simulation time according to the actual VM load.

5.4.2 Content push configuration

Pushing the requested HTML page with its embedded objects as responses without waiting for explicit requests from the client is a feature supported by the HTTP/2, the new version of the HTTP/1.1 protocol [77]. In the simulation environment, the server push mechanisms is implemented by the broker to improve the page load time. The configurations of this mechanism, such as the percentage and the types of the embedded objects to be pushed are set by the broker. Figure 5.6 shows a simple example of server push mechanism, where the client requests an HTML page and the server sends back the HTML file and its embedded objects i.e., a style sheet and an image. Three responses are sent back by the server, the *page.html* file and its two embedded objects (i.e., *style.css* and *image.png*).

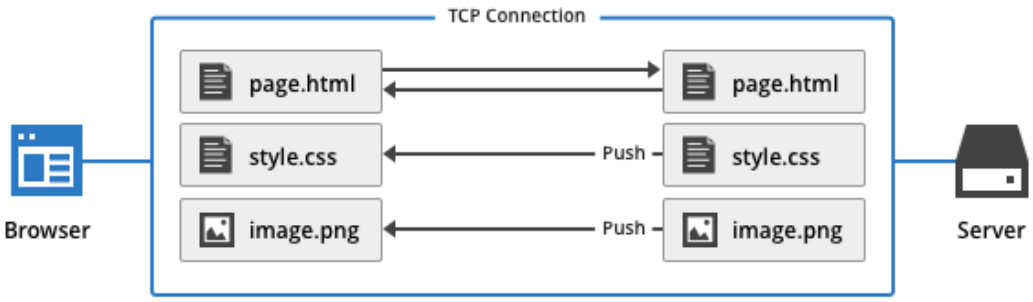


Figure 5.6: Example of server push feature

5.5 VM level extensions

As already discussed in Section 4.2, the *VM class* in the CloudSim represents the VM in the simulation environment. It runs inside a Host and processes Cloudlets. This processing happens according to a policy, defined by the CloudletScheduler. At VM level, we implemented a new concurrent-based scheduling policy for processing user requests. This policy is used to

- limit the maximum number of requests concurrently processed;
- implement a waiting queue;
- specify time-sharing granularity.

At the simulation time, the concurrent-based scheduling policy specifies a quantum time to each request and enables their processing concurrently. When the maximum number of requests concurrently processed is exceeded, the new arrival requests are moved to the waiting queue.

5.6 Monitoring component

The monitoring component is crucial for the autoscaling component since it collects measurements at the broker and VM side. In this thesis work, different monitoring strategies have been developed.
At the VM side, the monitoring component is invoked frequently during the simulation time by the broker to sample the status of the VMs at different time granularity. Monitoring consists of collecting information on the VM status, such as the number of requests concurrently processed, number of waiting requests. The performance of Web requests, that is, their processing time and waiting time, can also be monitored.
At the broker side, the monitoring component collects information about the number of allocated or deallocated VMs. The collected information are used by other simulation components such as autoscaling, and the broker itself for the VM allocation/deallocation.

5.7 Simulation configurations

Different Cloud environment can be simulated by varying changes according to the workload characteristics and the configurations of the simulator components.
Various parameters are set for each simulation experiment, such as number of VMs to start with and the VM boot time, that is, the time needed by a newly allocated VM to be up and running for processing the user requests. Time sharing quantum is another configuration parameter used to set the time given to each request during its processing on a VM. Similarly, the monitoring resolution is also set which specify the granularity of sampling time used by the monitoring component to collect measurements of the VM status.

Chapter 6

EXPERIMENTAL RESULTS

This chapter describes the dataset used in the experiments as well as the results of the Web workload characterization. The simulation scenarios developed for testing the behavior of the autoscaling policy under different HTTP/2 content push configurations and the results of these experiments are also presented.

6.1 Dataset description

The dataset used in this work is represented by the logs collected by the Web server that hosts the official site of the University of Pavia. These logs are stored according to the Extended Log File Format. In particular, each record contains various information, such as the IP address of the client that issued the HTTP request, a timestamp associated with the request, the User Agent string and the status code of the HTTP response. Table 6.1 summarizes the characteristics of the dataset. The data refers to four weeks between 24/01/2016 and 21/02/2016.
A preliminary analysis was applied for filtering, reformatting, and summarizing the dataset. Users have been identified using the pair, IP address and User Agent. Moreover, within user activity, sessions have been identified. A session is defined as the sequence of requests whose interarrival time, that is, the time between two consecutive requests, is less that a predefined threshold.

The users have been classified into users who requested only one page and users who requested more than one page. As results 252,474 users have been identified, one third of the users requesting only one page.

Logging period	4 weeks
Number of requests	8,608,774

Table 6.1: Main characteristics of the log files of the University of Pavia

6.2 Workload models

To identify workload models we analyze the user behavior in terms of its requests to the Web server. In particular, we describe each user by nine parameters, namely: total number of requests of primary pages, total number of requests of embedded objects, average interarrival time between primary pages, average interarrival time between a primary page and its first embedded object, average interarrival time between the first and last embedded object, average interarrival time between embedded objects, total size of the primary pages, number of requests of primary pages without embedded objects, total size of the embedded objects. Various statistical techniques are applied to analyze the user behavior and uncover differences and similarities among them. In particular, it is necessary to compute the correlation coefficient between parameters as to remove highly correlated parameters.

The Pearson correlation coefficient is computed for the users who requested more than one page each – which measures the linear relationship between two variables. For example, the correlation coefficient between the number of pages without embedded objects and the number of primary pages is equal to 96.78%. As result, the number of pages without embedded objects parameter has been removed.

In addition, we analyze the parameter distributions for the identification of the outliers. In fact, the outliers, that is, the components whose parameter values are very different from the typical ones, may distort the classification process. Hence, the original data set has to be trimmed by removing the components having one or more parameters with values greater than a predetermined percentile. In particular, the 99.2 and 99.5 percentiles are considered for the number of pages and number of embedded objects, respectively. Figure 6.1 shows the percentiles of the page interarrival time of the users who requested more than one page. As can be seen, values larger than 99.7 percentile are quite different from the other values.

Figure 6.2 shows the cumulative distribution function of the page interarrival time, highlighting the 99.7 percentiles that has been chosen to remove outliers. The total number of users who have requested more than one page is 165,518. As result of the filtering, 0.02% of the users who requesting more than one page have been removed.

The k-means clustering algorithm has been applied to group users with similar characteristics. Number of primary pages, number of embedded objects, interarrival time of the primary pages, average size of primary pages and average size of embedded objects

```
330002.790  345590.200  350582.100  360587.383  374514.955  387222.880
     98.4%       98.5%       98.6%       98.7%       98.8%       98.9%
403244.430  419927.000  432602.807  449921.135  472051.460  498902.250
     99.0%       99.1%       99.2%       99.3%       99.4%       99.5%
522360.167  554605.940  590195.920  615383.908  665141.980  724292.375
     99.6%       99.7%       99.8%       99.9%      100.0%
805837.940  906655.507 1082718.080 1423591.375 2259468.000
```

Figure 6.1: Percentile values of page interarrival time of the users who requested more than one page

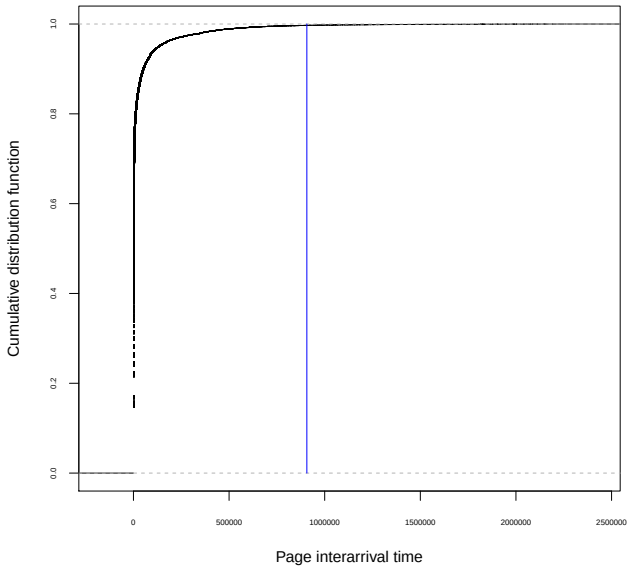


Figure 6.2: Cumulative distribution function of the page interarrival time

are the parameters used in the clustering.

