Service Level Provisioning in Cloud Systems: Models, Algorithms and Architectures

Luca Silvestri

June 2014
Doctorate
Computer Science, Control and Geoinformation
Cycle XXV

SERVICE LEVEL PROVISIONING IN CLOUD SYSTEMS: MODELS, ALGORITHMS AND ARCHITECTURES

Luca Silvestri

Advisor:
Dr. Emiliano Casalicchio

Rome, June 17, 2014
To my parents

To VP
Abstract

The cloud computing paradigm has emerged as an efficient and cost effective way for managing and delivering services over the Internet. Cloud computing allows customers to acquire resources in a very short time on a pay-per-use basis. This allows to minimize startup costs and to rapidly scale up or scale down resources obtaining an high degree of flexibility. Despite of the spread and growth of cloud-based solutions, the problem of autonomic service level provisioning in cloud systems is still an open issue. The simple autoscaling services currently offered by some IaaS providers are far away from giving a valid solution for an application service provider that wants to efficiently allocate resources minimizing costs and guaranteeing the desired level of performance in conditions of unpredictable and bursty traffic. Moreover, currently available service level agreements usually provide guarantees only on infrastructure availability, totally ignoring high level performance metrics such as average response time or throughput.

In this thesis architectures, models and algorithms to perform efficient QoS-aware resource allocation in clouds are presented. After the analysis of the services and features currently offered by the main IaaS providers, four different QoS-aware cloud architectures are introduced. The problem of optimal VM allocation is formulated as an optimization problem and solved using both proactive and reactive heuristic algorithms. Moreover, a prototype implementation of two of the proposed architectures is presented and evaluated. To properly stress the system during the experiments a stochastic workload model to generate realistic synthetic workloads has been presented and used.

Since monolithic cloud architectures are evolving in the direction of Inter-clouds of public and private resources, the architectures, models and algorithms proposed are extended to be effective also for a provider that decides to outsource resources to a cloud federation in order to obtain the maximum level of scalability and resource distribution.
First of all, I would like to thank my mother who supported me in all these years of study and in my whole life. I will never thank her enough for all the things she did for me. This thesis is dedicated to her and to my father. I hope he would have been happy about me.

A special thanks goes to my advisor, Emiliano Casalicchio. This thesis would not have been possible without his continuous guidance and support. Every time I needed help the door of his office was always open for me, even when he was overwhelmed with work (very often indeed!). His advices, revisions and moral support have been crucial for me to complete my graduate studies.

I would like to thank all the other Professors of the Computer Engineering research group at University of Rome Tor Vergata that helped me and gave me many precious advices during my Ph.D. studies: Valeria Cardellini, Vincenzo Grassi, Francesco Lo Presti and Salvatore Tucci. Thanks to Stefano Iannucci, Valerio Di Valerio and to all my colleagues and labmates with whom I shared many days of work and study. These days would, surely, have been much longer and boring without their company.

A sincere thanks goes to my family and to all my friends, that made my life less hard and helped me to overcome many difficulties and bad moments. A special mention goes to Claudio and Francesco that have been always very close to me, even if they live abroad.

Finally, I would like to thank a very special person who has been always on my mind during the writing of this thesis and made me rediscover beautiful things and feelings I thought lost. VP, whatever will happen in the future, every time I think about these months and this thesis I will think of you. And you can be sure that it will never, ever be a bad thought.

Rome, June 2014
CONTENTS

3 Service Level Provisioning in Clouds 29

3.1 Problem statement ............................................. 29

3.2 Service Level Provisioning Taxonomy .......................... 30

3.3 State of the art ................................................... 33

3.3.1 Where .......................................................... 34

3.3.2 When .......................................................... 38

3.3.3 What .......................................................... 42

3.3.4 Why .......................................................... 46

3.3.5 How .......................................................... 49

3.4 Thesis Contributions vs Taxonomy .............................. 53

4 Autonomic Architectures 57

4.1 Autonomic Architecture for VM Allocation ..................... 58

4.2 IaaS providers features taxonomy ............................... 59

4.2.1 Taxonomy .................................................... 60

4.2.2 Remarks .................................................... 65

4.3 QoS-aware architectures ......................................... 65

4.3.1 Extreme ASP control ....................................... 69

4.3.2 Full ASP control ............................................ 71

4.3.3 Partial ASP Control ......................................... 73

4.3.4 Limited ASP control ........................................ 74

4.4 Cloud Federation Scenario .................................... 75

4.4.1 Architecture of the inter-cloud resource manager ........ 76
## 5 Single Provider VM Allocation

5.1 Single Provider Optimal VM Allocation Model
   5.1.1 SLA Definition
   5.1.2 Problem Formulation

5.2 Heuristic VM Allocation

5.3 Workload Model

5.4 VM Allocation Experimental Evaluation
   5.4.1 Workload
   5.4.2 Performance Metrics
   5.4.3 Optimal VM Allocation
   5.4.4 Heuristic VM Allocation
   5.4.5 Heuristic versus Optimal VM Allocation

5.5 Prototype Implementation and Experimental Evaluation
   5.5.1 Partial ASP Control Implementation
   5.5.2 Limited ASP Control Autoscaling Policies
   5.5.3 Workload generation
   5.5.4 Testbed setup
   5.5.5 Performance and Responsiveness analysis

5.6 Annex: Recursive Least Square Based Prediction
   5.6.1 Autoregressive Process Prediction
   5.6.2 Recursive Least Square Process Prediction

## 6 Inter-Cloud VM Allocation

6.1 Cloud Federation Outsourcing Model
   6.1.1 Model
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Service Level Provisioning Taxonomy Dimensions.</td>
<td>33</td>
</tr>
<tr>
<td>3.2</td>
<td>Where Dimension</td>
<td>34</td>
</tr>
<tr>
<td>3.3</td>
<td>When Dimension</td>
<td>38</td>
</tr>
<tr>
<td>3.4</td>
<td>What Dimension</td>
<td>42</td>
</tr>
<tr>
<td>3.5</td>
<td>Why Dimension</td>
<td>46</td>
</tr>
<tr>
<td>3.6</td>
<td>How Dimension</td>
<td>49</td>
</tr>
<tr>
<td>3.7</td>
<td>Taxonomy dimensions addressed in this thesis</td>
<td>54</td>
</tr>
<tr>
<td>4.1</td>
<td>IaaS providers advanced features and services</td>
<td>66</td>
</tr>
<tr>
<td>5.1</td>
<td>Main notation adopted in the optimization problem formulation</td>
<td>87</td>
</tr>
<tr>
<td>5.2</td>
<td>Settings for the main system model parameters</td>
<td>96</td>
</tr>
<tr>
<td>5.3</td>
<td>System workload characteristics</td>
<td>97</td>
</tr>
<tr>
<td>5.4</td>
<td>Optimal allocation: sensitivity to $W$</td>
<td>101</td>
</tr>
<tr>
<td>5.5</td>
<td>$r$-1 heuristic: sensitivity to $\alpha$</td>
<td>101</td>
</tr>
<tr>
<td>5.6</td>
<td>Proactive allocation: sensitivity to $W$</td>
<td>102</td>
</tr>
<tr>
<td>5.7</td>
<td>Allocation cost for the smooth ramp workload</td>
<td>117</td>
</tr>
<tr>
<td>5.8</td>
<td>Allocation cost for the bursty workload (1-minute case)</td>
<td>119</td>
</tr>
<tr>
<td>5.9</td>
<td>Allocation cost for the bursty workload (5 minutes evaluation time)</td>
<td>119</td>
</tr>
<tr>
<td>6.1</td>
<td>Service Level Agreements</td>
<td>133</td>
</tr>
<tr>
<td>6.2</td>
<td>Service providers cost, availability and security level</td>
<td>134</td>
</tr>
</tbody>
</table>
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Cloud computing stack and its relationship with cloud computing actors</td>
<td>17</td>
</tr>
<tr>
<td>4.1</td>
<td>General architecture of the application service provider</td>
<td>59</td>
</tr>
<tr>
<td>4.2</td>
<td>Taxonomy of advanced features offered by IaaS providers</td>
<td>60</td>
</tr>
<tr>
<td>4.3</td>
<td>Autonomic QoS-aware general architecture for service level provisioning</td>
<td>67</td>
</tr>
<tr>
<td>4.4</td>
<td>Extreme ASP control architecture</td>
<td>70</td>
</tr>
<tr>
<td>4.5</td>
<td>Full ASP control architecture</td>
<td>72</td>
</tr>
<tr>
<td>4.6</td>
<td>Partial ASP Control architecture</td>
<td>73</td>
</tr>
<tr>
<td>4.7</td>
<td>Limited ASP Control architecture</td>
<td>74</td>
</tr>
<tr>
<td>4.8</td>
<td>Cloud federation scenario</td>
<td>76</td>
</tr>
<tr>
<td>4.9</td>
<td>General Architecture of the inter-cloud resource manager</td>
<td>77</td>
</tr>
<tr>
<td>5.1</td>
<td>Response Time</td>
<td>82</td>
</tr>
<tr>
<td>5.2</td>
<td>SLA Violations</td>
<td>83</td>
</tr>
<tr>
<td>5.3</td>
<td>Three days workload excerpt from the 1998 World Cup website</td>
<td>93</td>
</tr>
<tr>
<td>5.4</td>
<td>Workload obtained removing noise</td>
<td>94</td>
</tr>
<tr>
<td>5.5</td>
<td>Synthetic workload generated using a 10 states DTMC</td>
<td>94</td>
</tr>
<tr>
<td>5.6</td>
<td>Synthetic workload generated using a 50 states DTMC</td>
<td>95</td>
</tr>
<tr>
<td>5.7</td>
<td>Synthetic workload generated using a 100 states DTMC</td>
<td>95</td>
</tr>
<tr>
<td>5.8</td>
<td>The 20 workload instances used in the experiments</td>
<td>97</td>
</tr>
<tr>
<td>5.9</td>
<td>VM allocation for $K = 0$, $K = 5$, and $K = 10$</td>
<td>99</td>
</tr>
<tr>
<td>5.10</td>
<td>Fraction of SLO violations for $K = 0$, $K = 5$, and $K = 10$</td>
<td>100</td>
</tr>
<tr>
<td>5.11</td>
<td>Optimal VM allocation cost and average fraction of SLO violations for</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>different values of $K$</td>
<td></td>
</tr>
<tr>
<td>5.12</td>
<td>Comparison of heuristic allocation policies</td>
<td>103</td>
</tr>
</tbody>
</table>
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.13</td>
<td>Optimal versus heuristic VM allocation</td>
<td>104</td>
</tr>
<tr>
<td>5.14</td>
<td>Two examples of stress-test workload generated using MediaWiki and httperf</td>
<td>112</td>
</tr>
<tr>
<td>5.15</td>
<td>Architecture of the implemented testbed</td>
<td>112</td>
</tr>
<tr>
<td>5.16</td>
<td>The allocated VMs time series (left) and the Latency time series (right)</td>
<td>116</td>
</tr>
<tr>
<td>5.17</td>
<td>The allocated VMs time series and the Latency time series for the bursty workload</td>
<td>118</td>
</tr>
<tr>
<td>5.18</td>
<td>Empirical CDF of the latency for the bursty workload (1 minute evaluation period)</td>
<td>118</td>
</tr>
<tr>
<td>5.19</td>
<td>The allocated VMs and latency time series for the bursty workload (5 minutes evaluation period)</td>
<td>120</td>
</tr>
<tr>
<td>5.20</td>
<td>Empirical CDF of the Latency for the bursty workload (5 minutes evaluation period)</td>
<td>120</td>
</tr>
<tr>
<td>6.1</td>
<td>Cloud federation operation scenario</td>
<td>126</td>
</tr>
<tr>
<td>6.2</td>
<td>Optimization problem formulation</td>
<td>131</td>
</tr>
<tr>
<td>6.3</td>
<td>Traffic distribution among Zones for the different value of $X_{max}$</td>
<td>134</td>
</tr>
<tr>
<td>6.4</td>
<td>Total allocation cost $C$ sensitivity to $X_{max}$</td>
<td>136</td>
</tr>
<tr>
<td>6.5</td>
<td>Traffic distribution ($\lambda_j$) sensitivity to latency</td>
<td>137</td>
</tr>
<tr>
<td>6.6</td>
<td>Total allocation cost $C$ sensitivity to latency</td>
<td>137</td>
</tr>
<tr>
<td>6.7</td>
<td>Traffic distribution ($\lambda_j$) sensitivity to zone $Z_1$ arrival rate</td>
<td>138</td>
</tr>
<tr>
<td>6.8</td>
<td>Traffic distribution ($\lambda_j$) sensitivity to availability constraint</td>
<td>139</td>
</tr>
<tr>
<td>6.9</td>
<td>Traffic distribution ($\lambda_j$) sensitivity to availability constraint (weak constraint)</td>
<td>140</td>
</tr>
</tbody>
</table>
6.10 Total allocation cost C sensitivity to availability constraint . . . . . . . 141
6.11 Traffic distribution (\(\lambda_j\)) sensitivity to security constraint . . . . . . . 142
6.12 Total allocation cost C sensitivity to security constraints . . . . . . . 142
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASP</td>
<td>Application Service Provider</td>
</tr>
<tr>
<td>AWS</td>
<td>Amazon Web Services</td>
</tr>
<tr>
<td>B2B</td>
<td>Business to Business</td>
</tr>
<tr>
<td>B2C</td>
<td>Business to Consumer</td>
</tr>
<tr>
<td>DTMC</td>
<td>Discrete Time Markov Chain</td>
</tr>
<tr>
<td>EC2</td>
<td>Elastic Compute Cloud</td>
</tr>
<tr>
<td>HI</td>
<td>Historical Intelligence</td>
</tr>
<tr>
<td>IaaS</td>
<td>Infrastructure as a Service</td>
</tr>
<tr>
<td>LC</td>
<td>Least Connection</td>
</tr>
<tr>
<td>MAPE-K</td>
<td>Monitor, Analyze, Plan, Execute with Knowledge</td>
</tr>
<tr>
<td>MP</td>
<td>Markov Process</td>
</tr>
<tr>
<td>NIST</td>
<td>National Institute of Standards and Technology</td>
</tr>
<tr>
<td>PaaS</td>
<td>Platform as a Service</td>
</tr>
<tr>
<td>QN</td>
<td>Queueing Network</td>
</tr>
<tr>
<td>QoS</td>
<td>Quality of Service</td>
</tr>
<tr>
<td>RLS</td>
<td>Recursive Least Square</td>
</tr>
<tr>
<td>RND</td>
<td>Random</td>
</tr>
<tr>
<td>RR</td>
<td>Round Robin</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>SaaS</td>
<td>Software as a Service</td>
</tr>
<tr>
<td>SLA</td>
<td>Service Level Agreement</td>
</tr>
<tr>
<td>SLO</td>
<td>Service Level Objective</td>
</tr>
<tr>
<td>SP</td>
<td>Service Provider</td>
</tr>
<tr>
<td>VM</td>
<td>Virtual Machine</td>
</tr>
<tr>
<td>WLC</td>
<td>Weighted Least Connection</td>
</tr>
<tr>
<td>WRR</td>
<td>Weighted Round Robin</td>
</tr>
</tbody>
</table>
1

Introduction

Contents

1.1 Motivation ......................................................... 1
1.2 Research Challenges ............................................ 4
1.3 Thesis Contributions ............................................. 9
1.4 Thesis Outline .................................................... 10

1.1 Motivation

The cloud computing paradigm [1,2] has emerged as an efficient and cost effective way for managing and delivering services over the Internet. Cloud computing allows customers to acquire resources (i.e., computing power, storage, memory) in a very short time on a pay-per-use basis obtaining a high degree of elasticity. This allows to minimize startup costs and to rapidly scale up or down resources avoiding performance degradation in case of peak load and over-provisioning in case of scarce demand.

Among all the challenges that cloud computing poses, we focus our attention on the dynamic QoS provisioning problem. QoS provisioning is not a new issue in networked and distributed systems; however the cloud and service computing paradigms increase the system complexity and scale, therefore posing new challenges.