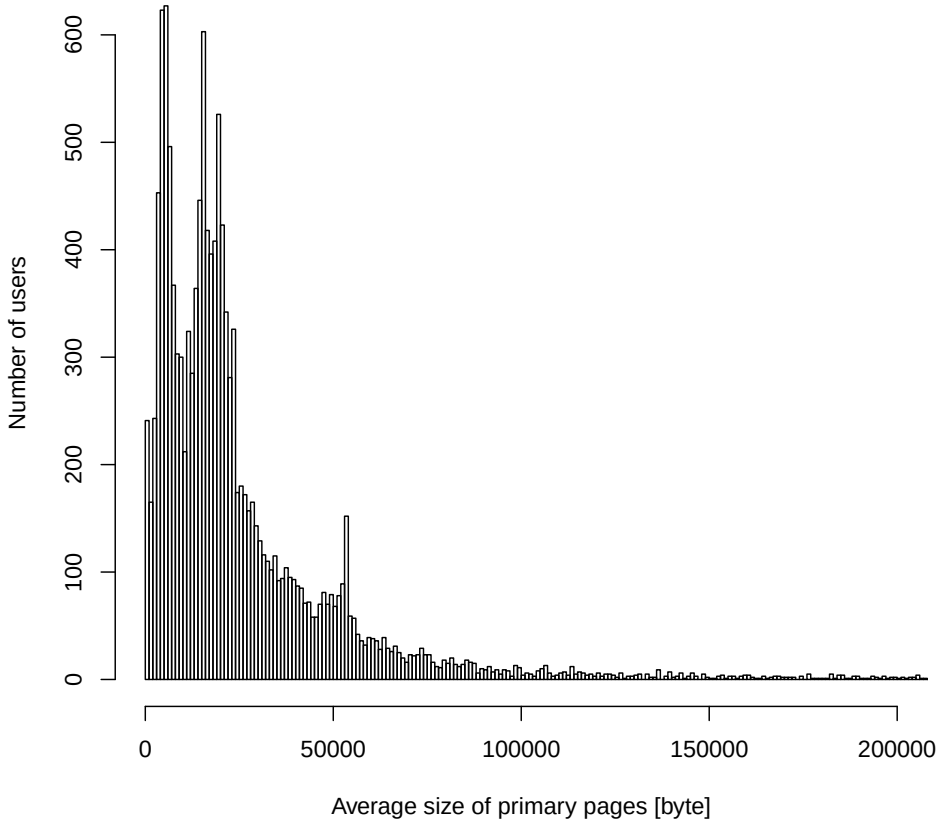
The users are partitioned into four groups, described by their centroid, i.e., the geometric centers of the clusters. The characteristics of the four clusters are shown in Table 6.2. As can be seen, cluster 1 is the largest one, grouping 49% of the users. The user requests have very short interarrival times, compared with those of other users. Cluster 3 groups 40% of the users requesting large embedded objects. embedded objects. Cluster 2 groups the users who request the largest number of pages and embedded objects, that is, on average 32.61 and 83, respectively. Moreover, the users of Cluster 4 request pages whose size is very large, i.e., 111,852 bytes, with longest interarrival time.

To summarize the behavior of the various parameters by means of probabilistic distri-

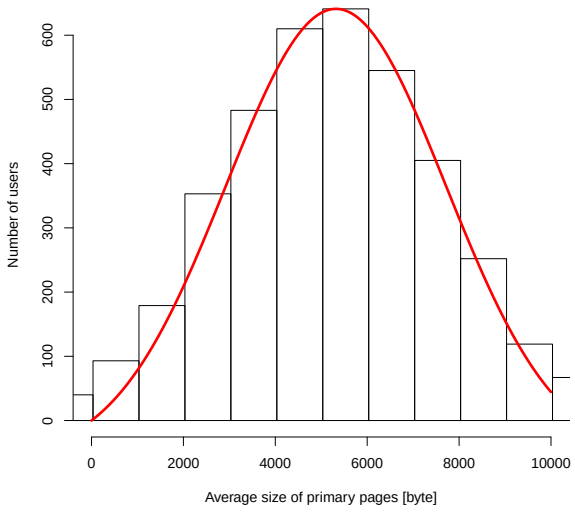
	Cluster 1	Cluster 2	Cluster 3	Cluster 4
Number of users	79,023	13,864	65,077	4,554
Number of primary pages	4.81	32.61	5.29	3.63
Number of embedded objects	13.4	83	38	18.5
Primary page interarrival time [second]	9.305	17.917	10.508	43.301
Primary page size [byte]	53,737	31,880	23,169	111,852
Embedded object size [byte]	12,828	35,573	66,592	15,403

Table 6.2: Centroids of the four clusters

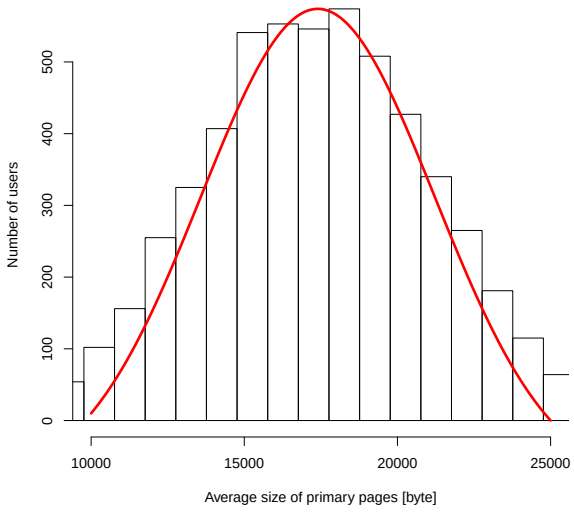
butions, fitting techniques are applied. A visual inspection of the parameters show that their patterns can be modeled by different types of distribution. Thus, we use piecewise functions. Figure 6.3(a) shows the distribution of the average size of primary pages in cluster 2. As can be seen, the pattern is not clear and it does not fit to individual distribution. In addition, some peaks can change the behavior of the pattern, therefore, it is necessary to fit the distribution to the proper ones. We fit this distribution by means of two Gaussian distributions in the ranges [0,10000[and [10000,25000] and of one uniform. Figure 6.3(b) shows the Gaussian distribution in range [0,10000[with standard deviation 2,395.44 and mean 5,319.74. Figure 6.3(c) shows the Gaussian distribution in range [10000,25000] with standard deviation 3,817.667 and mean 17,399.8. Primary page size is modeled as uniform distribution in the range [25001.19,119084.5] with frequency 4518 and mean 72,042.23. Similarly, the other parameters of the clusters have been fitted with proper distributions in order to obtain different workload models.
Users who have requested only one page are modeled separately since for these users many parameters previously are meaningless. These users are 86,968 and, after applying the filtering process, they are 40,268 i.e., 46.3%. Table 6.3 shows some of their characteristics.



(a)



(b)



(c)

Figure 6.3: Average size of primary pages in Cluster 2 (a)) and (Gaussian distribution fitted in the range [0,10000[(b)) and (Gaussian distribution fitted in the range [10000,25000] (c)).

	Users
Number of users	40,268
Average number of embedded objects	21.81
Average time between primary page and first embedded object [second]	0.8419
Average interarrival time between embedded objects [second]	0.39621
Average primary page size [byte]	20,825
Average embedded objects size [byte]	646,726

Table 6.3: Characteristics of the users who request only one page

6.3 Simulation experiments

The derived workload models, described in the previous section are used to drive the simulation experiments. A single class workload has been used in the simulations and, in particular, the users who request one page have been considered. This section presents the results of four simulation experiments aimed at assessing the behavior of the autoscaling policies (i.e., discrete sampling, average load) and testing the HTTP/2 content push feature. As already pointed out, server push optimization mechanisms allow Web servers to speculatively send resources without waiting for explicit client requests. The push configuration parameters identify the bundle of Web resources to be pushed.

6.3.1 User arrival patterns

The Web workload generator reproduces the workload according to the identified workload models. In the simulation experiments, more than 330,000 HTTP requests for Web pages have been generated. Figure 6.4 shows the daily arrival pattern of the requests as a function of time. As can be seen, the workload intensity is characterized by a clear diurnal pattern, with much fewer requests – as little as 42 requests per minute – during the night and early morning hours, and a peak of about 400 requests per minute around noon.

To describe the composition of the Web pages in terms of number and size of embedded objects, we analyze the structural properties of the Website (see the snapshot of Figure 6.5). On average the Web pages reference 40 embedded objects each. The processing time of each object is set proportional to its size. In detail, the processing time

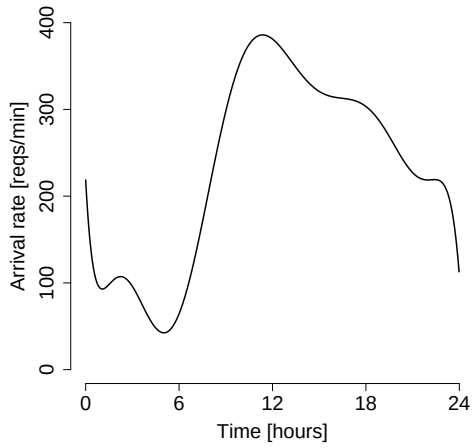


Figure 6.4: Daily request arrival pattern.

is described by two equally probable uniform distributions, one in the range $[0.2, 2]$ μs and the other in the range $[110, 118]$ μs.

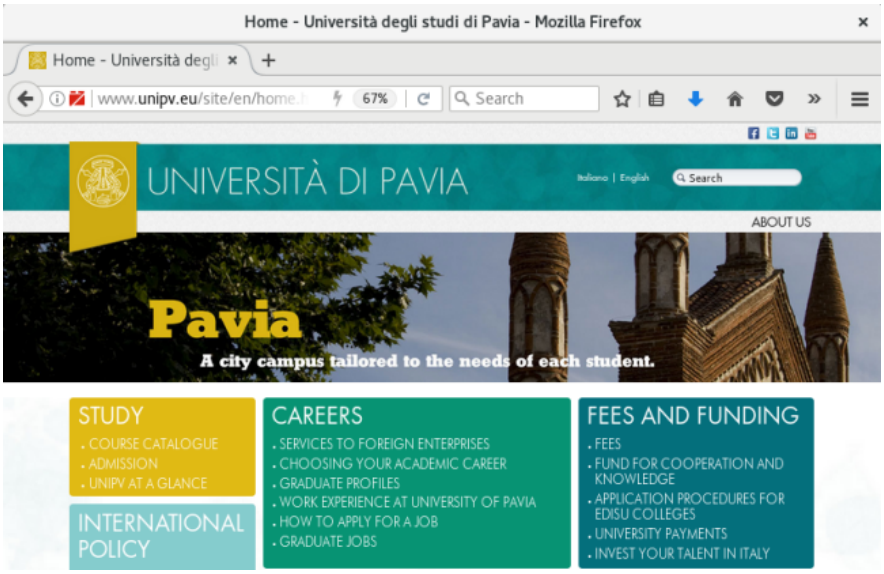


Figure 6.5: Snapshot of the home page of the Website of the University of Pavia.

Another important property of the workload is the interarrival time between two consecutive requests to the embedded objects of a given page. Since these times are usually quite small, that is, one second or less, we model them using two uniform distributions in the range $[0, 1]$ s and $[1, 2]$ s, respectively. The probabilities associated with these distributions are 0.97 and 0.03.

6.3.2 Simulation scenario

The experiments performed follow the simulation scenario depicted in Figure 6.6. As can be seen, a pool of VMs is allocated to process Web requests. Some VMs are

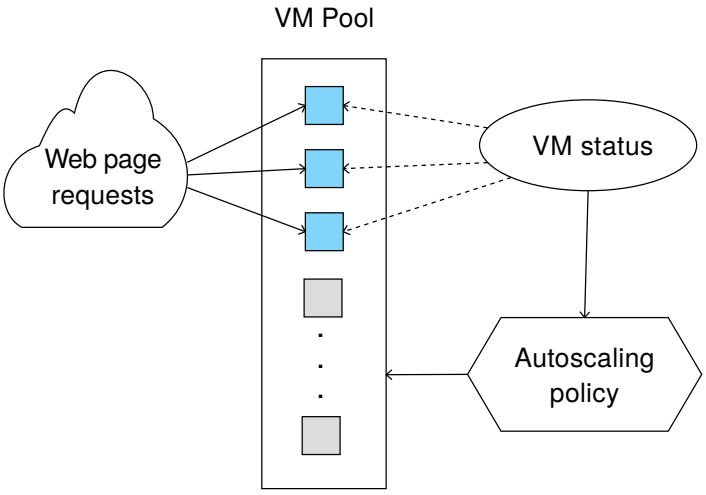


Figure 6.6: Experimental scenario.

allocated and some others are ready to be allocated depending on the workload intensity. The autoscaling policy frequently checks the load of the allocated VMs by receiving the monitoring data to react to the workload changes by allocating/deallocating VMs. Several experiments have been carried out to select the most appropriate values of the thresholds that trigger the autoscaling policy as well as the number of time steps or time window in order to adapt the VM allocation/deallocation to the workload dynamics, while minimizing the number of scaling operations. In details, the upper threshold is set to 0.8 to maximize the VM utilization, while the lower threshold is set to 0.4 to avoid fluctuation in VM allocation/deallocation. Note that, if the number of time steps and the time window are too large, the autoscaling policy reacts slowly to the workload changes.

Table 6.4 summarizes the parameters used in the simulations. In particular, these parameters specify the minimum and the maximum number of VMs to be allocated (equal to 1 and 100, respectively), the initial number of VMs (equal to 20). The boot time of a VM is set to 30 seconds. In addition, to avoid oscillations in the number of allocated/deallocated VMs, the cooling time, i.e., the minimum time between two scaling actions, is set to 40 seconds.

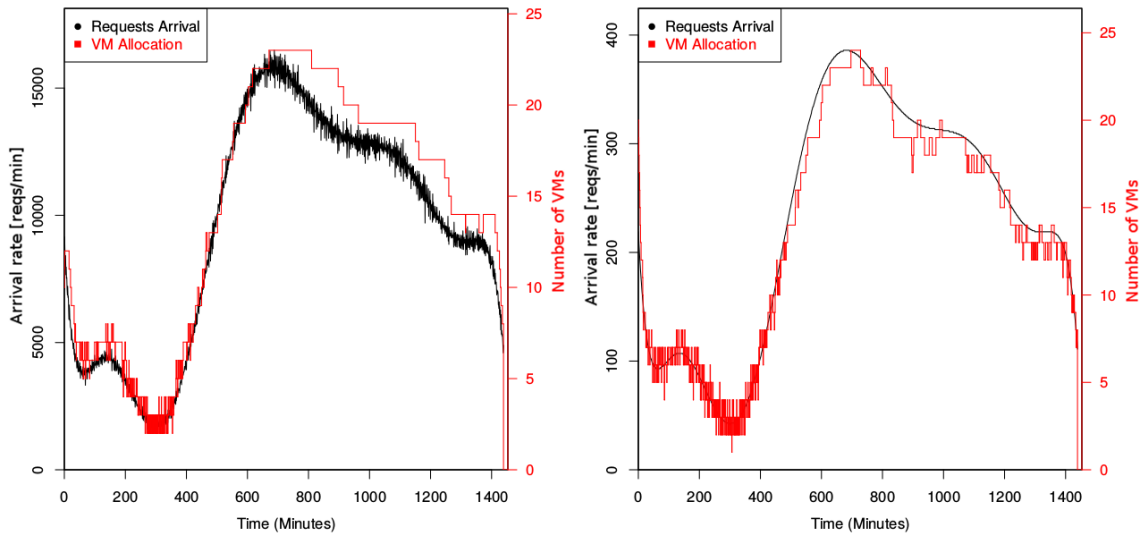
Number of VMs			Boot time	Cooling time	Thresholds		Monitoring resolution
Min	Max	Start	[s]	[s]	lower	upper	[s]
1	100	20	30	40	0.4	0.8	0.1

Table 6.4: Simulation parameters

As explained in Sect. 5.3, the autoscaling component developed in this thesis work implements two reactive autoscaling policies (i.e., Discrete sampling, Average load). These policies allocate/deallocate the VMs according to their actual load on a given number of time steps or time window. The simulation experiments are performed under different configurations aimed at assessing the behavior of autoscaling policies and testing the HTTP/2 content push mechanisms.
The discrete sampling and average load policies have been evaluated using performance metrics, such as number of scaling operations, VM utilization, page load time and number of allocated VMs.