Cloud computing is architected in a stack composed by three main service models
Chapter 1. Introduction

or abstract layers: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). Each layer offers to the upper layer its resources as a set of services, in the spirit of earlier distributed computing and network architectures. According to the layers of the cloud service, the dynamic QoS provisioning problem can be managed at infrastructure and platform levels (e.g., [3-6]), or at service level (e.g., [7-11]). At infrastructure and platform level the provisioning problem is related to the mapping of virtual machines (VMs) on physical resources and of application containers on VMs to satisfy Service Level Agreements (SLAs) with the infrastructure or platform users minimizing costs. In this thesis we focus on the service level provisioning problem, that is related to mapping applications to applications containers or application directly to VMs in order to satisfy SLAs agreed with the customers of the cloud-based applications and to maximize revenues of the service provider offering the application (Application Service Provider, ASP).

Despite of the success of cloud computing and of the spread and growth of cloud-based solutions, the problem of service level provisioning in cloud systems is still an open issue. Some IaaS providers (e.g., Amazon Web Services [12], Rackspace [13]) offer simple autoscaling services that are still far away from allow to an application service provider or a Software as a Service provider to efficiently allocate resources minimizing costs and guaranteeing the desired level of performance in conditions of unpredictable and bursty traffic. Moreover, currently available service level agreements usually provide guarantees only on infrastructure availability totally ignoring other high level performance metrics such as average response time or throughput [14].

Moreover, monolithic cloud architectures are evolving in the direction of Inter-clouds [15-18]. An Inter-Cloud is a ”cloud of clouds” that allows on-demand reassign-
1.1. Motivation

ment of resources and transfer of workload through a interworking of public and private cloud systems of different cloud providers based on coordination of each consumer requirements for service quality with each providers SLA and use of standard interfaces. A Cloud Federation is a particular type of Inter-Cloud where cloud providers voluntarily collaborate interconnecting their infrastructures in order to allow resource sharing among each others.

Resource sharing in a cloud federation allows a provider not only to maximize revenues and datacenter utilization, but also to scale performance, availability and security requirements outsourcing part of the incoming workload to other providers inside the federation, instead of rejecting service requests when unable to fulfill the SLAs agreed with customers. In this scenario, focusing on the service level provisioning problem and considering the opportunity of resource outsourcing in the federation, it is crucial to determine: (1) how many resources (e.g., VMs) to allocate; (2) where to allocate these resources inside the federation; (3) how to distribute the incoming load among the available resources.

Therefore there is a need for autonomic solutions that allow providers to guarantee application level SLAs to user (e.g., maximum response time) minimizing the resource usage when environmental conditions and workload level are extremely variable.

In this thesis architectures, models and algorithms to perform efficient QoS-aware resource allocation in clouds are presented. After addressing the problem of service level provisioning in clouds for an ASP using resources from a single IaaS provider, the architectures, models and algorithms proposed are extended to be effective also for a provider that, to obtain the maximum level of scalability and resource distribution, has the opportunity to outsource resources to a cloud federation.
Chapter 1. Introduction

1.2 Research Challenges

To deal with the problem of service level provisioning in clouds in an effective way different research issues have to be addressed.

- **Service Level Agreements**
  
  To perform service level provisioning the definition and formalization of service level agreements, contracts that define the minimal guarantees in terms of service level that a provider offers to its customers, is crucial [1]. Although SLAs can be used to specify the minimum level of availability, performance, serviceability, security and other requirements, the current available SLAs from IaaS providers consider only availability constraints [14]. For the purpose of service level provisioning SLAs that contemplate performance requirements such as the maximum guaranteed response time or the minimum throughput are needed. Moreover while, at the moment, SLA violation detection is left to providers or users, a third-party trusted authority able to detect violations and to resolve possible disputes between service providers and users would be very helpful.

Another open issue for SLAs is the lack of standardization. A structured representation of SLAs would be useful to allow customers to compare the guarantees offered by different providers and is essential to adopt automatic validation, management and monitoring of SLAs [19–22]. Although many different languages have been proposed to define SLAs, (e.g., WSLA [23], SLAng [24], WS-Agreement [25], CSLA [26]), none of them is currently recognized as a standard.

More open issues emerge in the case of a cloud federation, where different and
possibly conflicting SLA definitions from different providers should be considered and managed.

- **Cloud Monitoring**

Advanced and effective monitoring is required to perform service level provisioning in clouds. Monitoring of cloud systems is crucial both for service and infrastructure providers and for cloud customers. Providers need it to continuously control and manage the hardware and the software infrastructures and to evaluate performance at platform and application level. Monitoring is also needed both by providers and customers to detect, prevent and recover SLA violations and to measure the offered services, allowing users to be billed on a pay-per-use basis.

It is clear that cloud monitoring is an essential and not trivial task that includes many challenging issues [27][30]. First of all cloud monitoring should be performed at different levels (IaaS, PaaS, SaaS). The choice of the system layers where to put sensors and probes and of the metrics to evaluate is strictly dependent from the purpose for which monitoring is needed (e.g., data center management, SLA management, billing, performance management, resource management, security management). The monitoring of high level metrics (e.g., response time) in highly distributed systems such as clouds requires to take account for factors such as the network latency between system components and users possibly dispersed in different geographical zones. Moreover, in cross-domain environments such as multi-clouds, federated clouds and hybrid clouds other extremely challenging issues due to the heterogeneity of systems, tools and to the lack of widely accepted standard arise. The problem of cross domain monitoring
is object of ongoing research [31] and has not been properly addressed yet.

Despite many open source platforms (e.g., Amazon CloudWatch [32], Aneka [33], commercial platforms (e.g., OpenNebula [34], Nimbus [35]), and services (e.g., CloudSleuth [36], CloudHarmony [37]) are available to perform cloud monitoring, for the purpose of efficient autonomic service level provisioning the utilization of multiple tools, possibly extended by ASP personalized sensors and probes may be needed.

- Autonomic Architectures

To perform autonomic resource provisioning effective autonomic architectures are needed. A possible solution, presented in [38] and utilized in many works in literature (e.g., [39–41]) is to organize cloud systems according to the autonomic loop MAPE-K [42] (Monitor, Analyze, Plan, Execute, and Knowledge). In such a way, cloud-based applications should be able to automatically react to changing components, workload, and environmental conditions minimizing operating costs and preventing SLA violations. Other possible architectural approaches to achieve the same goal are based on control theory feedback loop (e.g., [43–45]) or on distributed architectures that perform multi-level (local and global) adaptation [9].

Furthermore, autonomic architectures may operate at multiple system or application layers. In case of multi layer applications, interactions between the various layers and data replication and consistency issues have to be taken into consideration. An example of an architecture that operates at different levels is the two-level resource manager for hosting service platform proposed in [9]. The manager allows to determine: 1) the number of VM to allocate for each appli-
1.2. Research Challenges

cation (VM provisioning); 2) the VM to physical machine mapping (VM packing). Other examples are the autonomic two-level resource management system with local controllers at the virtual-container level and a global controller at the resource-pool level proposed in [46] and the autonomic layered cloud architecture, where, for each layer (SaaS, PaaS, IaaS) there are different goals, sensors and actuators and a different feedback adaptive loop has to be implemented illustrated in [10].

Moreover, in the case of a cloud federation, autonomic architectures have to manage resources and traffic from different providers and resolve interoperability issues between the federation providers.

- **Cloud Benchmarking**

To evaluate and compare different service level provisioning approaches in clouds benchmarking tools are needed. However benchmarking cloud services is still an open issue and no standard or largely accepted solutions are available. For the purpose of evaluating the elasticity, responsiveness and performance of an autonomic system for service level provisioning in clouds a benchmark capable to reproduce the behavior of a typical Web 2.0 application would be useful.

Cloudstone [47], at the best of our knowledge, has been the first Web 2.0 benchmark designed expressly for cloud systems. Another cloud specific specific benchmark is CloudCmp [48], a comparator of the performance and cost of cloud providers that measures the elastic computing, persistent storage, and networking services offered by IaaS providers along metrics that directly reflect their impact on the performance of customer applications. Other cloud specific benchmarks are Cloudbench [49], an open-source framework that automates cloud-scale eval-
Chapter 1. Introduction

Evaluation and benchmarking through the running of controlled experiments, where complex applications are automatically deployed, and Malstone [50], a benchmark specifically designed to measure the performance of cloud computing middleware that supports data intensive computing. In [51] is introduced the idea of building custom benchmark applications, able to measure indexes strictly related to cloud user needs. The proposed solution is based on the mOSAIC framework [52], which offers a deployable platform and an API for building provider-independent applications.

Due to the lack of standard benchmarking tools, often other non-cloud specific web benchmarks such as Wikibench [53, 54] and TPC-W [55] are utilized to test solutions for autonomic provisioning. Moreover, the lack of publicly available recent workload traces from real-world web applications makes difficult to stress the autonomic systems with realistic traffic. Therefore, to properly stress the cloud systems under evaluation, there is a need for distributed load generators [56].

- **Workload and Performance Forecasting**

To prevent SLA violations avoiding resource over-provisioning effective proactive algorithms capable to anticipate possible traffic bursts, component failures and environmental changes are required. Although many interactive web applications and services are usually subject to workloads that present periodic characteristics, the short-term prediction of traffic levels is still a very hard task due to the high level of burstiness that characterizes web traffic.

Many techniques for time series analysis and forecasting [57, 58] such as autoregressive methods, multivariate regression methods and machine learning ap-
1.3 Thesis Contributions

Proaches can be used to perform short term prediction [59]. Since the effectiveness of a particular technique is strictly dependent from the nature and the characteristics of the phenomenon that has to be predicted (e.g., system failures and critical event prediction [60–62], threshold violations [62–64], workload level [65]), the choice of the best prediction model and method is usually very tricky and, in some particular cases, is almost impossible to perform accurate prediction. Moreover, to be the prediction effective, in many cases the parameters of the prediction method employed should be continuously updated using new observations of the values of the time series under analysis.

1.3 Thesis Contributions

To address the issues presented in the previous section in a way to perform efficient service level provisioning in clouds this thesis gives the following original contributions:

- A taxonomy of the state of the art solutions for service level provisioning in clouds is presented.

- Currently available IaaS providers features and services are described and classified.

- Four different QoS-aware cloud architectures and an inter-cloud architecture are introduced.

- An optimization problem formulation is given for the problems of optimal VM allocation using resource from a single IaaS provider and for resource outsourcing in a cloud federation.
Chapter 1. Introduction

- Different heuristic algorithms to perform VM allocation are proposed and compared to autoscaling policies.
- A stochastic workload model to generate synthetic workloads that reproduce the time dependency and bursty characteristics of real workloads is presented.
- The implementation of a prototype of two of the proposed architectures in the Amazon EC2 public cloud is presented and evaluated through simulation experiments.

1.4 Thesis Outline

This thesis is organized as follows.

In Chapter 2, the foundations of cloud computing are introduced. The different service models and deployment models are illustrated. An overview on Federated Clouds and on Service Level Agreements in clouds is also given.

In Chapter 3, the problem of service level provisioning in clouds is, first, formally stated and then examined in detail through the analysis of the state of the art in literature. To better understand the various problems and solutions for service level provisioning in clouds, the approaches in literature are organized in a service level provisioning taxonomy.

In Chapter 4, a general autonomic QoS-Aware resource provisioning architecture is presented. Then, after an analysis of the features of the currently available IaaS providers, four different implementation of the architecture, differing for the degree of usage of IaaS providers features and services, are introduced. An inter-cloud resource manager architecture that extends the presented architecture to allow interoperation
between different cloud providers in a cloud federation scenario is also presented.

In Chapter 5 the problem of optimal VM allocation for an ASP using resources from a single IaaS provider is addressed. After the definition of a SLA considering performance metrics, the system model for the autonomic VM allocation and a formulation as an optimization problem are presented. Reactive and proactive heuristic policies to solve the problem are proposed and evaluated. Moreover, the implementation and evaluation of a prototype of two of the proposed architecture is presented. A workload model used to generate realistic synthetic workloads is also introduced in this chapter.

In Chapter 6 the problem of resource outsourcing in a cloud federation is modeled. A model that considers users from different service classes and network latency is introduced. From this model and a SLA that contemplate factors such as performance, security and availability an optimization problem formulation is proposed. Finally, the evaluation of the proposed model is described.

Concluding remarks and future research are discussed in Chapter 7.
In this chapter the basic concepts in the field of cloud computing will be introduced and defined. After the description of the cloud computing foundations, the three service model and the four deployment models will be presented. Afterwards the concepts of Inter-Cloud and of SLAs will be examined and the problem of resource provisioning in clouds will be introduced.

Contents

2.1 Cloud Computing ................................. 14
  2.1.1 Cloud Computing Definition .......................... 14
  2.1.2 Cloud Computing Actors ............................ 16
2.2 Service Models ................................. 17
  2.2.1 Iaas .............................................. 18
  2.2.2 Paas .............................................. 18
  2.2.3 SaaS .............................................. 19
2.3 Deployment Models ............................. 19
  2.3.1 Private Cloud ..................................... 19
  2.3.2 Community Cloud .................................. 19
  2.3.3 Public Cloud ..................................... 20
  2.3.4 Hybrid Cloud .................................... 20
2.4 Inter-Cloud and Cloud Federation ............... 20
2.5 SLAs in Clouds .................................... 21
2.6 Resource Provisioning in Clouds .................. 23
  2.6.1 Infrastructure and platform provisioning ............ 24
  2.6.2 Service Level Provisioning .......................... 25
Chapter 2. Cloud Computing Foundations

2.1 Cloud Computing

Cloud Computing has emerged as a paradigm in information technology that lets cloud service clients deploy their applications in a large scale environment with an expectation of high scalability, availability, fault tolerance and reduced administration costs. Cloud Computing enables open market service providers and enterprises to outsource computational and storage utilities with the opportunity to scale up in case of peak load, to rely on high availability and to drastically reduce the start up and management cost of data centers. Moreover, the ”elastic” property of Cloud Computing, if properly exploited, avoids over-provisioning of resources in case of scarce demand, that, along with the resource sharing at infrastructure level, contributes to energy saving.

2.1.1 Cloud Computing Definition

Today, Cloud Computing is often used as a general term to indicate any solution that allows the outsourcing of any kind of hosting and computing resources. In the last years there have been many different definitions of Cloud Computing [1][2][66].

The definition generally accepted is the one proposed by the National Institute of Standards and Technology (NIST) [67]:

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

A detailed description of the five essential characteristics is given in the NIST def-
2.1. Cloud Computing

- **On-demand self-service** "A consumer can unilaterally provision computing capabilities, such as server time and network storage, as needed automatically without requiring human interaction with each service provider."

- **Broad network access** "Capabilities are available over the network and accessed through standard mechanisms that promote use by heterogeneous thick or thin client platforms (e.g., mobile phones, tablets, laptops and workstations)."

- **Resource pooling** "The providers computing resources are pooled to serve multiple consumers using a multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand. There is a sense of location independence in that the customer generally has no control or knowledge over the exact location of the provided resources but may be able to specify location at a higher level of abstraction (e.g., country, state, or datacenter). Examples of resources include storage, processing, memory, and network bandwidth."

- **Rapid elasticity** "Capabilities can be elastically provisioned and released, in some cases automatically, to scale rapidly outward and inward commensurate with demand. To the consumer, the capabilities available for provisioning often appear to be unlimited and can be appropriated in any quantity at any time."

- **Measured Service** "Cloud systems automatically control and optimize resource use by leveraging a metering capability (typically this is done on a pay-per-use or charge-per-use basis) at some level of abstraction appropriate to the type
Chapter 2. Cloud Computing Foundations

of service (e.g., storage, processing, bandwidth, and active user accounts). Resource usage can be monitored, controlled, and reported, providing transparency for both the provider and consumer of the utilized service.”

The service models (i.e., Software as a Service - SaaS; Platform as a Service - PaaS; Infrastructure as a Service - IaaS) and the deployment models (i.e., private cloud, community cloud, public cloud, hybrid cloud) will be described in detail in sections 2.2 and 2.3 respectively.

An alternative definition, more focused on the perception of the cloud as a service-oriented, dynamically configurable and SLA customizable system, is the one provided in [66]:

"Clouds are a large pool of easily usable and accessible virtualized resources (such as hardware, development platforms and/or services). These resources can be dynamically reconfigured to adjust to a variable load (scale), allowing also for an optimum resource utilization. This pool of resources is typically exploited by a pay-per-use model in which guarantees are offered by the Infrastructure Provider by means of customizable SLAs”

2.1.2 Cloud Computing Actors

No one of the definitions above reported explicitly mentions who are the cloud computing actors and how cloud computing is architected. The actors involved in the cloud are essentially three: the service provider, the service user, and the infrastructure provider. The service provider makes services accessible to end users through an Internet-based interface. To this end, the service provider uses resources (virtualized and software) that are furnished and managed by the infrastructure provider, thus to reduce costs and
2.2 Service Models

Cloud computing is architected in a stack composed by three main service models or abstract layers (Figure 2.1): the infrastructure layer, the platform layer, and the application layer. Each layer offers to the upper layer its resources as a set of services, in the spirit of earlier distributed computing and network architectures. In this new computing paradigm, that allows to access both virtualized and software resources which are commonly referred as Anything-as-a-Service (XaaS), the stack layers are classified on the basis of the service they provide.
2.2.1 Iaas

Infrastructure-as-a-Service (IaaS) is defined in [67] as "The capability provided to the consumer to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications."

IaaS virtualizes the hardware layer and offers computing services such as storage, CPU, memory, and network. The CPU, memory, and local storage are packaged as virtual machines of different sizes, each with a price per hour (or KB in case of memory/storage). Examples of IaaS are Amazon EC2, Google Compute Platform and Rackspace. An exhaustive listing of the most important IaaS providers and of the services they offer is given in section 4.2.

2.2.2 Paas

Platform-as-a-Service (PaaS) is "The capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages, libraries, services, and tools supported by the provider." [67]

PaaS offers platform services, such as Web, application, and database servers, and a programming environment associated with it. Programmers can use this environment to develop, test, and deploy their applications. PaaS leases/uses IaaS services by requesting from them virtual machines and storage and deploying application containers on the virtual machines. There can be many platforms running on the same cloud infrastructure. Example of PaaS are Google AppEngine, Force.com and Oracle Cloud Platform.
2.3. Deployment Models

2.2.3 SaaS

Software-as-a-Service (SaaS) is defined by NIST [67] as “The capability provided to the consumer to use the provider's applications running on a cloud infrastructure.” The SaaS layer consists of simple or composite services (e.g., SOA applications) offered to the end users. Those applications are deployed in PaaS containers on architectures specific to each application. Usually, many applications share the same PaaS. In general, a SaaS user pays a subscription. Examples of SaaS are Gmail, Google Apps and Microsoft Office 365.

2.3 Deployment Models

Cloud infrastructures are classified in four deployment models that differ for the type of users that have access to the cloud resources and for the ownership of the infrastructure.

2.3.1 Private Cloud

In private clouds the infrastructure is provisioned for exclusive use by a single organization comprising multiple consumers. It may be owned, managed, and operated by the organization, a third party, or some combination of them, and it may exist on or off premises. [67] An example of a private cloud is a cloud infrastructure of a private company that offers services accessible only by the company’s employees.

2.3.2 Community Cloud

In community clouds the infrastructure is provisioned for exclusive use by a specific community of consumers that have shared concerns (e.g., mission, security requirements, policy, and compliance considerations). It may be owned, managed, operated
Chapter 2. Cloud Computing Foundations

by one or more of the organizations in the community, a third party, or some combination of them, and it may exist on or off premises. An example of a community cloud is an infrastructure owned and managed by one or more academic institutions and accessible by all the users that work on a specific research project.

2.3.3 Public Cloud

In public clouds the infrastructure is provisioned for open use by the general public. It may be owned, managed, and operated by a business, academic, or government organization, or some combination of them. It exists on the premises of the cloud provider. Examples of public clouds providers are Amazon Web Services, Rackspace and Google Cloud.

2.3.4 Hybrid Cloud

Hybrid clouds are infrastructures composed by two or more distinct cloud infrastructures (private, community, or public) that remain unique entities, but are bound together by standardized or proprietary technology that enables data and application portability (e.g., cloud bursting for load balancing between clouds). A typical example of a hybrid cloud is the one of an organization that has its private cloud infrastructure and, in case of traffic bursts that overcome the private cloud capacity, acquires resources from a public cloud to guarantee the desired level of services to users.

2.4 Inter-Cloud and Cloud Federation

The term “Inter-Cloud” has been used in literature to define a cloud of clouds. Inter-cloud allow to distribute load among different cloud data centers. A
formal definition of Inter-Cloud is given in [72]:

"A cloud model that, for the purpose of guaranteeing service quality, such as the performance and availability of each service, allows on-demand reassignment of resources and transfer of workload through a interworking of cloud systems of different cloud providers based on coordination of each consumers requirements for service quality with each providers SLA and use of standard interfaces."

According to the classification given in [15] two different type of Inter-Clouds can be identified: Cloud Federation and Multi-cloud. The difference is that, while in a Cloud Federation two or more cloud providers collaborate to interconnect their infrastructures in a way that allows resource sharing among each others, in Multi-cloud there is no volunteer collaboration between the Inter-Cloud providers and the usage of multiple providers is responsibility of clients or external services such as brokers.

In the rest of this thesis we will focus our attention on Cloud Federation, assuming that there is an explicit and voluntary collaboration between the federation members.

Cloud Federation can be further classified in Centralized and Peer-to-Peer. In centralized federations a central entity is responsible for resource allocation and traffic distribution. In peer federations providers communicate and negotiate directly with each others. Examples of centralized federations are the InterCloud system proposed in [16] and the Contrail project [73]; a peer federation model is used in the Reservoir project [17]. In this thesis we focused on the case of centralized federations.

### 2.5 SLAs in Clouds

Service Level Agreements (SLAs) are contracts that define in a formal way the minimal guarantees in terms of service level that a provider offers to its customers.
The SLA may specify the levels of availability, serviceability, performance, operation, or other attributes of the service, such as billing. For each one of these functional and not functional QoS requirements a Service Level Objective (SLO) is defined, that is a threshold on the value assumed by the QoS metrics/attributes (e.g., the minimum availability averaged over a given period, the average response time or a stricter requirement such as the 90-percentile of the response time or the tail of the response time distribution). Non-compliance to the agreement may incur in penalties to the service providers. Typically, the SLA contains also an insurance clause offering monetary compensation to the user if the provider fails to provide the service objectives according to the minimum levels specified in the SLA (e.g., Amazon EC2 or Google App Engine SLA policies). The service contracts in the cloud environment typically differ from traditional SLA contracts for providing a more flexible service accounting scheme, typically referred to as pay-per-use or pay-as-you-go.

As stated in [14], a typical SLA of a cloud provider has the following components:

- **Service guarantee**: specifies the SLO that a provider has to meet over a service guarantee time period. Examples are $\text{Availability} \geq 0.99\%$; $\text{Throughput} \geq 100\text{requests/s}$; $\text{AverageResponseTime} \leq 0.5\text{ms}$.

- **Service guarantee time period**: describe the duration over which a service guarantee should be met (e.g., one month, one day, one hour).

- **Service guarantee granularity**: describe the resource scale on which a provider specifies a service guarantee (e.g., per service, per datacenter, per instance, per transaction).

- **Service guarantee exclusions**: specifies the instances that are excluded from
service guarantee metric calculation (e.g., downtime period caused by scheduled maintenance).

- **Service Credit**: determines the amount credited to the customers (i.e. the penalty that the provider has to pay) if the service guarantee is not met.

- **Service violation measurement and reporting**: describes how and who measures and reports the violation of service guarantee.

Analyzing the SLAs currently offered from cloud public cloud providers, different works [14, 74] show that none of the considered IaaS providers offer any performance-based SLA. All SLAs consider only minimum availability requirements (from 99.9% to 100% in a monthly or yearly basis). Moreover, most of the cloud providers leave to the customers the burden of providing evidence for SLA violations.

## 2.6 Resource Provisioning in Clouds

The three cloud computing layers are different also from a quality of service perspective (i.e., performance, dependability, and security). For example, the access to virtualized and software resources results in different dependability and security requirements; a B2B or B2C enterprise user has typically performance, dependability, and security requirements more stringent than the SLO of a consumer (e.g., a researcher); moreover, the revenue/economic model and business goal of the above mentioned users and the cloud computing layers are dissimilar [10]. This heterogeneity originates diverse and often contrasting challenges that must be addressed at the different layers. Here we focus on the cloud aspects related to the dynamic QoS provisioning which can be classified according to the layers in the cloud stack:
Chapter 2. Cloud Computing Foundations

- The infrastructure and platform provisioning, that is related to the mapping of virtual machines (VMs) on physical resources and of application containers on VMs to satisfy SLAs with the infrastructure/platform users and to maximize revenues for the infrastructure and platform providers;

- The service provisioning, that is related to the mapping of applications to application containers or applications directly to VMs, to satisfy SLAs agreed with the users of the cloud-based applications and to maximize revenues of the service providers offering the applications.

2.6.1 Infrastructure and platform provisioning

The infrastructure providers, among all the infrastructure management aspects, deal with the problems related to the mapping of virtual machines on physical resources and of application containers on VMs. At infrastructure and platform levels, the resource management is driven by: (i) the SLA exposed by IaaS, for example expressed as the minimum availability level guaranteed by the provider to the user; and (ii) the data center policy related to governance costs (e.g., maximization of resource usage and minimization of power consumption).

With regard to the SLA definition, we observe that most cloud providers define SLAs for the services they offer. However, as stated before, the non-functional attributes on which guarantees are typically provided focus mostly on availability, while one of the most important attributes for the users of the cloud-based service, i.e., the response time, is typically not addressed in the current SLAs. For example, the Amazon EC2 SLA states that AWS will use commercially reasonable efforts to make Amazon EC2 available with an Annual Uptime Percentage of at least 99.95% during the
Service Year. The motivation for the lack of the response time guarantee is simple: response times depend on many factors that may be not under the control of the service provider (e.g., the network path to the cloud platform), and the traffic addressed to the cloud-based service may be highly unpredictable (for example, Web-based services are characterized by flash crowds). Therefore, the enforcement of a particular level of the response time is not a trivial task for the cloud provider and requires the ability to dynamically manage the resources assignment to the cloud-based applications in an elastic way in order to face highly variable traffic conditions. For example, cloud providers can allocate additional computational resources on the fly to handle increased demand and deallocate them when the traffic directed to the application decreases. The main strategies adopted to guarantee SLA in infrastructure and platform provisioning can be classified as:

- **Virtual machine migration**, that provides significant benefits to balance load across the data center and to enable robust and highly responsive provisioning in data centers;

- **Server consolidation**, that is a practice to maximize resource utilization while minimizing energy consumption;

- **Geographical distribution of data centers**, along with optimal sizing and placement policies, allows to reduce significantly energy, infrastructures, and connectivity costs.

### 2.6.2 Service Level Provisioning

At application layer, the resource management problem can be formulate as follow: "To find, in case of unpredictable and suddenly changing workload conditions, the set
of VMs that should be allocated, or more in general, the set of services that should be selected to guarantee the SLA fulfillment and to minimize the management cost” [75].

The resources/services selection is driven by business-level SLA of the hosted applications, and therefore it is driven by the service user needs. It is in charge of the service provider to enforce the SLA agreed with its customers. The SaaS provider should size (by means of allocation or selection) the IaaS/PaaS resources used to provide the cloud-based application in order to satisfy the service level objectives of the service users, to scale up the application load, or to meet its own performance, dependability, and/or security requirements while minimizing its operating costs.

There is a variety of mechanisms that a service provider can adopt to guarantee a certain level of QoS, including, besides allocation and selection, admission control and performance isolation. However, when dealing with large scale services subject to suddenly changing workload conditions, all the solutions are based on the concept of autonomic computing [76]. The architecture of an autonomic system comprises a set of managed resources and managers. Each manager communicates with the resources through a sensor/actuator mechanism and the decision is elaborated using the so called MAPE (Monitor, Analyze, Plan, Execute) reference model from the IBM’s autonomic computing initiative [42]. This autonomic loop collects information from the system, makes decisions and then organizes the adaptation actions needed to achieve goals and objectives, and controls the execution. To achieve a high degree of flexibility in allocating the computing resources on one hand and to comply to users requirements specified in the SLAs on the other, also cloud systems should be organized according to the autonomic loop [38]. In such a way, cloud-based applications could be able to automatically respond to changing components, workload, and environmental con-
2.6. Resource Provisioning in Clouds

ditions while minimizing the operating costs and preventing violations of the agreed SLAs.

The problem of Service Level Provisioning in clouds will be examined in detail in chapter 3.
Service Level Provisioning in Clouds

In this chapter the problem of service level provisioning in clouds will be examined and explored. A taxonomy has been developed to classify the related scientific works into five dimension and, for every identified dimension, a detailed report of the state-of-the-art solutions is provided.

Contents

3.1 Problem statement ........................................... 29
3.2 Service Level Provisioning Taxonomy ....................... 30
3.3 State of the art ................................................ 33
  3.3.1 Where .................................................. 34
  3.3.2 When .................................................... 38
  3.3.3 What .................................................... 42
  3.3.4 Why ...................................................... 46
  3.3.5 How ...................................................... 49
3.4 Thesis Contributions vs Taxonomy .......................... 53

3.1 Problem statement

The problem of service level provisioning in clouds, defined in 2.6.2, consists essentially in determining the minimum quantity of resources that a service provider has to use to guarantee stipulated SLAs in a highly dynamic environment.
The main challenge is to determine in advance the needed resources since the incoming workload is extremely variable and difficult to predict. The service provider has to find a good trade-off between the risk of overestimating and underestimating the incoming traffic. Indeed, while in the first case there is an economic loss due to resource waste, in the second case the economic damage for the provider is determined by the penalty that has to be paid to customers if SLAs are violated. To guarantee the desired level of QoS, besides allocating/deallocating VMs, a provider may adopt different mechanisms such as admission control, performance isolation and traffic redirection.

To perform online service level provisioning in a dynamic environment subject to highly changing workload conditions, autonomic solutions are required. In the next section a taxonomy of the works that perform autonomic resource provisioning in cloud will be presented.

### 3.2 Service Level Provisioning Taxonomy

Different solutions have been proposed in literature to address the problem of autonomic resource provisioning in clouds. Besides all these works perform online adaptation through autonomic techniques, they differ for many aspects (e.g., the autonomic technique used, the autonomic architecture proposed, the adaptation goal, the way in which adaptation is performed). To classify this works, similarly to what has been done in [77] for self-adaptive software, we can start from these five questions, each one related to a dimension of the problem under examination:

- **Where** does provisioning take place?
3.2. Service Level Provisioning Taxonomy

- **When** is provisioning needed?

- **What** is provisioned?

- **Why** is provisioning needed?

- **How** is provisioning performed?

The *where* dimension can be considered from different viewpoints. At a high level this dimension consider both the adaptation model used for the adaptive system architecture (e.g., based on MAPE-K cycle, on feedback loop from control theory or on local and global adaptation) and the architectural components (e.g., Provisioning Manager, VM Allocator, Load Balancer, Federation Controller, ...). At a lower level the *where* dimension considers the system layers where the adaptation takes place (e.g., presentation layer, application layer, backend) and the fact that the adaptation involves one or more system tiers. From the cloud perspective this dimension considers the service level of the cloud stack where the adaptation is executed (i.e. IaaS, PaaS or SaaS service model).

The *when* question includes all the temporal aspects of the adaptation process. Depending on the system characteristics and on the adaptation goals it is important to determine when the adaptation is feasible and how often it is needed. Various approaches can be taken. Considering the time interval, adaptation can be event driven (started by some particular event or trigger) or periodic. In case of periodic adaptation the time scale of the interval between adaptation actions can be in the order of seconds and minutes (i.e., short period), hours (i.e., medium period) or variable. A mixed case in which adaptation is triggered by events but only at discrete time intervals can also be considered. Finally, related to the *when* dimension is important to determine if the
adaptation must be performed when something goes wrong (reactive adaptation) or in advance, preventing possible unwanted behaviors (proactive adaptation).

The what question refers to the system resources, components and policies that can be changed through adaptation actions. In this dimension it is important to determine what can be changed, what needs to be changed and what is the feasible range for the adaptable system parameters. In a cloud system there are essentially three types of resources: computing power, storage and bandwidth. Moreover, it is important to determine the metrics that have to be monitored (e.g., CPU utilization, response time, system throughput, ) and what are the events and thresholds that trigger adaptation. This dimension includes also the choice of the granularity of the resources that the service level provisioning considers (e.g., physical resource, virtual resource, single VM, VMs in a zone, VMs in a datacenter, VMs in a private cloud, . . . ) and the type of workload that the adaptive system has to bear (i.e., interactive or batch).