6.3.3 Discrete sampling policy

The behavior of the discrete sampling policy in response to the workload changes has been tested and evaluated. In particular, two simulations have been carried out under different content push configuration and the VMs usage is computed over four time steps. The first experiment does not consider any content push. This means that each embedded object is sent as response of a specific HTTP request. The second experiment considers full content push, that is the embedded objects are sent to the client without being explicitly requested.
Figure 6.7 shows the number of allocated VMs as a function of time. The request arrival rate is also displayed. In particular, Figure 6.7(a) refers to the experiment with no content push, while Figure 6.7(b) refers to the experiment with full content push. The scaling operations in the experiment with no push account for 131 and 133 for scaling up and down, respectively and the maximum number of allocated VMs is 23. The scaling operations in the experiment with full push account 253 and 265 for scaling up and down, respectively and the maximum number of allocated VMs is 24.
Note that, the request arrival rate is higher in the experiment with no push 6.7(a) compared to the one with full push 6.7(b), since the Web server receives a separate request for each embedded object.



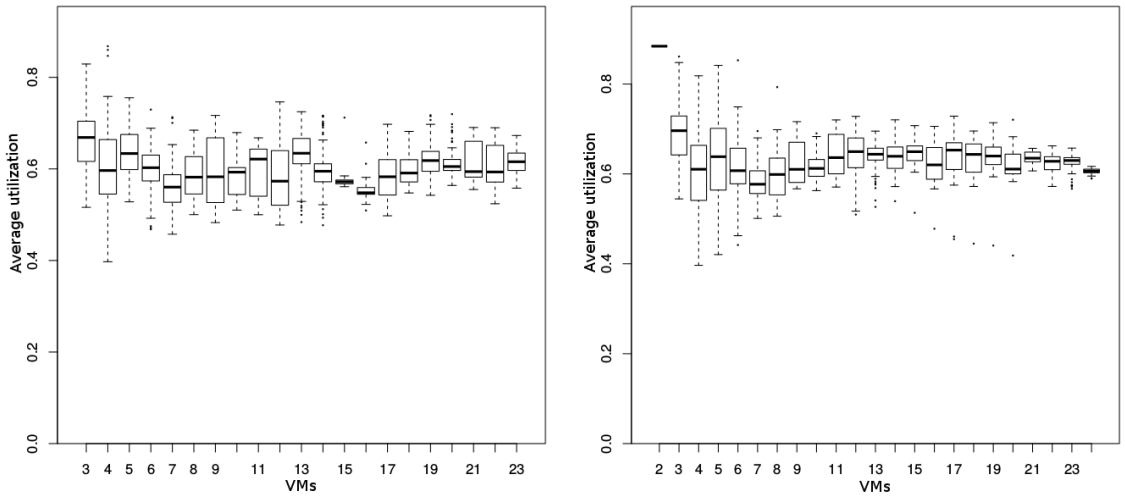
(a) (b)

Figure 6.7: Step functions of the number of allocated VMs as a function of simulated time with no push (a) and full push (b) of the Web content

The boxplots of Figure 6.8 summarize the actual utilization of the VMs as a function of the number of allocated VMs. Figure 6.8(a) refers to the experiment with no push. The average utilization of the allocated VMs is 0.60 and the maximum is 0.87. Figure 6.8(b) refers to the experiment with full push. The average utilization of the allocated VMs is 0.63 and the maximum is 0.88. As can be seen, in both experiments, increasing the number of allocated VMs leads to a balanced VM utilization.

Page load time, that is, the time needed by the Web server to load the HTML file with its embedded objects, is a good performance metric for evaluating the autoscaling policies. Figure 6.9 shows the cumulative distribution functions of the page load time as function of the simulated time. We notice very different behaviors with much smaller times in the case of full push.

Table 6.5 presents some statistics of the page load time. As can be seen, the page load time is significantly smaller in the case of full push, in particular its average is 2.8 s, while in case of no push it is 21.3 s.



(a) (b)

Figure 6.8: Boxplots of the VM utilization for the no push (a) and full push (b) experiments

	Avg.	Median	99-th percentile
No push	21.3	21.1	26.9
Full push	2.8	2.4	9.1

Table 6.5: Page load time expressed in seconds under no push and full push configurations

The autoscaling behavior in response to the workload changes has been further analyzed by looking at the status (idle, busy) of the allocated VMs. In particular, Figure 6.10 shows the number of allocated VMs (black step function) compared to the number of busy VMs (blue curve) as a function of time. The red area in the diagram represents the unused VMs allocated but in idle state. On average we discovered that 40% of the allocated VMs in the no push are idle, while 37% the allocated VMs in the full push. As can be seen, in the case of full push 6.10(b) the policy leads to better VM utilization and VM deallocation.

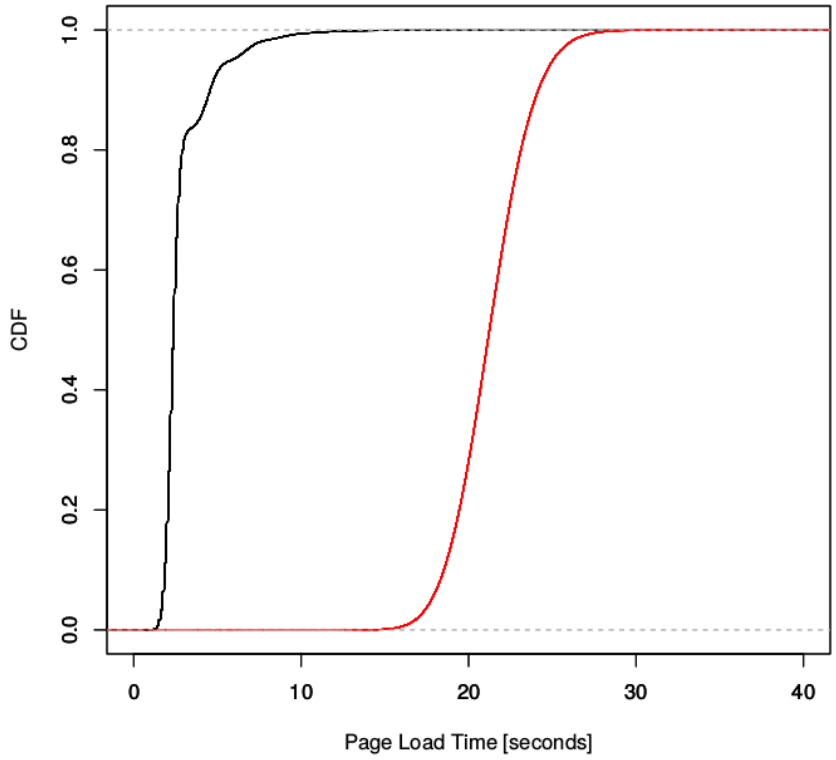
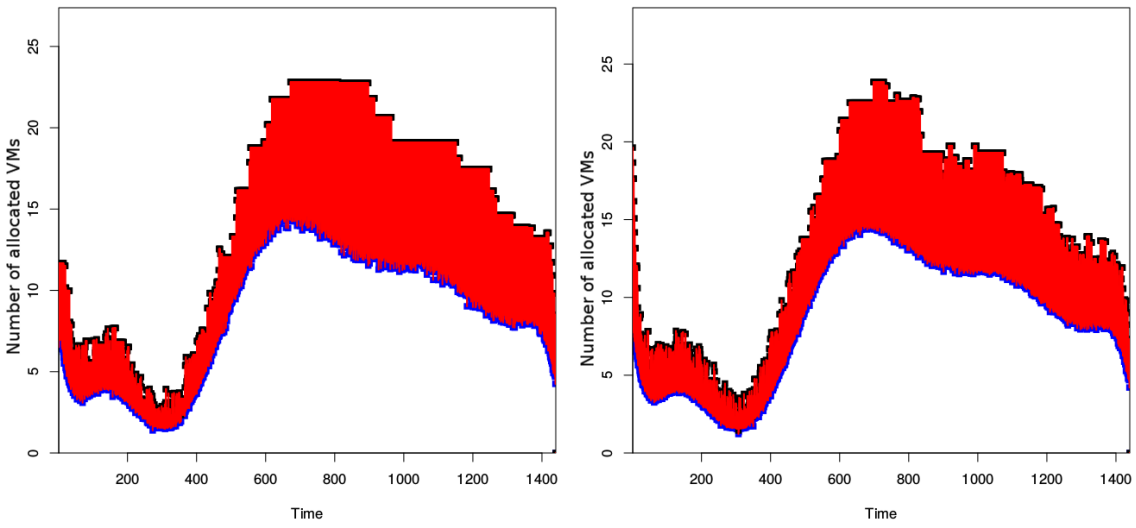