In the why dimension the adaptation objectives of the self-adaptive system are considered. Some of the possible objectives are costs minimization or revenues maximization guaranteeing Service Level Agreements, energy consumption minimization, performance or throughput maximization. If adaptation problem is formulated as an optimization problem, these goals can be formally defined in an objective function.

The how dimension includes all the aspects related to the means used to perform adaptation. First of all it is important to determine the adaptation action to take (i.e., resize, migrate or reallocate VMs, redistribute traffic, change service level, . . . ). In the how dimension have to be considered also the adaptation policies utilized and the physical and logical components that are the adaptation actuators. Moreover, this dimension comprises also the method used to evaluate the adaptive system (e.g., analytical evalu-
### 3.3 State of the art

In this section, for every taxonomy dimension, the attributes of interests are identified and their possible values are listed. A summary of the attributes and values is given in table [3.1]. State of the art solutions are then examined for every attribute of the taxonomy dimensions.

#### Table 3.1: Service Level Provisioning Taxonomy Dimensions.

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Attributes</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>WHERE?</td>
<td>Architecture Type</td>
<td>Single-Tier, Multi-Tier</td>
</tr>
<tr>
<td></td>
<td>Service Model</td>
<td>IaaS, SaaS, PaaS</td>
</tr>
<tr>
<td></td>
<td>Adaptation Model</td>
<td>MAPE-K cycle, feedback loop, local and global adaptation</td>
</tr>
<tr>
<td>WHEN?</td>
<td>Adaptation Type</td>
<td>Proactive, Reactive</td>
</tr>
<tr>
<td></td>
<td>Adaptation Period</td>
<td>Periodic, Event-driven, Mixed</td>
</tr>
<tr>
<td></td>
<td>Adaptation Interval</td>
<td>short, medium, variable</td>
</tr>
<tr>
<td>WHAT?</td>
<td>Resource Type</td>
<td>Computing, Storage, Bandwidth</td>
</tr>
<tr>
<td></td>
<td>Granularity</td>
<td>VM Resource, VM, Zone, Data Center, Private Cloud</td>
</tr>
<tr>
<td></td>
<td>Workload Type</td>
<td>Batch, Interactive</td>
</tr>
<tr>
<td></td>
<td>Metrics</td>
<td>CPU Utilization, Response Time, Throughput, Arrival Rate, Memory Utilization, Energy Consumption, ...</td>
</tr>
<tr>
<td>WHY?</td>
<td>Adaptation Goal</td>
<td>Maximize Performance, Minimize Costs, Maximize Revenue</td>
</tr>
<tr>
<td>HOW?</td>
<td>Action to take</td>
<td>Allocate/Deallocate VMs, Resize/Migrate VMs, Reroute Traffic, ...</td>
</tr>
<tr>
<td></td>
<td>Adaptation Policy</td>
<td>Optimal Solution, Heuristics</td>
</tr>
<tr>
<td></td>
<td>Evaluation</td>
<td>Prototype, Simulation, Analytical</td>
</tr>
</tbody>
</table>
3.3.1 Where

Architecture Type

For what concerns the architecture type, most of the works in literature \([7, 11, 40, 41, 43, 44, 78-82]\) consider a single-tier architecture.

Only few of the considered works \([9, 45, 83]\) perform a multi-tier adaptation. From this works, in \([83]\) a dynamic provisioning technique for multi-tier Internet applications is presented. Different applications may run simultaneously and provisioning is performed for each tier of each application to guarantee SLAs reducing the resource usage of a private datacenter. Multi-tier applications are modeled as a queueing network where each queue represent a tier and queues from a tier feed into the next tier. VMs at each tier are modeled as \(G/G/1\) systems.

In \([45]\) the authors propose an adaptive resource control system that dynamically adjusts the resource shares to individual tiers in order to meet application-level QoS goals while achieving high resource utilization for multi-tier enterprise applications in a shared hosting environment.
3.3. State of the art

**Service Model**

All the works considered in this survey perform adaptation at Infrastructure level (IaaS). While most of this works consider that resources are from a public IaaS provider, in [79, 80] the adaptation is limited to resources belonging to a single private data-center.

**Adaptation Model**

Different adaptation models are used to perform adaptation. Many of the solution in literature perform adaptation implementing the MAPE-K autonomic cycle [42]. This autonomic cycle consists of four phases: Monitor, Analysis, Planning, Execution. All the phases are performed on the basis of a shared knowledge (K).

From solutions that employ a MAPE-K based architecture, [40] proposes an opportunistic service provisioning policy that leverages the variability of VM performance and the IaaS providers billing policies to minimize service provisioning costs maintaining a target VM utilization. The scenario considered is the one of a service provider that offers various type of atomic services. For each service there is a specific load balancer that distributes traffic to the VMs with fewer outstanding requests and a VM controller that monitors arrival rates, VM performance and keeps track of billing periods of active VMs. A VM broker collects information from controllers and performs dynamic resource provisioning.

In [11] is proposed an Integer Programming formulation of the problem of scaling up and scaling down VM instances by considering two aspects of a cloud application: performance and budget. The proposed auto-scaling mechanism includes four components: a Performance Monitor, a History Repository, an Auto Scaling Decider and a
VM Manager.

In our previous research [81, 82] we proposed both reactive and proactive policies and mechanisms to perform service level provisioning in clouds. All the proposed solutions are based on a MAPE-K architecture. In detail, the Workload Monitor and Performance Monitor components are responsible for the Monitoring Phase. The SLA Analyzer and Provisioning Manager components perform, respectively, the Analysis and Planning phases. Execution is demanded to the Load Balancer and VM Allocator.

Architectures based on a control theory feedback loop are the ones proposed in [43–45].

The Autoflex architecture [43] is based on a standard control feedback loop and is composed by three main components: a monitor that gathers information about VMs utilization, a controller that, relying on monitored and predicted utilization values, decides to start/stop VMs and a selector module that tries to find out the most appropriate prediction for a particular layer. The selector uses data on the predictors’ errors to reduce the probability of under provisioning and minimize SLO violations. In [44] an autoscaling approach in which both cloud and application dynamics are modeled in the context of stochastic, model predictive control problem is proposed. The model accounts for the delay in resource provisioning times, for the stochastic nature of real workloads and for billing constraints (i.e., does not release resources until their lease expiration).

In [45] two controllers are used: an utilization controller that controls the resource allocation for a single application tier, and an arbiter controller that controls the resource allocations across multiple application tiers and multiple application stacks.
sharing the same infrastructure. The controller algorithms were designed based on input-output models inferred from empirical data using a black-box approach.

Local and Global adaptation is performed in [9] where an autonomic resource manager for hosting service platform with a two level architecture is proposed. An application specific Local Decision Module computes a utility function which gives a measure of application satisfaction with specific resource allocation given current workload and SLA goals. A Global Decision Module is responsible to compute a global utility function from the local application specific utility functions and system level performance metrics (i.e., CPU utilization) from virtual and physical servers. First a VM Provisioning Module explores the space of all possible CPU allocation and find the solution with the best global utility; Then a VM Packing Module places VMs on physical machines for the solution while minimizing the number of active physical machines and the required migrations.

In [79] different heuristics for dynamic reallocation of VMs according to current resources requirements, while ensuring reliable QoS, are proposed. The reallocation is executed employing a distributed architecture composed by a dispatcher, local managers and global managers. The local managers (one per physical node) collect VMs performance information and issue commands for VM allocation, deallocation and resizing and for the startup/shutdown of physical machines. Global managers control a set of physical nodes, receives data from local managers and, after computing an allocation decision using an heuristic for semi-online multi dimensional bin-packing, issue commands for VM migration.
### 3.3.2 When

#### Adaptation Type

Many of the considered works use proactive techniques to perform adaptation. In [78] different auto-scaling policies to maximize the profit of a SaaS provider that offers a service to customers using resources provided by an IaaS provider are proposed and evaluated. Arrival rate prediction has been done employing a modified version of the Holt-Winters’ algorithm and two heuristics have been proposed and evaluated.

In [39] both the cloud capacity planning and the instant VM provisioning problem are formulated as time series prediction problems and are solved through a domain-specific self-training prediction mechanism. A cost-sensitive model to quantify cost caused by wrong predictions is introduced. To predict the future demands of single VM types an ensemble prediction models that employs five different time series prediction techniques and combine their prediction results to obtain a lower prediction error is used. The model is self-training: from historical data it adapts its parameters to minimize the prediction error.

---

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Values</th>
<th>Works</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adaptation Type</td>
<td>Proactive</td>
<td>[39, 41, 44, 78, 80]</td>
</tr>
<tr>
<td></td>
<td>Reactive</td>
<td>[9, 11, 45, 79, 82]</td>
</tr>
<tr>
<td></td>
<td>Proactive + Reactive</td>
<td>[43, 81, 83]</td>
</tr>
<tr>
<td>Adaptation Period</td>
<td>Periodic</td>
<td>[9, 11, 39, 41, 44, 45, 78, 83]</td>
</tr>
<tr>
<td></td>
<td>Event-driven</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Mixed</td>
<td>[43]</td>
</tr>
<tr>
<td>Adaptation Interval</td>
<td>short (i.e., 1-5 minutes)</td>
<td>[11, 40, 43, 80, 82]</td>
</tr>
<tr>
<td></td>
<td>medium (i.e., 1 hour)</td>
<td>[7, 41, 78]</td>
</tr>
<tr>
<td></td>
<td>variable</td>
<td>[9, 44, 83]</td>
</tr>
</tbody>
</table>
3.3. State of the art

In [40] the invocation rate of atomic services offered by a service provider is predicted using last value prediction and, to simplify the model, average performance of identical VMs are supposed to fluctuate in a discrete range of values.

In [44] autoscaling was formulated as a stochastic model predictive control problem, in which both cloud and application dynamics are modelled.

In [41] an optimal VM-level autoscaling scheme for web application providers with cost-latency trade-off is proposed. The web request distribution in each time unit (of one hour) is predicted by exploiting machine learning techniques and time series analysis. The prediction model is based on linear regression.

In [80] a system, called PRESS, for predictive elastic resource scaling in cloud systems has been implemented. To avoid SLA violations and minimize resource consumption PRESS continuously tracks the dynamic resource requirements of applications and predicts resource demand using two complementary techniques. A signature-driven resource demand prediction is performed for workloads with repeating patterns. A signature obtained using signal processing techniques is used to predict the future resource usage. In the case of applications without repeating patterns, PRESS uses a discrete-time Markov chain with a finite number of states to perform a short term prediction for the considered metric values. When the model make three consecutive mis-predictions, both prediction models are updated using new monitored values. Moreover, to minimize the risk of under-provisioning, PRESS increase the predicted value by an amount from 5% to 10%.

Only reactive adaptation is performed in [9, 11, 45, 79, 82]. In detail, the auto-scaling mechanism proposed in [11] computes the optimal solution on the basis of the current
workload level and performance requirements. No proactive action is performed.

In [82] a reactive policy, named Reactive 1 step early (r-1), that estimates the average arrival rate in the next time interval as the same observed in the last interval increased by a factor $a$ is proposed and compared with four different threshold based reactive autoscaling policies.

From works that perform both proactive and reactive adaptation Autoflex [43] is an auto-scaling framework that uses a hybrid approach to perform IaaS resource deployment. The framework is defined "service agnostic" because it only needs to monitor the resource utilization of the infrastructure where the services run and does not require specific information about the services offered. Proactive adaptation is performed using different predictors (based on auto-correlation, linear regression, auto regression, auto regression with integrated moving average and on the value observed in the last window): for each service and each layer the system dynamically selects the most appropriate predictor relying on historical information about the use that the service layers made of the infrastructure. Different layers may require different predictors and a single layer may require different predictors over time. Reactive adaptation is performed when VMs utilization is above a predefined threshold.

In [83] proactive and reactive methods are used at different time scales to determine when to provision resources. Proactive provisioning is used on a daily and hourly time scale to estimate the tail of the arrival rate distribution. Reactive provisioning is invoked every few minutes checking for deviations from predicted workload rates to allocate additional resources in a short time being capable to handle traffic bursts.

In [81] we propose both reactive and proactive heuristic policies for VM allocation.
3.3. State of the art

Besides the reactive r-l policy described above, a proactive policy, named proactive l step ahead, Y% (p1-Y) is introduced. p1-Y uses an autoregressive process process of order 2 AR(2), whose weights and white noise process variance are continuously estimated using the Recursive Least Square (RLS) method [84]. The p1-Y policy predicts the one step ahead Y% upper bound of the request arrival rate using the RLS-based prediction.

Adaptation Period

In almost all the works considered in this survey the adaptation is performed periodically at fixed time intervals. No work uses a completely event-driven approach; the only work that uses a mixed approach is [43], that performs proactive adaptation in a periodic way and triggers reactive adaptation when VMs utilization value is higher than a predefined threshold.

Adaptation Interval

In works that perform periodic adaptation different values for the adaptation interval are used. While some works [7, 41, 78] consider a medium adaptation interval of one hour (the same length of the usually adopted billing period), others considers shorter adaptation intervals of 5 minutes [11, 40, 43, 81, 82] or 1 minute [80, 82]. In some works [9, 44, 83] the adaptation interval is not fixed and can be varied according to the user and application requirements.
### Table 3.4: What Dimension

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Values</th>
<th>Works</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Granularity</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Resource</td>
<td></td>
<td>[45]</td>
</tr>
<tr>
<td>Homogeneous VMs</td>
<td>[39, 41, 43, 78, 80, 83]</td>
<td></td>
</tr>
<tr>
<td>VMs of different sizes</td>
<td>[7, 9, 11, 40, 44, 79]</td>
<td></td>
</tr>
<tr>
<td>Physical Machines</td>
<td></td>
<td>[85]</td>
</tr>
<tr>
<td><strong>Workload Type</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interactive</td>
<td>[9, 41, 44, 45, 78, 80, 83]</td>
<td></td>
</tr>
<tr>
<td>Batch</td>
<td>[7, 11]</td>
<td></td>
</tr>
<tr>
<td>Mixed</td>
<td>[39, 40, 43, 79]</td>
<td></td>
</tr>
<tr>
<td><strong>Metrics</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPU Utilization</td>
<td>[9, 43, 45, 79, 82, 85]</td>
<td></td>
</tr>
<tr>
<td>Response Time</td>
<td>[11, 39, 41, 44, 81, 83]</td>
<td></td>
</tr>
<tr>
<td>Throughput</td>
<td>[40, 78]</td>
<td></td>
</tr>
<tr>
<td>Arrival Rate</td>
<td>[7, 40, 78]</td>
<td></td>
</tr>
<tr>
<td>Memory Utilization</td>
<td></td>
<td>[9, 79]</td>
</tr>
</tbody>
</table>

#### 3.3.3 What

**Granularity**

Although all the works in the survey use computing power as the type of resource to adapt, they consider it at different granularity. Many works [39, 41, 43, 78, 80–83] uses homogeneous VMs as the atomic resource to adapt. In this case the resource provisioning problem becomes a VM allocation problem with VMs with identical characteristics.

In [78], since homogeneous performance VMs are considered, a fixed minimum guaranteed capacity is used to hide the non trivial performance difference experienced between virtual servers of the same type.

A basic unit of VM resource, named *VM Unit* is used in [39] to quantify the amount of resources needed in a time slot. The use of VM unit allows, defining the real VM as a multiple of the VM unit, to consider homogeneous resources although the VMs may be of different type and performance.
3.3. State of the art

Other works, still considering the VM as the atomic resource unit, take in account of VMs with different characteristics and computing power.

In [44] 11 different types of VM images matching Amazon EC2 instances have been simulated.

In [9] different type of VM are considered to determine the number of VM to allocate for each application subject to performance business-level SLAs (VM provisioning) and the VM to physical machine mapping subject to resource management costs (VM packing).

In [40] each service type runs on a variable number (pool) of VMs of different capacities and performance and each VM can run only one type of service. In the model is assumed that average performance of VMs with the same specification can fluctuate in time (due, for example, to server consolidation carried out by the IaaS provider).

In [7] Amazon EC2 spot instances of different types are used to execute compute-intensive, parallel and divisible batch jobs. The optimal instance type and bid price are computed checking for all relevant combinations of the considered parameters (i.e., spot instances prices, workload intensity, execution time and budget constraints).

In the model proposed in [11] three types of VMs (High CPU, High I/O and General) are taken into consideration. The types of VMs correspond to the three classes of jobs introduced in the model (computing intensive, I/O intensive and mixed).

Physical nodes are the adaptation resource unit in [79], where an infrastructure composed by a fixed number of heterogeneous physical nodes, each one characterized
by its own amount of CPU power (expressed in MIPS), RAM and network bandwidth is considered. Users submit requests for provisioning heterogeneous VMs; SLA violations happen when a VM cannot get the requested amount of resources.

Physical machines are the atomic resource unit also in [85] where is proposed an analytical performance model, with two interactive job classes, with the aim to determine the minimum number of physical machines needed to satisfy SLAs.

Adaptation at the level of single resources is performed in [45] where, to meet application-level QoS goals while achieving high resource utilization, an adaptive resource control system that dynamically adjusts the resource shares to individual tiers is proposed.

**Workload Type**

From works that use an interactive workload to stress the adaptive systems proposed, in [78], to compare the heuristics to the optimal policy and to standard threshold based policies, a Wikipedia replica has been deployed on Amazon EC2 and traffic from real Wikipedia traces has been replicated.

Three real world web log datasets running on Amazon EC2 have been used to experimentally evaluate the approach proposed in [41].

A synthetic workload based on FIFA ’98 World Cup workload has been used to stress the system in [83] where the proposed dynamic provisioning technique has been tested on a forty machine Xen/Linux based hosting platform running two different 3-tier web applications: an e-Bay-like auction site and a bulletin-board application.

An excerpt from FIFA ’98 World Cup workload has been used to stress the system also in [44] while a synthetic workload modeled from the WC ’98 trace has been
3.3. State of the art

generated in [81].

A sample of the real Wikipedia access trace has been used to generate a realistic synthetic workload in [82].

In [80] a prototype of the system has been implemented using the RUBiS benchmark [86] stressed with synthetic workload derived from interactive application traces.

RUBiS among with TPC-W [55] is one of the two-tier applications used to evaluate the approach proposed also in [45].

To test the optimal allocation of VMs a simulated environment that considers two online applications with different SLAs has been used in [9].

Batch workload has been considered to stress the systems in [7,11]. Batch workload from real applications has been used in the first work. In the second a batch workload of non dependent job has been considered. The jobs have been divided into three classes: computing intensive, I/O intensive or mixed. All the job classes have a one-hour response time deadline.

Some works use workloads of mixed type (from both interactive and batch applications)

In [40] the proposed adaptive policy has been evaluated through simulation using production traces from IBM.

The approach proposed in [39] has been validated through simulation experiments using real traces from various application running on IBM Smart Cloud Enterprise. To obtain better prediction performance data from traces are aggregated in daily time series.
Chapter 3. Service Level Provisioning in Clouds

Metrics

To drive the adaptation, in the works in literature different metrics are used: performance metrics (CPU Utilization, Response Time, Throughput, Memory Utilization), VM allocation and usage costs and the service request arrival rate.

An original metric that allows to quantify cost caused by wrong predictions in the model is introduced in [39]. This asymmetric cost measure, called *Cloud Prediction Cost* (CPC) allows to differentiate the costs of over-prediction (resource wasting) and under-prediction (SLA violation). Since CPC is a generic measure, it can been used for both capacity planning and instant VM provisioning scenarios.

To perform energy efficient resource management for virtualized cloud data centers also physical node temperature is considered in [79].

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Values</th>
<th>Works</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adaptation Goal</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Maximize Performance</td>
<td>[9, 11, 39, 41, 43, 44]</td>
</tr>
<tr>
<td></td>
<td>Minimize Costs</td>
<td>[7, 11, 39, 41, 80, 82]</td>
</tr>
<tr>
<td></td>
<td>Maximize Revenues</td>
<td>[78]</td>
</tr>
</tbody>
</table>

3.3.4 Why

Adaptation Goal

Different are the goals use to drive adaptation in the considered works.

The adaptation goal in [43] is to minimize SLOs violations given a target system utilization.

A similar goal is the one of the opportunistic service provisioning policy proposed in [40]: minimize service provisioning costs maintaining a target VM utilization.
The goal of the autoscaling policies proposed in [78] is to maximize the profit of a SaaS provider that offers a service to customers using resources provided by an IaaS provider. The SaaS provider profit is determined hourly by the revenue generated by every completed job per the number of completed jobs minus the cost paid for running the VMs that execute the jobs. A limitation of this model is that, currently, no explicit SLA is considered: incoming jobs that exceeds system capacity are rejected without penalty and no minimum service availability is required.

In [44], to find a good trade off between the conflicting goals of response time and cost minimization, the autoscaling problem is formulated as an optimal control problem in which the system behavior is represented by a single cost function that accounts both for the actual cost of leased resources and for the cost of deviation from desired performance objectives (i.e., cost of SLA violations).

To tackle the conflicting objectives of minimizing costs and reduce latency an optimization model with an objective function that considers the trade-off between cost and latency is defined in [41].

In [9] the authors formulate both VM provisioning (i.e., number of VM to allocate for each application subject to performance business-level SLAs) and VM packing (i.e., the VM to physical machine mapping subject to resource management costs) mapping problems as two constraints satisfaction problems with the goal to maximize a global utility function.

The goal of the probabilistic model proposed in [7] is to optimize monetary costs, performance and reliability given user and application requirements and dynamic conditions.

To consider both performance and budget aspects of cloud applications in [11] two
complementary goals are proposed: to minimize cost with performance constraints or to maximize computing power with budget constraints.

The goal of the provisioning technique for multi-tier applications presented in [83] is to guarantee SLAs reducing the resource usage of a private datacenter. Since resources are not leased from public IaaS providers, explicit costs of VMs are not considered.

In [79] different factors such as performance (CPU, RAM, bandwidth), network topologies and thermal optimization are considered to reach the goal of energy efficient resource management in a virtualized datacenter.

The goal of the PRESS system [80] is to minimize resource consumption avoiding SLA violations.

In [85] the goal of the proposed performance model is to determine the minimum number of resources (i.e. servers) needed to satisfy SLAs based on the percentile of the response time probability distribution.

The adaptive resource control system proposed in [45] aims to dynamically adjusts the resource shares to individual tiers in order to meet application-level QoS goals while achieving high resource utilization for multi-tier enterprise applications in a shared hosting environment.

In our previous research works [81] [82] the adaptation goal is to minimize VM allocation costs avoiding SLA violations.
3.3. State of the art

Table 3.6: How Dimension

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Values</th>
<th>Works</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action to take</td>
<td>Allocate/ Deallocate VMs</td>
<td>[9, 11, 39–41, 43, 44, 78–80, 82, 83]</td>
</tr>
<tr>
<td></td>
<td>Resize/ Migrate VMs</td>
<td>[9, 40, 79, 83]</td>
</tr>
<tr>
<td></td>
<td>Reroute Traffic</td>
<td>-</td>
</tr>
<tr>
<td>Adaptation policy</td>
<td>Optimal Solution</td>
<td>[9, 11, 41]</td>
</tr>
<tr>
<td></td>
<td>Heuristics</td>
<td>[40, 43, 44, 78–83]</td>
</tr>
<tr>
<td>Evaluation</td>
<td>Analytical</td>
<td>[85]</td>
</tr>
<tr>
<td></td>
<td>Simulation</td>
<td>[7, 9, 11, 40, 43, 44, 79, 81]</td>
</tr>
<tr>
<td></td>
<td>Prototype</td>
<td>[11, 39, 41, 45, 78, 80, 82, 83]</td>
</tr>
</tbody>
</table>

3.3.5 How

Action to take

Although most of the considered works [9, 11, 39–41, 43, 44, 78–80, 83] consider the allocation/deallocation of VMs as the only adaptive action to take, some works contemplate different adaptation actions.

In [40], since in the proposed model is assumed that average performance of VMs with the same specification can fluctuate in time (due, for example, to server consolidation carried out by the IaaS provider) and that a lower number of faster VMs may have the same performance of a higher number of slower VMs at a lower cost, two particular type of control action are considered beside turning on and shutting down VMs: VM replacement and VM reconfiguration. The former allows to replace a VM at the end of the billing period if its performance are below the average trying to obtain a better performing VM. The latter allows to reconfigure a previously allocated VM to run a different type of service. Since all considered actions involve a not negligible overhead, the broker has to evaluate if the estimated benefit of replacing or reconfiguring VMs is
greater than the adaptation actions cost.

The VM placement optimization performed in [79] is conducted in three stages: reallocation according to multiple system resources (CPU, RAM, bandwidth) utilization; optimization of virtual network topologies established between intercommunicating VMs; reallocation considering the thermal state of the resources.

**Adaptation Policy**

Since the optimal solution for the adaptation problem, depending on the model proposed, is usually to difficult to compute, different heuristics are proposed in the works in literature.

The optimization problem proposed in [44] has been solved (using the convex optimization solver `cvx`) for virtual compute units, and then a heuristic has been used to choose the best choice of VM instance configuration needed to obtain the requested computational power. Experimental results show that the proposed heuristic mapping function pushes the system towards using homogeneous instance types, penalizing small instances (i.e., using a limited number of medium-large instances is more convenient than using an high number of small instances).

In [78] homogeneous performance VMs are considered and, to estimate the system throughput, a G/GI/n/n queuing network model is employed. The model takes account of the non negligible VM setup and release time and of the fact that the IaaS provider bills customers per server-hour regardless of whether VMs are allocated for an entire hour or less. Two heuristics have been proposed and evaluated to perform adaptation.

In [83] an analytical model based on queuing networks is used to determine how much resources to allocate at each tier of the applications and a combination of predictive and reactive methods is used to determine when to provision these resources.
SLAs consider the percentile of the response time.

A few works valuate analytically the proposed adaptation policies.

In [41], assuming that the number of VMs that a provider can purchase is limited, an exhaustive search is conducted to find the optimal solution.

In the analytical performance model proposed in [85] two different allocation strategies and scheduling disciplined to determine the minimum number of resources needed to satisfy SLAs are proposed and evaluated analytically.

The cloud auto-scaling mechanism proposed in [11] computes the optimal solution on the basis of the current workload level and performance requirements not performing any proactive action.

In [81], to solve the optimization problem formulated, that has comprise an objective function that allows to consider the trade-off between allocation cost and user perceived performance, both proactive and reactive heuristic policies are proposed. The IaaS cloud, composed by a load balancer and a set of homogeneous VMs, is modeled as a network of M/M/1 queues. The same reactive policies among with other threshold based heuristic policies, have been used to perform VM allocation also in [82].

**Evaluation**

Simulation has been used in various works to evaluate the proposed adaptation solutions.

Autoflex [43] has been evaluated using trace-driven simulations. In the simulations scaling decisions are proactively taken every 5 minutes and VMs are shut down only when the billing period of one hour has elapsed.

The policy proposed in [40] has been evaluated through simulation using production traces from IBM and considering a control window of 5 minutes and a billing
Chapter 3. Service Level Provisioning in Clouds

period of 1 hour.

The approach proposed in [44] has been evaluated simulating an application provider that purchases VMs from an IaaS provider to build a web server cluster to handle transactional workloads.

To test the optimal allocation of VMs a simulated environment that considers two online applications with different SLAs has been used in [9].

The probabilistic model to optimize monetary costs, performance and reliability given user and application requirements and dynamic conditions proposed in [7] has been evaluated by simulation driven by real price of EC2 spot instances and batch workload from real applications.

The CloudSim toolkit [87] has been used to evaluate the resource management policy proposed in [79].

In [81] a stochastic workload model reproducing the time dependency and bursty characteristics of real workloads has been used to evaluate the propose VM allocation policies through simulation.

Other works implement a system prototype to evaluate the approach proposed.

A prototype developed on Amazon EC2 using three real word web log datasets has been used to experimentally evaluated the approach proposed in [41].

The auto-scaling mechanism proposed in [11] has been evaluated using both simulation and a real scientific application on a prototype. Although the prototype has been implemented in Windows Azure, it can be adapted to other clouds since the VM Manager hides the cloud provider interface and can be replaced with different cloud adapters.
The dynamic provisioning technique proposed in [83] has been tested on a forty machine Xen/Linux based hosting platform running two different 3-tier web applications stressed with a synthetic workload based on real traces.

A prototype of the PRESS system [80] has been implemented using the RUBiS benchmark [86] stressed with synthetic workload derived from interactive application traces.

In [78], to compare the proposed heuristics to the optimal policy and to standard threshold based policies, a prototype consisting of a Wikipedia replica has been deployed on Amazon EC2 and traffic from real Wikipedia traces has been replicated.

In [45], a testbed using Xen VMs has been built to evaluate the approach proposed using RUBiS [86] and TPC-W [55].

In [82], we implemented a testbed based on the MediaWiki application running on Amazon EC2 and stressed with a realistic workload based on Wikipedia access traces.

### 3.4 Thesis Contributions vs Taxonomy

The topics in the taxonomy addressed in this thesis are summarized in table 3.7.

Both for the single provider VM and for the Inter-Cloud VM allocation problems we consider a single-tier architecture based on the MAPE-K autonomic cycle that operates at IaaS level (see chapter 4 for details).

We consider a periodic adaptation with a short time adaptation interval varying from 1 to 5 minutes. Both reactive and proactive allocation policies are used to perform service level provisioning (detail on the adopted policies are given in section 5.2).

For what concerns the What dimension, we consider computing resources at the granularity of a single VM. In all the conducted experiments an interactive workload
Chapter 3. Service Level Provisioning in Clouds

Table 3.7: Taxonomy dimensions addressed in this thesis

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Attributes</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>WHERE?</td>
<td>Architecture Type</td>
<td>Single-Tier</td>
</tr>
<tr>
<td></td>
<td>Service Model</td>
<td>IaaS</td>
</tr>
<tr>
<td></td>
<td>Adaptation Model</td>
<td>MAPE-K cycle</td>
</tr>
<tr>
<td>WHEN?</td>
<td>Adaptation Type</td>
<td>Proactive, Reactive</td>
</tr>
<tr>
<td></td>
<td>Adaptation Period</td>
<td>Periodic</td>
</tr>
<tr>
<td></td>
<td>Adaptation Interval</td>
<td>short, variable</td>
</tr>
<tr>
<td>WHAT?</td>
<td>Resource Type</td>
<td>Computing</td>
</tr>
<tr>
<td></td>
<td>Granularity</td>
<td>Virtual Machine</td>
</tr>
<tr>
<td></td>
<td>Workload Type</td>
<td>Interactive</td>
</tr>
<tr>
<td></td>
<td>Metrics</td>
<td>CPU Utilization, Response Time, Arrival Rate</td>
</tr>
<tr>
<td>WHY?</td>
<td>Adaptation Goal</td>
<td>Minimize Costs</td>
</tr>
<tr>
<td>HOW?</td>
<td>Action to take</td>
<td>Allocate/Deallocate VMs, Reroute Traffic</td>
</tr>
<tr>
<td></td>
<td>Adaptation Policy</td>
<td>Optimal Solution, Heuristics</td>
</tr>
<tr>
<td></td>
<td>Evaluation</td>
<td>Prototype, Simulation, Analytical</td>
</tr>
</tbody>
</table>

has been adopted. The metrics used to drive the adaptation are response time, CPU Utilization and request arrival rate.

The goal for service level provisioning we consider is cost minimization subject to QoS constraints. A formulation of the VM allocation problem as an optimization problem is presented in section 5.1.2 for the single provider case and in section 6.1.2 for the cloud federation scenario.

In all the considered scenarios we use the allocation/deallocation of VMs to perform service level provisioning. In the cloud federation case traffic redirection is also adopted. Since the optimal solution for the VM allocation problem is very difficult or even infeasible to calculate, heuristics algorithms are used. To compare the performance of the heuristic policies proposed, in a single case the optimal solution is
3.4. Thesis Contributions vs Taxonomy

computed for a limited time horizon (see section 5.4.3).

The proposed architectures, models an algorithms are evaluated through both sim-
ulation and the implementation of a prototype for the single provider case and analyti-
cally for the cloud federation case.
In this chapter different autonomic architectures to perform service level provisioning in clouds will be proposed. After the introduction of a general architecture, a survey on current available features of the main IaaS providers will be presented. Starting from the general architecture and survey results, four different single-provider QoS-aware autonomic architectures will be introduced. Furthermore, an adaptation of the general architecture for the cloud federation scenario will be described.