Figure 6.9: Cumulative distribution functions of the page load time with full push (black curve) and no push (red curve)



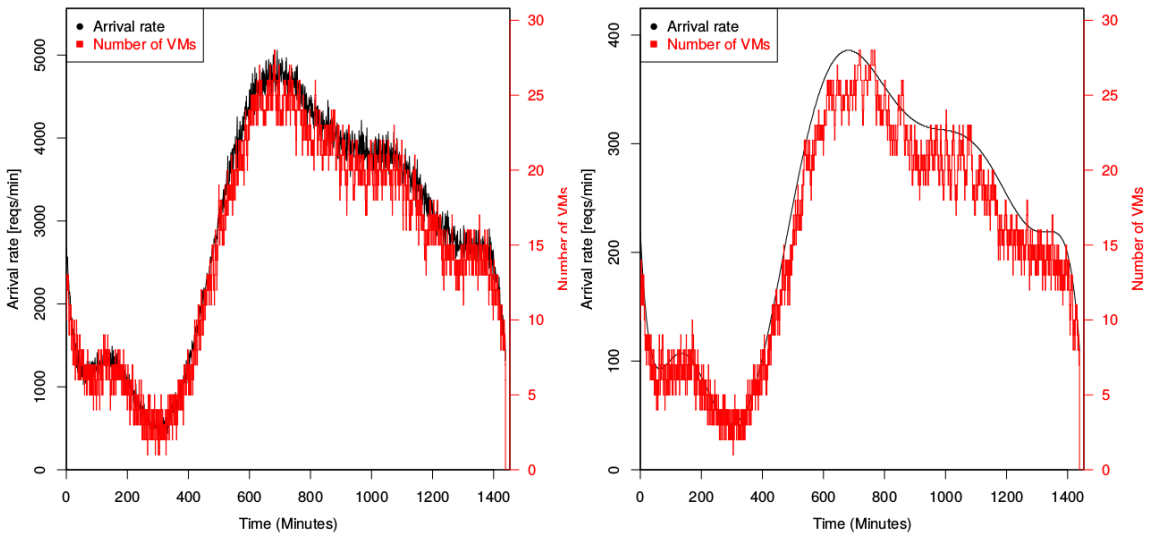
(a) (b)

Figure 6.10: Number of allocated VMs and of busy VMs as a function of time for no push (a) and full push (b)

6.3.4 Average load policy

Two experiments aimed at assessing the behavior of the average load policy have been performed as well. The first experiment considers partial push, in which 45% of the embedded objects are pushed by the Web server. The second experiment considers full content push. In addition, the VMs usage is computed by choosing a time window value of 5 seconds. The results of these experiments are explained in this section.

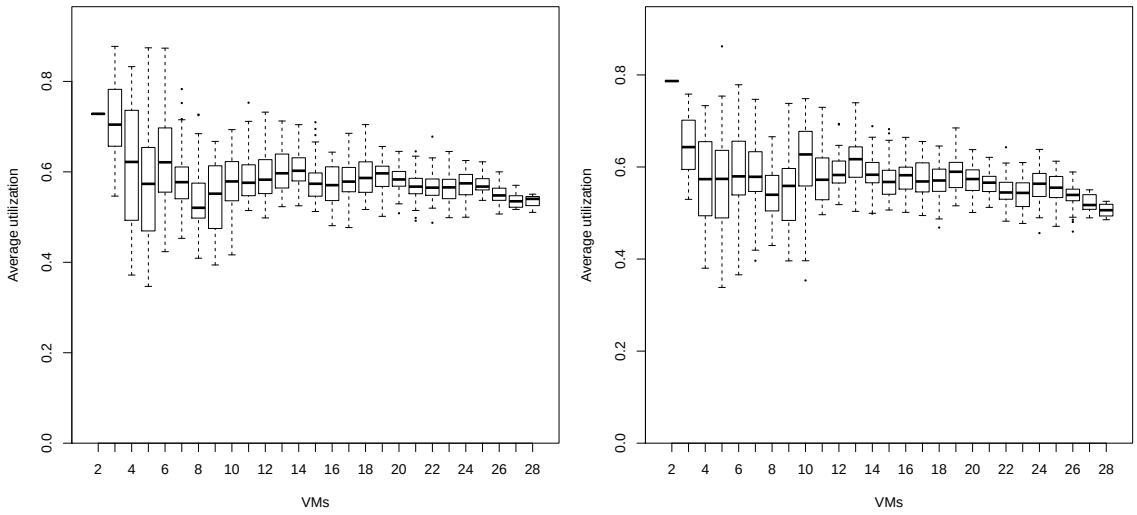
Firstly, we evaluate how the average load policy reacts to the workload dynamics. The number of allocated VMs and request arrival rate as a function of time are shown in Figure 6.11. Figure 6.11(a) refers to the experiment with partial content push, while Figure 6.11(b) refers to the experiment with full content push. The VM allocation/deallocation is evaluated by analyzing how many scaling operations are performed. In the partial push experiment a total of 992 and 996 scaling up and down operations are counted, whereas in the full push they are 835 and 838, respectively. In both experiments, at maximum 28 VMs are allocated.



(a) (b)

Figure 6.11: Step functions as a function of simulated time with partial push (a) and full push (b)

The average VMs utilization have been evaluated in both experiments. The actual utilization of the VMs as a function of the number of allocated VMs is summarized by the boxplots of Figure 6.12. Figure 6.12(a) refers to the experiment with partial push.



(a) (b)

Figure 6.12: Boxplots of the VMs utilization of for the partial push (a) and full push (b)

The average VMs utilization is 0.58 and the maximum is 0.88. Figure 6.12(b) refers to the experiment with full push. The average VMs utilization is 0.57 and the maximum is 0.86. As can be seen, the behavior is similar as in the discrete sampling policy.
Figure 6.13 shows the results of the cumulative distribution function of the page load time as function of time for the two experiments. Page load times exhibit very different behaviors with much smaller times in the case of full push.

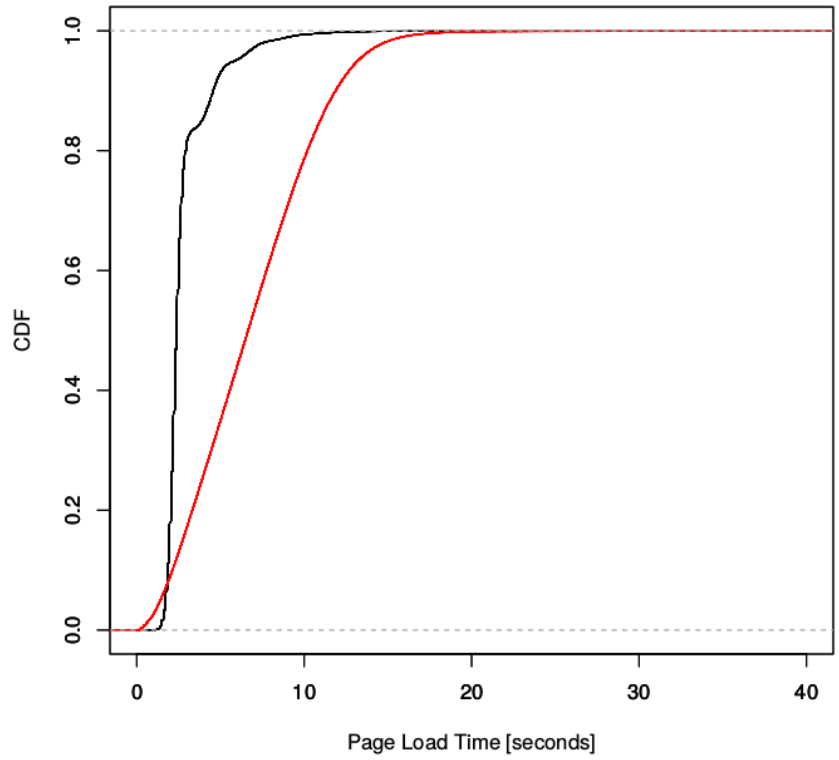


Figure 6.13: Cumulative distribution functions of the page load time with full push (black curve) and with partial push (red curve)

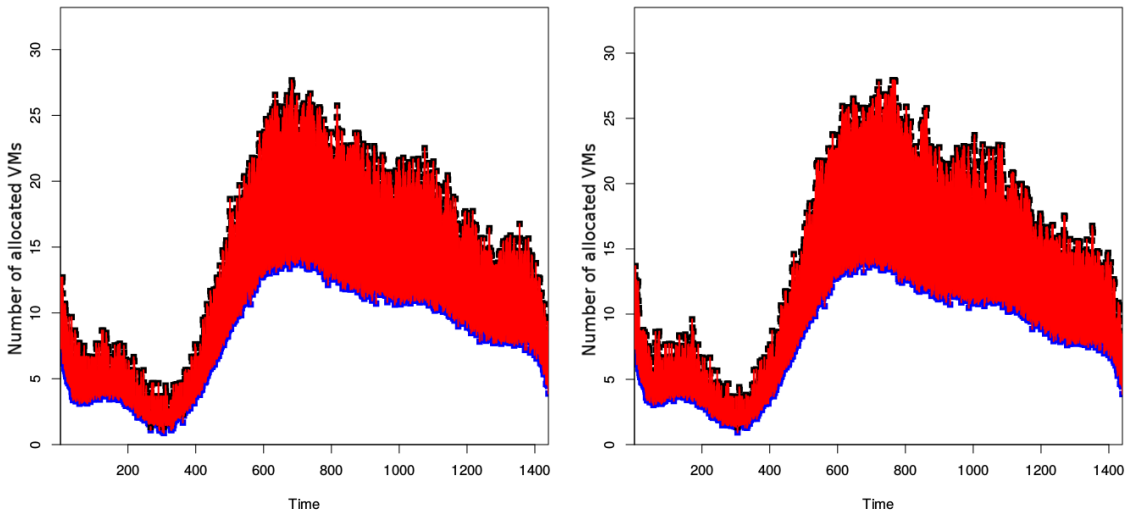
Table 6.5 presents some statistics of the page load time. As can be seen, the page load time decreases when the number of pushed objects increases. The page load time is significantly smaller in the case of full push, in particular the average page download is 2.8 s while in case of partial push is 6.9 s.

	Avg.	Median	99-th percentile
Partial push	6.9	6.7	15.9
Full push	2.8	2.4	9.1

Table 6.6: Page load time expressed in seconds under partial and full push configurations.

The average load policy behavior when allocating/deallocating VMs in response to the workload changes has been analyzed by looking at the status (idle, busy) of the

allocated VMs. In particular, Figure 6.14 shows the number of allocated VMs (black step function) compared to the number of busy VMs (blue curve) as a function of time. The red area in the diagram represents the unused VMs allocated but in idle state. We discovered that about 42% of the allocated VMs are idle in the partial push, while 43% of the allocated VMs in the full push.



(a) (b)

Figure 6.14: Number of allocated VMs and of busy VMs as a function of time for partial content push (a) and full content push (b)

6.3.5 Summary

The simulation experiments have shown that autoscaling considered in this work react well to the workload changes by allocating or deallocating the VMs. In details, the scaling actions performed by discrete sampling are fewer than the scaling actions performed by average load. This leads to fewer fluctuation in VM allocation or deallocation.
The experiments have also shown an overall benefit in adopting the push feature in terms of page load time. Page load times exhibit very different behaviors with much smaller times in the case of full push compared with no push and partial push.
Moreover, by comparing the two policies under full content push, we notice that more than half of the allocated VMs are idle. This emphasizes the importance of autoscaling policies for saving costs by reducing the number of VMs. These results have been published in a conference paper (see [78]).

Chapter 7

CONCLUSIONS

This thesis work focuses on the study of Cloud workloads and on characterization of Web workloads with the objective of devising models to be used to test the performance of autoscaling policies. The workload characterization is based on measurements collected on a Web server. Data analysis and exploratory techniques have been applied for identifying the parameters that describe the user behavior. Moreover, workload models have been derived by applying clustering and fitting techniques.

A simulation environment based on the CloudSim simulation toolkit has been designed and developed to exploit a Web workload on a realistic Cloud infrastructure and implement state-of-the-art reactive autoscaling policies. These policies allocate/deallocate the VMs according to their resource usage evaluated on a given number of time steps or time window.

Various experiments aimed at assessing the impact of HTTP/2 content push mechanism on Cloud environment have been performed. The benefits of the server push mechanisms have been evaluated using performance metrics, such as page load time, VM utilization and number of allocated VMs. The autoscaling policies tested in the experiments lead to rather balanced utilizations across VMs, when a larger number of VMs allocated. The results of the simulation experiments have shown an overall benefit in adopting the server push mechanism. In particular, the page load time is significantly reduced and the amount of content being pushed does not influence the behavior of the autoscaling policies.

7.1 Future work

As a future work it is possible to model different scenarios and workload classes. Reactive and proactive autoscaling polices will be designed and tested to study their sensitivity to push configuration parameters. In addition, the various features of the HTTP/2 protocol will be tested on benchmarks deployed for the purpose and executed on real a Cloud infrastructure that relies on the OpenStack software.

Bibliography

[1] Cisco, "Cisco global Cloud index: Forecast and methodology, 2014-2019," http://www.cisco.com/c/en/us/solutions/collateral/service-provider/global-cloud-index-gci/Cloud_Index_White_Paper.pdf, 2015.

[2] M. Ghahramani, M. Zhou, and C. Hon, "Toward Cloud computing QoS architecture: Analysis of Cloud systems and Cloud services," *IEEE/CAA Journal of Automatica Sinica*, pp. 6–18, 2017.

[3] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," National Institute of Standards and Technology, Tech. Rep. SP 800-145, 2011.

[4] J. E. Smith and R. Nair, "The Architecture of Virtual Machines," *Computer*, pp. 32–38, 2005.

[5] N. R. Herbst, S. Kounev, and R. Reussner, "Elasticity in Cloud Computing: What It Is, and What It Is Not," in *Proc. of the 10th International Conference on Autonomic Computing (ICAC'13)*. USENIX Association, 2013, pp. 23–27.

[6] N. M. Calcavecchia, B. A. Caprarescu, E. D. Nitto, D. J. Dubois, and D. Petcu, "DEPAS: a decentralized probabilistic algorithm for auto-scaling," *Computing*, vol. 94, no. 8-10, pp. 701–730, 2012.

[7] C. Qu, R. N. Calheiros, and R. Buyya, "Auto-scaling Web Applications in Clouds: A Taxonomy and Survey," *ACM Computing Surveys (CSUR'17)*, vol. 9, no. 4, pp. 39:1–39:34, 2017.

[8] M. S. Aslanpour, M. Ghobaei-Arani, and A. N. Toosi, "Auto-scaling Web applications in Clouds: A cost-aware approach," *Journal of Network and Computer Applications*, vol. 95, pp. 26 – 41, 2017.

[9] H. Arabnejad, C. Pahl, P. Jamshidi, and G. Estrada, "A Comparison of Reinforcement Learning Techniques for Fuzzy Cloud Auto-Scaling," in *Proc. of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid'17)*. IEEE, 2017, pp. 64–73.

[10] P. Tang, F. Li, W. Zhou, W. Hu, and L. Yang, "Efficient Auto-Scaling Approach in the Telco Cloud Using Self-Learning Algorithm," in *Proc. of the IEEE Global Communications Conference (GLOBECOM'15)*, 2015, pp. 1–6.

[11] T. Lorido-Botran, J. Miguel-Alonso, and J. A. Lozano, "A Review of Auto-scaling Techniques for Elastic Applications in Cloud Environments," *Journal of Grid Computing*, vol. 12, no. 4, pp. 559–592, 2014.

[12] A. Gandhi, M. Harchol-Balter, R. Raghunathan, and M. A. Kozuch, "AutoScale: Dynamic, Robust Capacity Management for Multi-Tier Data Centers," *ACM Transactions on Computer Systems*, vol. 30, no. 4, pp. 14:1–14:26, 2012.

[13] F. Farahnakian, P. Liljeberg, and J. Plosila, "LiRCUP: Linear Regression Based CPU Usage Prediction Algorithm for Live Migration of Virtual Machines in Data Centers," in *Proc. of the 39th Euromicro Conference on Software Engineering and Advanced Applications*, 2013, pp. 357–364.