Contents

4.1 Autonomic Architecture for VM Allocation ..................................... 58
4.2 IaaS providers features taxonomy .................................................. 59
  4.2.1 Taxonomy .............................................................................. 60
  4.2.2 Remarks ................................................................................. 65
4.3 QoS-aware architectures ................................................................. 65
  4.3.1 Extreme ASP control ............................................................... 69
  4.3.2 Full ASP control ................................................................. 71
  4.3.3 Partial ASP Control ............................................................... 73
  4.3.4 Limited ASP control ............................................................ 74
4.4 Cloud Federation Scenario ............................................................ 75
  4.4.1 Architecture of the inter-cloud resource manager ................. 76
4.1 Autonomic Architecture for VM Allocation

To address the problem of autonomic service level provisioning for SaaS providers that employ resources leased by one or more IaaS providers to offer services subject to SLAs, we consider VM allocation/deallocation as the only feasible adaptation action to react to traffic variations. The case under consideration is the one of the provider of a service $S$, accessible through a web service interface or a web interface (see Figure 4.1), that, to improve its performance in terms of availability and response time, leases computational capacity from one (or more) IaaS provider(s). Typically, an IaaS cloud allows to instantiate on-demand an arbitrary number of VMs running the same system image, and to distribute the incoming load among them. The load distribution is carried on by a request dispatcher (cloud dispatcher hereafter) controlled by the IaaS provider.

The service provider needs to allocate/deallocate VMs to fulfill SLAs stipulated with the service clients (end-users or service brokers) and aims to seek the maximum profit while minimizing operating costs. The incoming load addressed to the cloud-hosted application is distributed among active VMs by a cloud dispatcher on the basis of some dispatching policy. Performance measures (e.g. response time, request count, network traffic, VMs utilization, etc.) measured at the dispatcher are available to the SaaS provider.

The service provider has to decide the number of VMs that need to be leased from the IaaS provider by means of some allocation policy, which takes into account the values of the application performance indexes and the intensity of the incoming request.
4.2 IaaS providers features taxonomy

In what follows we suppose, without loss of generality, that a VM is allocated for a given time period (a multiple of the minimum allocation period) at a given cost, and that all the VMs have the same characteristics (CPU speed, memory, bandwidth etc).

4.2 IaaS providers features taxonomy

In this section our analysis is oriented to identify the interesting characterization elements of IaaS providers in an autonomic resource management perspective; that is, we identify the features useful (and needed) to realize self-adaptable solutions. In the specific we consider the type of SLAs, the billing model and the resource management features proposed to the end-user.
4.2.1 Taxonomy

Several companies enter the business of cloud computing organizing their portfolio around infrastructure, platform and software services. While an end-user (e.g., an ASP) has a rich choice, it is difficult to compare alternatives.

Here we try to shed light in the world of public IaaS providers analyzing seventeen of them (listed on the first column of Table 4.1). From our study we identified seven main attributes (see Fig 4.2) to characterize an IaaS provider: VM Customization model, Resource Usage Billing model, SLA model, Type of Interface, Load Balancing services, Monitoring tools and Auto Scaling services.

Figure 4.2: Taxonomy of advanced features offered by IaaS providers
The **Customization model** is intended as the degree of VM customizability offered by IaaS providers. We identified two types of customization models: Full and Partial. In the first model the end user can specify and, in some cases, modify at runtime, the amount of processing power, memory and storage of every VM. In the Partial customization model the IaaS provider offers a limited set of predefined VM instance types characterized by different computing power and characteristics. In this case a VM instance cannot be modified once running. For what concerns system images, all the providers offer a set of predefined images (from about ten to about one hundred, mostly based on Linux or Windows operating systems). Amazon Web Services provide a marketplace where users can share and buy customized system images.

The **Billing Model** is related to the policy adopted to bill the usage of VMs, storage and network resources. For what concern the VMs billing model, different temporal scales (per minute or per hour basis) are adopted by different providers. The majority of the considered IaaS providers applies a hourly billing. This imply that, if a customer uses a VM for a fraction of an hour, he is billed for the full hour. Few providers adopt a fine grained billing period of 5 minutes (CloudSigma) and 1 minute (Rackspace, Google Cloud Platform, Windows Azure). It is worth to note that Google, although using a 1-minute billing, has a minimum charge of 10-minutes. Many providers apply discounts if resources are reserved in advance for a longer period (from one month to one year). Amazon EC2 offers also variable price Spot Instances: users bid on spare Amazon EC2 instances and run them whenever their bid exceeds the current Spot Price, which varies in real-time based on supply and demand.
Chapter 4. Autonomic Architectures

It is important to remark here that we are considering only VM usage billing and not the one for extra services such as load balancing, network usage, storage, and auto scaling. Many providers apply extra fees if the network traffic or the number of I/O operations are higher than a predefined threshold.

It is worth to mention that, for the sake of automated VM provisioning, monthly based billing is not acceptable since it doesn’t allow to react quickly to sudden traffic changes. Furthermore, Spot Instances are not suitable for interactive services/applications but are effective only for batch processing.

- **Service Level Agreement Model.** Almost all the considered providers guarantee a minimum Service Level to users. In case of SLA violations users are partially or totally refunded by their IaaS provider. All the considered providers have SLAs based only on availability requirements (from 99.9% to 100% on a monthly basis). In some cases availability (or uptime) is referred to the cloud infrastructure (power and network) and single machines uptime is not considered; however a maximum time period for machine recovery is often defined. All of the considered providers also define a set of conditions in which SLAs may be violated without penalties (e.g., during scheduled maintenance). No provider currently considers SLAs based on performance indicators such as response time.

- **The Interface Type** determines how a user can interact with the cloud infrastructure. All the providers offer both a web interface and APIs to manage resources. The web GUIs allow users to perform simple actions such as creating VMs, starting/stopping VMs, configuring VMs and storage, setting firewall rules and, in some cases (e.g., AWS EC2 Management Console) give monitoring information
4.2. IaaS providers features taxonomy

like real-time CPU and memory utilization charts and VM status report. APIs give access to advanced functionalities such as load balancing and autoscaling configuration and detailed monitoring. Moreover, APIs allow users to interact with the cloud infrastructure through standard or customized client applications. Once that a VM has been created and started and that the firewall rules have been properly configured, all the providers allow direct access through standard remote terminal and remote desktop protocols like SSH and RDP.

- **Load Balancing** is the capability to distribute the incoming load among various VMs, possibly distributed in different zones. The load balancing feature is characterized by the layer at which balancing is performed (i.e., layer 4 or layer 7), by the supported protocols (e.g., HTTP, HTTPS, FTP, TCP, . . . ), by the balancing policy used and by the capability to support session persistence (stickiness). Moreover, load balancing can be performed at hardware or software level.

The load balancing policies can be classified into state-blind policies like round robin (RR), weighted round robin (WRR), Random (RND), source address based routing (Source), or state-aware policies such as least connection (LC) and weighted least connection (WLC). A particular state-aware balancing policy called Historical Intelligence (HI) that predicts the appropriate node to use by historical data analysis is offered by Storm [92]. Windows Azure [90] has a balancing policy based on performance that allows to distribute incoming traffic on the basis of users location and network latency, but cannot take into account real-time changes in network configuration or load. Different providers (e.g., Google Cloud Platform [70], Datapipe [93], GoGrid [94]) offer state-blind policies that performs balancing applying an hash function on source IP adresses.
Although most of the considered providers offer load balancing services, in some cases the balancing policy used is fixed and cannot be modified by users. Session persistence can be supported both at application or IP level. Only a limited set of providers (e.g., Amazon Web Services [12], AT&T Synaptic [95]) offer a cookie based (application level) session persistence.

- **The Monitoring Services** refer to mechanisms, offered by the IaaS provider, allowing users to monitor and collect system performance metrics (e.g. VM CPU and memory usage, load balancing statistics, etc.) at a fine level of granularity (1-5 minutes). Many providers offer a dashboard showing usage statistics and performance indicators or have an alarm system that warns users by email in case of failures. However the dashboard, usually accessible through a web portal, is useless to implement an autonomic system. For our purpose advanced built-in monitoring services accessible using APIs are needed. Examples of advanced monitoring systems are Amazon CloudWatch [32], HP Cloud Monitoring [96] and Windows Azure Diagnostics [90]. If advanced monitoring capabilities are missing, users have to implement their own monitoring agents to collect data and a monitoring manager component that aggregates and integrates the collected data.

- **Auto Scaling Services** refers to the capacity of the IaaS provider to automatically add/remove VMs on the basis of observed performance metrics (e.g. CPU utilization, number of service requests, average response time, etc.). Autoscaling services allow users to specify thresholds on performance metrics and actions to trigger when these thresholds are violated. A simple example of an autoscaling policy is to add a VM each time the average utilization of all VMs in a pool is
4.3. QoS-aware architectures

higher than 66%. The providers currently offering advanced autoscaling services are Amazon Web Services [12], Rackspace [13] and Windows Azure [90].

Table 4.1 shows a selection of the features implemented by the IaaS cloud providers we considered. The Interface Type feature is omitted since all the providers offer both a web interface and APIs to interact with the cloud infrastructure.

4.2.2 Remarks

From the IaaS providers feature taxonomy it is possible to see that, although the number of providers and of the features they offer is rapidly increasing, no efficient ready-to-use solution for service level provisioning in clouds is currently available. Even if some providers are starting to offer autoscaling services, these services are still limited and poorly customizable. Moreover, the SLAs currently available on the market are limited only to availability guarantees and do not consider other QoS metrics like performance and security constraints. Nevertheless, the increasing availability of advanced monitoring and load balancing features and the use of advanced APIs allows user to build customized solutions for autonomic service level provisioning in clouds employing services offered by IaaS providers and integrating them with proprietary personalized components and services. In the next section we will propose different architectures for autonomic service level provisioning in clouds differing for the degree of control that the service provider has over the various architectural components.

4.3 QoS-aware architectures

As mentioned before, the effort of the community goes in the direction of providing solutions for the autonomic management of cloud resources and the MAPE-K (Moni-
Table 4.1: A selection of IaaS providers features (updated February 2014). "Y" in the load balancing column means "Yes, but policy not specified". The load balancing policies considered are Round Robin (RR), Weighted RR (WRR), Random (RND), source address based (So), Least Connection (LC), Weighted LC (WLC), Historical Intelligence (HI) and Performance based (Perf). The SLA refers to availability computed on a monthly base.

<table>
<thead>
<tr>
<th>IaaS Provider</th>
<th>Customization model</th>
<th>Billing model</th>
<th>Load Balancing</th>
<th>SLA (Avail)</th>
<th>Mon. services</th>
<th>A.S. services</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon Web Services [12]</td>
<td>Partial</td>
<td>Full</td>
<td>WRR, WLC, So</td>
<td>99.99%</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>AT&amp;T Synaptic [95]</td>
<td>Full</td>
<td>Partial</td>
<td>So</td>
<td>99.9%</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>CloudSigma [89]</td>
<td>Full</td>
<td>5m</td>
<td>Y</td>
<td>100%</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Datapipe [93]</td>
<td>ND</td>
<td>ND</td>
<td>Y</td>
<td>99.95%</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>dimensiondata [97]</td>
<td>Full</td>
<td>1h</td>
<td>LC</td>
<td>99.95%</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>ElasticHosts [98]</td>
<td>Full</td>
<td>1h</td>
<td>N</td>
<td>100%</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Google Cloud Platform [70]</td>
<td>Partial</td>
<td>1m</td>
<td>So</td>
<td>99.95%</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>HP Public Cloud [99]</td>
<td>Partial</td>
<td>1h</td>
<td>LC, RR</td>
<td>99.95%</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Joyent [100]</td>
<td>Partial</td>
<td>1h</td>
<td>Y</td>
<td>100%</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>layeredtech [101]</td>
<td>ND</td>
<td>ND</td>
<td>Y</td>
<td>99.95%</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>LunaCloud [102]</td>
<td>Full</td>
<td>1h</td>
<td>Y</td>
<td>99.99%</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Rackspace [13]</td>
<td>Partial</td>
<td>1m</td>
<td>WRR, WLC, Rnd</td>
<td>99.99%</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>SoftLayer [103]</td>
<td>Partial</td>
<td>In</td>
<td>N</td>
<td>100%</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Storm on Demand [102]</td>
<td>Partial</td>
<td>In</td>
<td>LC</td>
<td>100%</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>AT&amp;T Synaptic [95]</td>
<td>Full</td>
<td>Partial</td>
<td>So</td>
<td>99.9%</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Google Cloud Platform [70]</td>
<td>Partial</td>
<td>1m</td>
<td>So</td>
<td>99.95%</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>HP Public Cloud [99]</td>
<td>Partial</td>
<td>1h</td>
<td>LC, RR</td>
<td>99.95%</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Joyent [100]</td>
<td>Partial</td>
<td>1h</td>
<td>Y</td>
<td>100%</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>layeredtech [101]</td>
<td>ND</td>
<td>ND</td>
<td>Y</td>
<td>99.95%</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>LunaCloud [102]</td>
<td>Full</td>
<td>1h</td>
<td>Y</td>
<td>99.99%</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Rackspace [13]</td>
<td>Partial</td>
<td>1m</td>
<td>WRR, WLC, Rnd</td>
<td>99.99%</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>
4.3. QoS-aware architectures

As shown in Figure 4.3, the general components of the proposed autonomic QoS-aware service provisioning architecture are:

- **Performance and workload monitors**
  
  These modules are the sensors of the QoS-aware service provisioning system and are in charge of collecting information on the performance state of the computational, storage and network resources used, as well as on the workload submitted by users (e.g. requests rate, type, size). Through Monitors relevant performance
indexes (e.g., response time, network traffic, CPU utilization, etc.) and workload information are collected, aggregated and elaborated.

- **SLA Analyzer**
  This module is in charge of elaborating and analyzing data collected by monitors. Moreover, the Analyzer should trigger the Planner component if significant changes in the data pattern (e.g., SLA violations or abnormal workload fluctuations) are detected. Data obtained by monitors (average response time, observed and expected arrival rate, etc.) are examined by this component to check whether adaptive actions are needed to guarantee the requested level of QoS.

- **Planner (or Provisioning Manager)**
  This module, instrumented with a system model, evaluates the system adaptation actions (e.g., VM allocation or deallocation, configuration changes) that must be operated to guarantee SLAs and optimize costs. Reconfiguration is triggered by the Analyzer and the system model is usually parameterized with information collected by the performance and workload monitors and elaborated by the Analyzer. If an adaptation is needed, the Planner decides, on the bases of the system performance model and of a resource allocation algorithm, the corrective actions to take (e.g., how many VMs to allocate or deallocate).

- **VM Allocator (or Resource Manager)**
  This component is in charge to actuate the strategies determined by the Planner to properly dimension and manage resources. Although here we consider VM allocation/deallocation as the main adaptive action, a more general Resource Manager component capable of performing different actions such as resizing the
4.3. QoS-aware architectures

capacity of running VMs, rerouting traffic and performing admission control can be considered.

- **Load Balancer**

  This component distributes requests among the instantiated resources (e.g., application servers, web server, and databases) running on leased virtual machines. This module should also be capable of guaranteeing session persistency.

With the proposed taxonomy we show that an ASP has various options to manage cloud resources and services, ranging from: to completely use an IaaS integrated solution (with a limited degree of customization), to build its own autonomic service provisioning system using some features and services offered by an IaaS provider, or to implement all the required features and services building a customized autonomic system inside a private cloud.

In the following we propose and discuss four different design solutions, namely: *Extreme ASP control, Full ASP control, Partial ASP Control*, and *Limited ASP control*.

### 4.3.1 Extreme ASP control

As shown in Figure 4.4, in this architecture all the cloud layers are under control of the Application Service Provider.

Performance measures are collected by *Monitoring Agents* placed on the VMs and on the *Load Balancer*.

The *Performance Monitor* and *Workload Monitor* periodically collect performance indexes related to the application, the virtualized resources usage and the workload
intensity. Moreover, depending on its sophistication, the monitoring components may aggregate, estimate and forecast the system performance level and resources demand.

The **SLA Analyzer** determines whether a new allocation decision has to be taken (for example if a violation is observed or predicted or at fixed time intervals) and, if necessary, triggers the Provisioning Manager.

The **Provisioning Manager** is the core of the system. As before mentioned it is equipped with a system model that is solved to determine the adaptive action to take in order to maintain the desired system performance level also if the system state and workload condition change. The system model is parameterized with observed and estimated system state information from the performance and workload monitors. The system model is typically used to derive an optimization problem. The solution of this optimization model produce a resource management plan, that determines the adaptation actions to take. The adaptation actions are typically determined solving an opti-
mization problem (exact solution) or through heuristic algorithms (sub-optimal solution).

The resource management plan is actuated by the VM Allocator service, responsible for the allocation and deallocation of VMs.

The Load Balancer distributes incoming requests among active VMs and is capable to manage sticky user sessions; it may be either one-way or two-way.

Finally, the Knowledge module is intended as the set of mechanisms used to share information among all the other modules. Typically there are sensors that acquire data (e.g., the performance and workload monitors), software module that clean/post-process data and a repository (e.g., a database) where data are stored. Moreover a protocol used to exchange information has to be defined (e.g., each module can extract data directly from the database, or a system module can provided data as input parameters of the API used to invoke the functionalities offered by another module).

The Extreme ASP control is a solution suitable for in-house data centers (or private clouds), where no resources are leased from IaaS providers: the ASP has total control over all the components at the cost to be forced to build and manage its own data center.

4.3.2 Full ASP control

In this architectural configuration (see Fig. 4.5), the ASP has total control over all the MAPE-K cycle phases. VMs are managed by the ASP using IaaS provider VM Allocation Service APIs. The VMs and applications performance are directly monitored by the ASP installing monitoring agents on the VMs. A possible variant of this architecture could be realized if a Performance Monitoring Service is provided. In this case the
Chapter 4. Autonomic Architectures

IaaS manages its own monitoring agents that collect aggregated VMs and load balancer performance indexes and provide them to the Performance and Workload Monitors.

This architecture has several benefits. First, it allows the ASP to gain total control over the Analyze and Plan phases of the autonomic cycle. Second, the ASP can avoid the high costs for starting-up, running and managing a data center, and, third, it can take advantage of the high scalability and availability guaranteed by a public cloud infrastructure. The main drawback is that the ASP is required to set up a robust and scalable network infrastructure including a Load Balancer to manage the incoming flow of requests (and outgoing traffic). To overcome this issue, at the drawback of a reduced degree of control over the load balancing policies, load balancing features offered by many IaaS providers could be used.
4.3. QoS-aware architectures

4.3.3 Partial ASP Control

This architecture overcomes some drawbacks of the Full ASP control above mentioned, but leaves the ASP in control of the Analyze and Plan phases (see Fig. 4.6).

The performance and workload monitors elaborate data collected by the IaaS Performance Monitoring Service and pass them to the SLA Analyzer. When the SLA Analyzer detects the need to change the number of allocated resources, it invokes the Provisioning Manager that determines the adaptive actions to perform. Therefore, the Provisioning Manager uses the VM Allocation service APIs to actuate the adaptation actions.

In our vision, this solution is a good tradeoff between the need of the ASP to have total control over the components that perform the adaptation (i.e., Performance Monitor, Workload Monitor, SLA Analyzer and Provisioning Manager) and the opportunity to exploit the scalable and robust load balancing and monitoring functionalities offered.
by IaaS providers. For example, to setup a custom and robust load balancer, an ASP must acquire, at least, gigabit bandwidth connection and a dedicated load balancing device supporting gigabit throughput, plus costs of personnel. On the contrary, using a cloud load balancing service is cheaper and reduce infrastructure and personnel costs.

Concerning the Performance Monitoring service, if it does not provide enough information for the adaptation algorithm, additional monitoring agents could be installed by the ASP.

### 4.3.4 Limited ASP control

This architecture can be implemented only employing resources from IaaS providers that offer auto scaling functionalities. In this case the IaaS provider controls all the MAPE-K phases: we assume that the SLA Analyzer and the Provisioning Manager are part of the Auto Scaling Service offered by the IaaS provider. The ASP is only responsible to set the appropriate scaling rules on the IaaS provider auto scaling component.

![Limited ASP Control architecture](image)

Figure 4.7: Limited ASP Control architecture
4.4 Cloud Federation Scenario

This architecture variant is very easy and fast to implement but suffers the drawbacks imposed by the limited customizability of autoscaling services currently offered by IaaS providers (see Section 4.2).

4.4 Cloud Federation Scenario

In this section we consider an inter-cloud composed by various IaaS providers collaborating with each others inside a cloud federation. In this scenario we take in consideration the perspective of a service provider willing to exploit the outsourcing opportunity offered by the cloud federation to satisfy performance, availability and security requirements of its customers minimizing costs. In our context a federation is a set of service providers sharing interoperability mechanisms and agreeing on a set of policies regulating outsourcing and insourcing of resources.

The considered scenario is described in Figure 4.8. We suppose the service providers collaborating in the federation are logically partitioned into different zones. A clustering of providers into zones can be obtained considering their geographical location or the autonomous system the service providers belong to and/or manages. Being our model independent from the specific clustering, we do not make any specific assumption on the nature of the service providers partitioning.

Moreover, we assume that also customers are partitioned into the same geographical zones. If a customer in zone $i$ accesses a resource in zone $j$, and $i \neq j$, he experiences a not negligible latency. In addition, customers may be divided into two or more service classes differentiated by performance, security and availability requirements.

Each service provider shares a set of resources, i.e. VMs, that can be allocated by other federation members paying an hourly usage cost.
The application service provider is equipped with a component, the *cloud manager*, described in the next section, that is in charge of determining the resource provisioning and traffic distribution policies. The service provider receives the incoming traffic and, using the cloud manager, decides where it should be managed inside the federation, allocates/deallocates resources (if needed) and redirects requests to the appropriate data centers.

### 4.4.1 Architecture of the inter-cloud resource manager

In the scenario under consideration we suppose that all the federation members are equipped with a *cloud manager* component in charge of supervising interoperability.
4.4. Cloud Federation Scenario

among federation members and of managing resources (locally and in the federation). Interoperability is not a matter of this thesis while we concentrate our attention on autonomic resource provisioning.

Figure 4.9: General Architecture of the inter-cloud resource manager

Extending the proposed QoS-aware architectures to inter-clouds, we enhance the autonomic resource provisioning architecture as follows (see Figure 4.9). In the monitoring phase, a Federation Monitor module is introduced to gather information on the available resources in the federation and on the level of performance, availability and security they provide, and at what price (variable over time). The SLA analyzer has to be enhanced to catch the new state information about the federation and to trigger
adaptation actions when SLAs cannot be satisfied without changing the current configuration. The Provisioning Manager (or Planner), if triggered by the SLA manager, evaluates the new system configuration on the basis of the information collected by the monitors. Of course, the provisioning manager should be equipped with a model capable to determine the appropriate configuration of the extended infrastructure and how to properly balance the load among the whole set of available resources. The cloud manager load balancer is a one-way dispatcher capable to perform multi cluster dispatching using DNS redirection. In this way the cloud manager should not become the system bottleneck because, once the load distribution decision is taken, all the sessions will be managed directly by the remote service providers. The outsourcing model we designed to equip the Planner will be described in section 6.1.
In this chapter the problem of optimal VM allocation for an ASP using resources from a single IaaS provider is modeled. A SLA considering performance metrics is defined and a formulation as an optimization problem is proposed. Different reactive and proactive heuristic policies to perform VM allocation and a stochastic workload model used to generate synthetic workloads that reproduce the time dependency and bursty characteristics of real workloads are introduced. Moreover, a system prototype implementation of the Partial ASP control and Limited ASP control architectures is presented. Finally, extensive experiments to evaluate the proposed architectures, models, algorithms and prototype will be described and discussed.

Contents

5.1 Single Provider Optimal VM Allocation Model . . . . . . . . . 80
   5.1.1 SLA Definition . . . . . . . . . . . . . . . . . . . . . . 81
   5.1.2 Problem Formulation . . . . . . . . . . . . . . . . . . . . 83
5.2 Heuristic VM Allocation . . . . . . . . . . . . . . . . . . . . . 88
5.3 Workload Model . . . . . . . . . . . . . . . . . . . . . . . . . 91
5.4 VM Allocation Experimental Evaluation . . . . . . . . . . . . . 93
   5.4.1 Workload . . . . . . . . . . . . . . . . . . . . . . . . . . 96
   5.4.2 Performance Metrics . . . . . . . . . . . . . . . . . . . . . 98
   5.4.3 Optimal VM Allocation . . . . . . . . . . . . . . . . . . . 98
   5.4.4 Heuristic VM Allocation . . . . . . . . . . . . . . . . . . . 101
   5.4.5 Heuristic versus Optimal VM Allocation . . . . . . . . . . 103
5.5 Prototype Implementation and Experimental Evaluation . . . . 104
5.1 Single Provider Optimal VM Allocation Model

In this section we present and model the VM allocation problem for an ASP that uses resources from a single IaaS provider. We assume time to be logically divided into slots of equal length $\tau$ and that VM allocation/deallocation can only occur at slot boundaries. We also assume that the different performance measures, e.g., expected response time, are computed and averaged over a period of time of one slot. Hereafter, for the sake of simplicity, we will assume that all time intervals are integer multiples of a slot.

We consider an application service provider whose goal is to determine a VM allocation schedule over a time horizon of length $T$ which minimizes the allocation cost paid to the IaaS provider while guaranteeing, at the same time, some service level objective defined in the SLA offered by the application service provider to its users. The allocation schedule takes the form of a sequence $z_1, z_2, \ldots$, where $z_i$ is the number of VMs allocated at the beginning of slot $i$. Without loss of generality, we assume that a VM can be allocated for a period multiple of time interval of length $W_a$ (corresponding to $M_{W_a} = W_a/\tau$ time slots), after which it is automatically deallocated (unless implicitly renewed if a new VM is allocated at the same time).
5.1. Single Provider Optimal VM Allocation Model

5.1.1 SLA Definition

We assume that the SLA offered by the application service provider to its users is a tuple \( \langle R_{\text{max}}, W, V_{\text{max}} \rangle \) where:

- \( R_{\text{max}} \) is the maximum value allowed for the average application response time, \textit{i.e.}, the SLO;
- \( W \) is the SLA time window length (we will denote by \( M_W = W/\tau \) the corresponding number of slots);
- \( V_{\text{max}} \) is the maximum fraction of SLO violations allowed during \( W \) (\( 0 \leq V_{\text{max}} \leq 1 \)).

The SLO is violated if \( r_i > R_{\text{max}} \), where \( r_i \) is the measured application response time at time slot \( i \).

If we suppose the SLA is evaluated at each slot, at time slot \( i \) the SLA is violated if a fraction \( V_i > V_{\text{max}} \) violations have been observed over the last time window \( W \), \textit{i.e.}, over the last \( M_W \) slots \( \{i - M_W + 1, \ldots, i\} \), where

\[
V_i = \frac{1}{M_W} \sum_{j=i-M_W+1}^{i} 1 \{r_j > R_{\text{max}}\}, \quad 0 \leq V_i \leq 1. \quad (5.1)
\]

The application service provider periodically monitors the performance of the virtual machines (specifically, their service rate \( \mu \)) and the request arrival rate \( \lambda \) to the application running in the cloud through the Workload Monitor and Performance Monitor components, as described in Chapter 4. Let us suppose that the ASP system is also instrumented with a model to compute the application response time as a function of \( \mu \), \( \lambda \), and the number \( x \) of allocated VMs. Considering that the fraction of SLO violations
can be evaluated periodically by measuring how many times the request response time exceeds the $R_{max}$ threshold, we can evaluate the number $x$ of VMs that are needed to satisfy the SLA.

To this end, we model the IaaS system composed by the Load Balancing Service and the set of VMs as a network of $M/G/1/PS$ queues. Under the hypothesis that the demand follows a Poisson distribution and that the flow of incoming requests is equally distributed among the set of $x$ VMs, the expected application response time $r$ is given by

$$r = \frac{x}{\mu x - \lambda} \quad (5.2)$$

Using this simple model, it is easy to compute the number of VMs such that $r \leq R_{max}$ (in our model, we do not consider the delays possibly introduced by the application service provider and the Load Balancing Service).

![Figure 5.1: Response Time](image)
5.1. Single Provider Optimal VM Allocation Model

As an example, in Fig. 5.1 and 5.2 we show the relationship between the estimated response time and the estimated number of SLO violations. We consider a window of $M_W = 6$ time slots, $R_{max} = 0.55$ and $V_{max} = 0.2$, i.e., at most 20% of the response time values can exceed the $R_{max}$ threshold. In Fig. 5.1 the curve of the expected response time $r_i$, evaluated at time slot $i$, shows that moving the time window $\{i - 5, \ldots, i\}$ from $i = 6$ to $i = 16$ we observe the following numbers of SLO violations: 1 for $i = 6$ and $i \geq 8$, 2 for $i = 7$ and $i = 13$ and 3 for $8 \leq i \leq 12$. As a result, the SLA is violated for $7 \leq i \leq 14$ (see Fig. 5.2).

5.1.2 Problem Formulation

We now describe how to determine the optimal VM allocation over a time horizon $T$, starting from the problem constants and variables.
Chapter 5. Single Provider VM Allocation

Constants

- $T$ is the time interval considered by the ASP for the VM allocation problem; $M$ is the corresponding number of slots;

- $W_a$ is the VM allocation time period; $M_{W_a}$ is the corresponding number of slots;

- $\lambda_i$ is the request arrival rate expected at time slot $i \in \{1, \ldots, M\}$;

- $\mu$ is the service rate of each VM. We assume that the service time is exponentially distributed and that all the VMs have the same performance characteristics and therefore the same service rate;

- $x_{i,\text{min}}$ is the minimum required number of VMs to guarantee a response time of $R_{\text{max}}$. From (5.2), we readily obtain:

$$x_{i,\text{min}} = \frac{\lambda_i}{\mu - 1/R_{\text{max}}}$$  \hspace{1cm} (5.3)

- $c_i \in \mathbb{R}$ is the cost to use a VM for $W_a$ time units when the allocation is operated at time $i \in [1, M]$.

Usually, $c_i$ is constant, but different billing models where the allocation cost changes over time (e.g., due to energy costs) can be considered.

Variables

- $z_i \in \mathbb{N}$ is the number of VMs to be allocated at the beginning of time slot $i \in \{1, \ldots, M\}$;
5.1. Single Provider Optimal VM Allocation Model

- \( x_i \in \mathbb{N} \) is the number of VMs available during the time slot \( i \in \{1, \ldots, M\} \). We have,

\[
x_i = \begin{cases} 
  x_{i-1} + z_i & \text{if } 1 < i < M_{W_a} \\
  x_{i-1} + z_i - z_{i-M_{W_a}} & \text{if } i \geq M_{W_a}
\end{cases}
\]  

(5.4)

i.e., the number of VMs available during slot \( i \) is equal to the number of VMs available during the previous slot \( (x_{i-1}) \) plus the number of new allocated VMs \( (z_i) \) minus the number of VMs whose allocation period has just terminated \( (z_{i-M_{W_a}}) \);

- \( \xi_i \) is the number of additional VMs required to ensure a response time equal to \( R_{\text{max}} \) during time slot \( i \).

- \( \xi = \max_{i=1}^{M} \xi_i \) is the maximum of the \( \xi_i \) over the time horizon \( T \);

- \( y_i \in \{0, 1\} \) is a binary variable which indicates whether there is a SLO violation at slot \( i \) or not.

Cost and Objective Function  Our goal is to minimize the allocation cost over the interval \( T \). The VM allocation cost is simply given by:

\[
C(z) = \sum_{i=1}^{M} c_i z_i.
\]  

(5.5)

Nevertheless, as objective function for the VM allocation problem, we consider the following more general objective function:

\[
F(z, \xi) = \sum_{i=1}^{M} c_i z_i + K\xi
\]  

(5.6)

which allows us to explore the trade-off between allocation cost and user perceived performance, expressed as function of the degree of SLO violations. Here \( K \geq 0 \) is a suitable non-negative constant: if \( K = 0 \), \( F(z, \xi) = C(z) \); otherwise, \( F(z, \xi) \)
Chapter 5. Single Provider VM Allocation

is a weighted sum of the allocation cost and the overall level of SLO violation, here simply captured by the maximum number of additional VMs which would be required to ensure the SLO over $T$.