[14] A. Evangelidis, D. Parker, and R. Bahsoon, "Performance Modelling and Verification of Cloud-Based Auto-Scaling Policies," in *Proc. of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID'17)*, 2017, pp. 355–364.

[15] X. Dutreilh, A. Moreau, J. Malenfant, N. Rivierre, and I. Truck, "From Data Center Resource Allocation to Control Theory and Back," in *Proc. of the IEEE 3rd International Conference on Cloud Computing*, 2010, pp. 410–417.

[16] R. Han, L. Guo, M. M. Ghanem, and Y. Guo, "Lightweight Resource Scaling for Cloud Applications," in *Proc. of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computin (CCGrid'12)*, 2012, pp. 644–651.

[17] M. Z. Hasan, E. Magana, A. Clemm, L. Tucker, and S. L. D. Gudreddi, "Integrated and autonomic Cloud resource scaling," in *Proc. of the IEEE Network Operations and Management Symposium*, 2012, pp. 1327–1334.

[18] T. C. Chieu, A. Mohindra, A. A. Karve, and A. Segal, "Dynamic Scaling of Web Applications in a Virtualized Cloud Computing Environment," in *Proc. of the IEEE International Conference on e-Business Engineering*, 2009, pp. 281–286.

[19] F. Al-Haidari, M. Sqalli, and K. Salah, "Impact of CPU Utilization Thresholds and Scaling Size on Autoscaling Cloud Resources," in *Proc. of the IEEE 5th International Conference on Cloud Computing Technology and Science*, 2013, pp. 256–261.

[20] M. Daswani, P. Sunehag, and M. Hutter, "Q-learning for history-based reinforcement learning," in *Proc. of the 5th Asian Conference on Machine Learning*, vol. 29. PMLR, 2013, pp. 213–228.

[21] P. Jamshidi, A. M. Sharifloo, C. Pahl, A. Metzger, and G. Estrada, "Self-Learning Cloud Controllers: Fuzzy Q-Learning for Knowledge Evolution," in *Proc. of the International Conference on Cloud and Autonomic Computing (ICCAC'15)*, 2015, pp. 208–211.

[22] B. Asgari, M. Arani, and S. Jabbehdari, "An efficient approach for resource autoscaling in Cloud environments," *International Journal of Electrical and Computer Engineering*, vol. 6, no. 5, pp. 2415–2424, 2016.

[23] X. Dutreilh, S. Kirgizov, O. Melekhova, J. Malenfant, N. Rivierre, and I. Truck, "Using Reinforcement Learning for Autonomic Resource Allocation in Clouds: towards a fully automated workflow," in *Proc. of the 7th International Conference on Autonomic and Autonomous Systems (ICAS'11)*, 2011, pp. 67–74.

[24] E. Barrett, E. Howley, and J. Duggan, "Applying reinforcement learning towards automating resource allocation and application scalability in the Cloud," *Concurrency and Computation: Practice and Experience*, vol. 25, no. 12, pp. 1656–1674, 2013.

[25] P. Jamshidi, C. Pahl, and N. C. Mendona, "Managing Uncertainty in Autonomic Cloud Elasticity Controllers," *IEEE Cloud Computing*, vol. 3, no. 3, pp. 50–60, 2016.

[26] Q. Zhu and G. Agrawal, "Resource Provisioning with Budget Constraints for Adaptive Applications in Cloud Environments," *IEEE Transactions on Services Computing*, vol. 5, no. 4, pp. 497–511, 2012.

[27] P. Padala, K. Hou, K. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, and A. Merchant, "Automated Control of Multiple Virtualized Resources," in *Proc. of the 4th ACM European Conference on Computer Systems (EuroSys'09)*. ACM, 2009, pp. 13–26.

[28] E. Kalyvianaki, T. Charalambous, and S. Hand, "Self-adaptive and Self-configured CPU Resource Provisioning for Virtualized Servers Using Kalman Filters," in *Proc. of the 6th International Conference on Autonomic Computing (ICAC'09)*. ACM, 2009, pp. 117–126.

[29] P. Bodík, R. Griffith, C. Sutton, A. Fox, M. Jordan, and D. Patterson, "Statistical Machine Learning Makes Automatic Control Practical for Internet Datacenters," in *Proc. of the Conference on Hot Topics in Cloud Computing (HotCloud'09)*. USENIX Association, 2009.

[30] Y. Liu, D. Gureya, A. Al-Shishtawy, and V. Vlassov, "OnlineElastMan: self-trained proactive elasticity manager for Cloud-based storage services," *Cluster Computing*, pp. 1977–1994, 2017.

[31] D. Gmach, J. Rolia, L. Cherkasova, and A. Kemper, "Capacity Management and Demand Prediction for Next Generation Data Centers," in *Proc. of the IEEE International Conference on Web Services (ICWS'07)*, 2007, pp. 43–50.

[32] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes, "CloudScale: Elastic Resource Scaling for Multi-tenant Cloud Systems," in *Proc. of the 2nd ACM Symposium on Cloud Computing (SOCC'11)*. ACM, 2011, pp. 5:1–5:14.

[33] G. Zhenhuan, G. Xiaohui, and J. Wilkes, "PRESS: PRedictive Elastic ReSource Scaling for Cloud systems," in *Proc. of the International Conference on Network and Service Management*, 2010, pp. 9–16.

[34] Y. Hu, B. Deng, and F. Peng, "Autoscaling prediction models for Cloud resource provisioning," in *Proc. of the 2nd IEEE International Conference on Computer and Communications (ICCC'16)*, 2016, pp. 1364–1369.

[35] R. S. Shariffdeen, D. T. S. P. Munasinghe, H. S. Bhathiya, U. K. J. U. Bandara, and H. M. N. D. Bandara, "Workload and Resource Aware Proactive Auto-scaler for PaaS Cloud," in *Proc. of the IEEE 9th International Conference on Cloud Computing (CLOUD'16)*, 2016, pp. 11–18.

[36] M. N. A. H. Khan, Y. Liu, H. Alipour, and S. Singh, "Modeling the Autoscaling Operations in Cloud with Time Series Data," in *Proc. of the IEEE 34th Symposium on Reliable Distributed Systems Workshop (SRDSW'15)*, 2015, pp. 7–12.

[37] Y. Hu, B. Deng, F. Peng, and D. Wang, "Workload prediction for Cloud computing elasticity mechanism," in *Proc. of the IEEE International Conference on Cloud Computing and Big Data Analysis (ICCCBDA'16)*, 2016, pp. 244–249.

[38] R. N. Calheiros, R. Rajiv, A. Beloglazov, C. A. F. D. Rose, and R. Buyya, "CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms," *Software - Practice & Experience*, vol. 41, no. 1, pp. 23–50, 2011.

[39] A. Núñez, J. L. Vázquez-Poletti, A. C. Caminero, G. G. Castañé, J. Carretero, and I. M. Llorente, "iCanCloud: A Flexible and Scalable Cloud Infrastructure Simulator," *Journal of Grid Computing*, vol. 10, no. 1, pp. 185–209, 2012.

[40] D. Kliazovich, P. Bouvry, and S. U. Khan, "GreenCloud: a packet-level simulator of energy-aware Cloud computing data centers," *The Journal of Supercomputing*, vol. 62, no. 3, pp. 1263–1283, 2012.

[41] S. H. Lim and B. Sharma and G. Nam and E. K. Kim and C. R. Das, "MDCSim: A multi-tier data center simulation, platform," in *Proc. of the IEEE International Conference on Cluster Computing and Workshops*, 2009, pp. 1–9.

[42] R. N. Calheiros, M. A. Netto, C. A. D. Rose, and R. Buyya, "EMUSIM: an integrated emulation and simulation environment for modeling, evaluation, and validation of performance of Cloud computing applications," *Software: Practice and Experience*, vol. 43, no. 5, pp. 595–612, 2013.

[43] B. Wickremasinghe, R. N. Calheiros, and R. Buyya, "CloudAnalyst: A CloudSim-Based Visual Modeller for Analysing Cloud Computing Environments and Applications," in *Proc. of the 24th IEEE International Conference on Advanced Information Networking and Applications*, 2010, pp. 446–452.

[44] M. C. Calzarossa, M. L. Della Vedova, L. Massari, D. Petcu, M. I. M. Tabash, and D. Tessera, "Workloads in the Clouds," in *Principles of Performance and Reliability Modeling and Evaluation*, ser. Springer Series in Reliability Engineering, L. Fiondella and A. Puliafito, Eds. Springer, 2016, pp. 525–550.

[45] J. Du, N. Sehrawat, and W. Zwaenepoel, "Performance Profiling of Virtual Machines," *ACM SIGPLAN Notices*, vol. 46, no. 7, pp. 3–14, 2011.

[46] G. Ren, E. Tune, T. Moseley, Y. Shi, S. Rus, and R. Hundt, "Google-wide Profiling: A Continuous Profiling Infrastructure for Data Centers," *IEEE Micro*, vol. 30, no. 4, pp. 65–79, 2010.

[47] K. Alhamazani, R. Ranjan, K. Mitra, F. Rabhi, P. Jayaraman, S. Khan, A. Guabtni, and V. Bhatnagar, "An overview of the commercial Cloud monitoring tools: research dimensions, design issues, and state-of-the-art," *Computing*, vol. 97, no. 4, pp. 357–377, 2015.

[48] F. Azmandian, M. Moffie, J. Dy, J. Aslam, and D. Kaeli, "Workload Characterization at the Virtualization Layer," in *Proc. of the 19th Int. Symp. on Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS'11)*. IEEE, 2011, pp. 63–72.

[49] R. Birke, L. Chen, and E. Smirni, "Multi-Resource Characterization and their (In)dependencies in Production Datacenters," in *Proc. of the Network Operations and Management Symposium (NOMS'14)*. IEEE, 2014.

[50] A. Do, J. Chen, C. Wang, Y. Lee, A. Zomaya, and B. B. Zhou, "Profiling applications for virtual machine placement in Clouds," in *Proc. of the 4th Int. Conf. on Cloud Computing (CLOUD'11)*. IEEE, 2011, pp. 660–667.

[51] R. Weingärtner, G. Bräscher, and C. Westphall, "Cloud resource management: A survey on forecasting and profiling models," *Journal of Network and Computer Applications*, vol. 47, pp. 99–106, 2015.

[52] J. Calero and J. G. Aguado, "Comparative analysis of architectures for monitoring Cloud computing infrastructures," *Future Generation Computer Systems*, vol. 47, pp. 16–30, 2015.

[53] K. Fatema, V. Emeakaroha, P. Healy, J. Morrison, and T. Lynn, "A survey of Cloud monitoring tools: Taxonomy, capabilities and objectives," *Journal of Parallel and Distributed Computing*, vol. 74, no. 10, pp. 2918–2933, 2014.

[54] J. Huang and D. Nicol, "Trust mechanisms for Cloud computing," *Journal of Cloud Computing*, vol. 2, no. 1, pp. 1–14, 2013.

[55] S. Meng and L. Liu, "Enhanced Monitoring-as-a-Service for Effective Cloud Management," *IEEE Transactions on Computers*, vol. 62, no. 9, pp. 1705–1720, 2013.

[56] J. Mueller, D. Palma, G. Landi, J. Soares, B. Parreira, T. Metsch, P. Gray, A. Georgiev, Y. Al-Hazmi, T. Magedanz, and P. Simoes, "Monitoring as a Service for Cloud Environments," in *Proc. of the 5th Int. Conf. on Communications and Electronics (ICCE'14)*. IEEE, 2014, pp. 174–179.

[57] T. Somasundaram and K. Govindarajan, "CLOUDRB: A framework for scheduling and managing High-Performance Computing (HPC) applications in science Cloud," *Future Generation Computer Systems*, vol. 34, pp. 47–65, 2014.

[58] R. Duan, R. Prodan, and X. Li, "Multi-Objective Game Theoretic Scheduling of Bag-of-Tasks Workflows on Hybrid Clouds," *IEEE Transactions on Cloud Computing*, vol. 2, no. 1, pp. 29–42, 2014.

[59] F. Zhang, J. Cao, K. Li, S. Khan, and K. Hwang, "Multi-objective scheduling of tasks in Cloud platforms," *Future Generation Computer Systems*, vol. 37, pp. 309–320, 2014.

[60] Z.-H. Zhan, X.-F. Liu, Y.-J. Gong, J. Zhang, H. S.-H. Chung, and Y. Li, "Cloud Computing Resource Scheduling and a Survey of Its Evolutionary Approaches," *ACM Computing Surveys*, vol. 47, no. 4, pp. 63:1–63:33, 2015.

[61] M. Mao and M. Humphrey, "Scaling and Scheduling to Maximize Application Performance within Budget Constraints in Cloud Workflows," in *Proc. of the 27th Int. Symp. on Parallel and Distributed Processing (IPDPS'13)*. IEEE, 2013, pp. 67–78.

[62] S. Dutta, S. Gera, A. Verma, and B. Viswanathan, "Smartscale: automatic application scaling in enterprise Clouds," in *Proc. of the 5th Int. Conf. on Cloud Computing (CLOUD'12)*. IEEE, 2012, pp. 221–228.

[63] C. Ardagna, E. Damiani, F. Frati, D. Rebeccani, and M. Ughetti, "Scalability Patterns for Platform-as-a-Service," in *Proc. of the 5th Int. Conf. on Cloud Computing - CLOUD'12*. IEEE, 2012, pp. 718–725.

[64] D. Cheng, C. Jiang, and X. Zhou, "Heterogeneity-Aware Workload Placement and Migration in Distributed Sustainable Datacenters," in *Proc. of the 28th Int. Symp. on Parallel and Distributed Processing (IPDP'14)*. IEEE, 2014, pp. 307–316.

[65] R. Singh, U. Sharma, E. Cecchet, and P. Shenoy, "Autonomic Mix-aware Provisioning for Non-stationary Data Center Workloads," in *Proc. of the 7th Int. Conf. on Autonomic Computing (ICAC'10)*. ACM, 2010, pp. 21–30.

[66] S. Spicuglia, M. Björkqvist, L. Chen, G. Serazzi, W. Binder, and E. Smirni, "On Load Balancing: A Mix-aware Algorithm for Heterogeneous Systems," in *Proc. of the 4th Int. Conf. on Performance Engineering - ICPE'13*. ACM, 2013, pp. 71–76.

[67] D. A. Menasce, "Scaling for e-business," in *Proc. of the 8th Int. Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2000, pp. 511–513.

[68] F. Duarte, B. Mattos, J. Almeida, and V. Almeida, "Hierarchal Characterization and Generation of Blogosphere Workloads," Boston University, Tech. Rep. BUCS-TR-2008-028, 2008.

[69] M. Shams, D. Krishnamurthy, and B. Far, "A Model-based Approach for Testing the Performance of Web Applications," in *Proc. of the 3rd Int. Workshop on Software Quality Assurance (SOQUA'06)*. ACM, 2006, pp. 54–61.

[70] F. Benevenuto, T. Rodrigues, M. Cha, and V. Almeida, "Characterizing User Behavior in Online Social Networks," in *Proc. of the 9th ACM SIGCOMM Conference on Internet Measurement (IMC'09)*. ACM, 2009, pp. 49–62.

[71] D. Feitelson, *Workload Modeling for Computer Systems Performance Evaluation*. Cambridge University Press, 2015.

[72] M. C. Calzarossa, L. Massari, and D. Tessera, "Workload Characterization: A Survey Revisited," *ACM Computing Surveys*, vol. 48, no. 3, pp. 1–43, 2016.

[73] D. Magalhães, R. N. Calheiros, R. Buyya, and D. G. Gomes, "Workload Modeling for Resource Usage Analysis and Simulation in Cloud Computing," *Comput. Electr. Eng.*, vol. 47, no. C, pp. 69–81, 2015.

[74] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data Clustering: A Review," *ACM Computing Surveys*, vol. 31, no. 3, pp. 264–323, 1999.

[75] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Angela, "An Efficient k-Means Clustering Algorithm: Analysis and Implementa-

tion," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 7, pp. 881–892, 2002.

[76] M. E. Tipping and C. M. Bishop, "Probabilistic principal component analysis," *Journal of the Royal Statistical Society, Series B*, vol. 61, no. 3, pp. 611–622, 1999.

[77] M. Belshe, R. Peon, and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)," https://www.rfc-editor.org/info/rfc7540, 2015, IETF.

[78] M. C. Calzarossa, L. Massari, M. I. M. Tabash, and D. Tessera, "Cloud autoscaling for HTTP/2 workloads," in *Proc. of the 3rd International Conference on Cloud Computing Technologies and Applications (CloudTech'17)*, 2017.