**Optimization Problem** Under the assumption that $r_i$, defined in (5.2), approximates the observed average response time of the set of $x_i$ VMs, the VM allocation $z = (z_1, z_2, ..., z_M)$ of VMs, which minimizes the objective function $F(z, \xi)$ while guaranteeing the fulfillment of the SLA $\langle R_{max}, W, V_{max} \rangle$, is obtained by the solution of the following optimization problem:

$$\min F(z, \xi) = \sum_{i=1}^{M} c_i z_i + K \xi$$

subject to:

1. $x_i = x_{i-1} + z_i$, \hspace{1cm} $1 < i < M_{Wa}$ (5.7)
2. $x_i = x_{i-1} + z_i - z_{i-M_{Wa}}$, \hspace{1cm} $i \geq M_{Wa}$ (5.8)
3. $x_i + \xi_i \geq x_{i,\text{min}}$, \hspace{1cm} $i \in \{1, \ldots, M\}$ (5.9)
4. $\xi_i \leq y_i B$, \hspace{1cm} $i \in \{1, \ldots, M\}$ (5.10)
5. $\xi_i \leq \xi$, \hspace{1cm} $i \in \{1, \ldots, M\}$ (5.11)

$$\frac{1}{M_W} \sum_{j=i-M_W+1}^{i} y_j \leq V_{max}$$ \hspace{1cm} $i \in \{M_W, \ldots, M\}$ (5.12)

$$x_i \leq X_{max}$$ \hspace{1cm} $i \in \{1, \ldots, M\}$ (5.13)

$$z_i, \xi_i, \xi \geq 0$$ \hspace{1cm} $i \in \{1, \ldots, M\}$ (5.14)
$$x_i, z_i \in \mathbb{N}$$ \hspace{1cm} $i \in \{1, \ldots, M\}$ (5.15)

(5.7)-(5.8) are just equations (5.4) which relate the number of available VMs $x_i$ with the number of allocated VMs $z_i$. Inequalities (5.9) are the SLO constraints: if $x_i \geq x_{i,\text{min}}$, the SLO is satisfied and $\xi = 0$; if, instead, $x_i \leq x_{i,\text{min}}$, there is a SLO violation, $\xi$ is then equal to the number of additional VMs which would ensure a response time
5.1. Single Provider Optimal VM Allocation Model

equal to \( R_{\text{max}} \). Inequalities (5.10) ensure that \( y_i = 1 \) whenever \( \xi_i > 0 \), i.e., when there is a SLO violation. Here \( B \) is a large constant. (5.12) is the SLA constraint on the maximum number of SLO violations. The left hand side \( \frac{1}{M_W} \sum_{j=i-M_W+1}^{i} y_j \) is \( V_i \), the number of SLO violations in the interval \( \{i-M_W+1, \ldots, i\} \). Finally, Eqs. (5.13) are the functional constraints: here we assume that the maximum number of VMs that can be allocated at any given time cannot exceed a given constant \( X_{\text{max}} \). We observe that, while we included them in the constraints, it is not necessary to consider the integrality constraints for the variable \( z_i, i = \{1, \ldots, M\} \) which are implicitly enforced by the constraints (5.7)-(5.8) and by the integrality constraints on the variables \( x_i \).

For the sake of readability, Tab. 5.1 summarizes the notation we used in the problem formulation.

Table 5.1: Main notation adopted in the optimization problem formulation

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( R_{\text{max}} )</td>
<td>Maximum average application response time</td>
</tr>
<tr>
<td>( M_W )</td>
<td>Number of slots in the SLA time window ( W )</td>
</tr>
<tr>
<td>( V_{\text{max}} )</td>
<td>Maximum fraction of SLO violations allowed in ( W )</td>
</tr>
<tr>
<td>( T )</td>
<td>Time interval considered by the ASP for VM allocation</td>
</tr>
<tr>
<td>( M )</td>
<td>Number of slots in ( T )</td>
</tr>
<tr>
<td>( M_{W_a} )</td>
<td>Number of slots in the VM allocation time period ( W_a )</td>
</tr>
<tr>
<td>( \lambda_i )</td>
<td>Expected request arrival rate at slot ( i )</td>
</tr>
<tr>
<td>( \mu )</td>
<td>Service rate of each VM</td>
</tr>
<tr>
<td>( x_{i,\text{min}} )</td>
<td>Minimum number of VMs required to guarantee ( R_{\text{max}} ) at slot ( i )</td>
</tr>
<tr>
<td>( c_i )</td>
<td>VM allocation cost for ( W_a )</td>
</tr>
<tr>
<td>( z_i )</td>
<td>Number of VMs to be allocated at slot ( i )</td>
</tr>
<tr>
<td>( x_i )</td>
<td>Number of VMs available during slot ( i )</td>
</tr>
<tr>
<td>( \xi_i )</td>
<td>Number of additional VMs required to guarantee ( R_{\text{max}} ) for slot ( i )</td>
</tr>
<tr>
<td>( \xi )</td>
<td>Maximum ( \xi_i )</td>
</tr>
<tr>
<td>( y_i )</td>
<td>Binary variable indicating whether there is a SLO violation at slot ( i )</td>
</tr>
<tr>
<td>( X_{\text{max}} )</td>
<td>Maximum number of VMs that could be allocated</td>
</tr>
<tr>
<td>( K )</td>
<td>Weight of the overall level of SLO violations</td>
</tr>
</tbody>
</table>
Chapter 5. Single Provider VM Allocation

The proposed optimization problem is a Mixed Integer Linear Programming (MILP) problem which can be solved via standard techniques. Since the complexity is exponential in the number of integer variables, the computational cost might turn to be prohibitive for online operations, unless we consider small time interval $T$ and/or coarser time granularity, i.e., large $\tau$. In section 5.2 we will present heuristics to solve the VM allocation problem.

5.2 Heuristic VM Allocation

Here we propose different heuristic VM allocation policies that can be classified in two broad categories: reactive and proactive. Reactive policies decide how many VMs have to be allocated on the basis of the request arrival rate recently observed. On the contrary, proactive policies evaluate the number of needed VMs on the basis of the prediction of future workload through autoregressive models. We first present the generic heuristic VM allocation process and then describe the specific heuristic policies.

Algorithm 1 Heuristic VM allocation

1: $\text{reqVM}_i = \text{heuristicAlg}(\log)$; \{heuristicAlg$(\log)$ estimates the number of VMs needed to avoid SLO violations in the time slot $i$ using one of the heuristic policies\}
2: if $\text{reqVM}_i > x_{i-1}$ then
3: \hspace{1cm} $x_i = \text{reqVM}_i$
4: else if $\text{reqVM}_i < x_{i-1}$ then
5: \hspace{1cm} $x_i = \min\{(x_{i-1} - \text{reqVM}_i), \text{expVM}_{i-1}\}$; \{expVM$_{i-1}$ = number of VMs whose billing period expires at the end of time slot $i - 1$ \}
6: end if

Algorithm [1] describes the heuristic VM allocation process. The first step (line 1) is to estimate the number of VMs reqVM$_i$ required to guarantee the SLA fulfillment,
5.2. Heuristic VM Allocation

using one of the heuristic policies and the past history of request arrivals (log). Known \( \text{reqVM}_i \), the allocation/deallocation decisions are taken as follows. If the number of required VMs \( \text{reqVM}_i \) is equal to the number of already running (active) VMs \( x_i - 1 \), then no action is taken. If the number of required VMs is greater than the number of active VMs (line 2), then \( \text{reqVM}_i - x_i - 1 \) VMs will be allocated at the beginning of the next time slot. If the number of required VMs is lower than the number of active VMs (line 4), the number of VMs that will be deallocated at the beginning of the next slot is equal to \( \min\{(x_i - 1 - \text{reqVM}_i), \text{expVM}_{i-1}\} \), where \( \text{expVM}_{i-1} \) is the number of VMs whose billing period expires at the end of the current time slot. In such a way, the deallocation of VMs before the end of the billing period is prevented, thus avoiding the waste of already paid computational power.

The proactive policies we propose use a Recursive Least Square (RLS) based prediction algorithm \([106]\) to forecast the workload in the next time slot of length \( \tau \). Specifically, the prediction algorithm (detailed in section 5.6) uses an autoregressive process of order 2 \( AR(2) \), whose weights and white noise process variance are continuously estimated using the RLS method \([84]\).

For example, let us suppose to perform one step ahead prediction. At time \( i \), we forecast the average request arrival rate \( \hat{\lambda}_i \) and the white noise variance \( \hat{\sigma}^2(i) \) of the process. Therefore, we set the estimated arrival rate \( \hat{\lambda}_i \) to the \( Y\% \)-upper bound of \( \bar{\lambda}_i \), that is \( \hat{\lambda}_i = \bar{\lambda}_i + a(Y) \times \hat{\sigma}^2(i) \), where \( a(Y) \) is equal, for example, to 1.96 for the 95-percentile upper bound \( (Y = 0.95) \) or to 2.56 for the 99-percentile upper bound \( (Y = 0.99) \).

We propose the following heuristic allocation policies.

*Exact knowledge (EK)* assumes that the average arrival rate \( \lambda_i \) in the upcoming time
slot \( i \) is known exactly at the beginning of that time slot and that the arrivals are uniformly distributed within the single time slots. The minimum number of VMs needed to meet the SLO in every slot is given by Eq. 5.3. To prevent workload fluctuations that will exceed \( \lambda_i \) and therefore that could result in SLO violations, this policy overestimates the arrival rate as follows:\n\[ \hat{\lambda}_i = (1 + \alpha)\lambda_i, \]
where \( \alpha > 0 \).

**Reactive 1 step early (r-1)** measures the average arrival rate \( \lambda_{i-1} \) observed in the previous time slot \( i-1 \) and sizes the set of VMs assuming that the estimated arrival rate in the following time slot will be \( \hat{\lambda}_i = (1 + \alpha)\lambda_{i-1} \), where \( \alpha > 0 \).

As in the EK policy, the number of needed resources in every slot is given by Eq. 5.3 taking into account only the constraint on the maximum response time \( R_{\text{max}} \).

**Proactive 1 step ahead, Y% (p1-Y)** predicts the one step ahead \( Y \% \) upper bound of the request arrival rate using the RLS-based prediction model described above. The number of needed VMs is computed by Eq. 5.3 assuming that the estimated arrival rate is given by \( \hat{\lambda}_i = \bar{\lambda}_i + a(Y)\sigma^2(i) \). In the experiments (see section 5.4), we will consider \( Y = 0.95 \) and \( Y = 0.99 \). Since the prediction is carried out only one step ahead, this policy is not capable to prevent SLA violations but only to guarantee that the SLO is honored in the current time slot.

**Proactive N steps ahead, Y% (pN-Y)** predicts the \( Y \% \) upper bound of the request arrival rate using the RLS-based prediction model described above. The prediction is carried on \( N \) steps ahead, that is for the next \( N\tau \) time period. The number of VMs needed to guarantee the SLA is computed from Eqs. 5.1 and 5.2 assuming
that the next \( N \) request arrival rates are given by \( \hat{\lambda}_i = \bar{\lambda}_i + a(Y)\hat{\sigma}^2(i) \). In the experiments detailed in section 5.4, we will consider \( Y = 0.95 \) and \( Y = 0.99 \) and \( 12 \leq N \leq 48 \).

### 5.3 Workload Model

One of the main problems in evaluating the performance of resource management policies in the cloud environment is the lack of public available benchmark tools, access logs, and workload characterization studies. To address this issue, here we consider a stochastic workload model that reproduces the time dependency and bursty characteristics of real workloads. Specifically, using standard techniques, we first model an excerpt of the 1998 FIFA World Cup traces \[107\] with a discrete time Markov chain (DTMC). This trace, although may be considered "old", is very interesting because is a good example of a workload very hard to predict. Indeed, in this case, peak demands are influenced by unpredictable factors such as the outcome of matches \[108\]. Then, through the obtained model, we generate a synthetic trace that has the same stochastic properties of the original trace.

Algorithm 2 shows the main steps of the process we use to generate the synthetic workload. This algorithm has been implemented in Matlab\textsuperscript{®}. The inputs are: the original trace (log); the average time window used for noise reduction (\( w \)); the number of states of the Markov process (MP) (\( n\text{State} \)); the number of samples to generate (\( n\text{Sample} \)). We remark that the states of the MP are equiprobable. The first step (line 2) is to remove the noise from the original trace log to obtain the trend of the arrivals. In such a way, the Markov process learns only the dynamic of the original workload trend and is not influenced by the noise observed in the original trace. To this purpose,
**Algorithm 2** Generation of synthetic workload

```plaintext
1: function Wlg(log, w, nState, nSample);
   {log is the original trace log; w is the average time windows for noise reduction;
    nState is the number of states of the MP; nSample is the number of samples
    that will be generated}
2:     noNoiseLog=RemoveNoise(log, w);
3:     states=ComputeDTMCStates(noNoiseLog, nState);
4:     P=ComputeStateTransitionProb(noNoiseLog, states);
5:     π=ComputeEquilibriumProbVector(P);
6:     p0=ComputeInitialState(π);
7:     wl=GenerateWorkload(P, p0, nSample, log, noNoiseLog);
8: end
```

we calculated the noise as the difference from the original trace and the same trace averaged over a moving time window of 10 minutes. The averaged log is then used to compute the states of the MP (line 3) and the state transition probability matrix $P$ (line 4). From this matrix, the equilibrium state vector $\pi = P\pi$ is computed (line 5) and the initial state $p_0$ is stochastically determined (line 6). Then, starting from $P$ and $\pi$, we generate a sequence of arrivals from the DTMC (line 7), that is our synthetic workload. In the synthetic workload generation, we add again a noise equivalent to that removed in the previous steps; specifically, we add to the mean arrival rate of each state a Gaussian noise with the same mean and variance of the previously removed noise.

Figure 5.3 shows the effect of the noise reduction on an excerpt of 3 days from the 1998 World Cup trace. Figures 5.5, 5.6, 5.7 show different traces generated using the DTMC with a different number of equiprobable states. Increasing the number of states of the discrete time Markov process, it is possible to capture better the dynamics of the original process. In the experiments illustrated in later in this chapter we consider 100
5.4 VM Allocation Experimental Evaluation

In the single provider scenario, to evaluate the performance of the heuristic policies proposed in section 5.2 versus the optimal VM allocation policy we proceed as follows.

- First, in Sec. 5.4.1 we define the parameters used to generate a synthetic workload from the stochastic workload model that presents the long range dependencies and bursty behavior described in section 5.3.

- Second, in Sec. 5.4.2 we define an appropriate set of performance metrics to evaluate and compare the proposed policies.

- Third, we present a large set of simulation experiments to evaluate the sensibility
Figure 5.4: Workload obtained removing noise

Figure 5.5: Synthetic workload generated using a 10 states DTMC
Figure 5.6: Synthetic workload generated using a 50 states DTMC

Figure 5.7: Synthetic workload generated using a 100 states DTMC
Chapter 5. Single Provider VM Allocation

of the proposed heuristic policies to their tunable parameters and to compare the performance of the heuristic policies with the optimal allocation.

The setting of the main system model parameters we used in the simulation experiments are reported in Tab. 5.2.

Table 5.2: Settings for the main system model parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Notation</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time horizon</td>
<td>$T$</td>
<td>4 - 168 hours</td>
</tr>
<tr>
<td>Length of VM allocation time</td>
<td>$W_a$</td>
<td>1 hour</td>
</tr>
<tr>
<td>Time slot</td>
<td>$\tau$</td>
<td>5 min.</td>
</tr>
<tr>
<td>Service rate of each VM</td>
<td>$\mu$</td>
<td>10 req/sec</td>
</tr>
<tr>
<td>Allocation cost</td>
<td>$c_i$</td>
<td>0.1 $ per hour</td>
</tr>
<tr>
<td>Maximum response time</td>
<td>$R_{max}$</td>
<td>0.5 sec</td>
</tr>
<tr>
<td>Maximum fraction of SLO violations</td>
<td>$V_{max}$</td>
<td>{0.1, 0.25, 0.35}</td>
</tr>
<tr>
<td>Length of SLA time window</td>
<td>$W$</td>
<td>{0.5$W_a$, $W_a$, 2$W_a$}</td>
</tr>
<tr>
<td>Weight of the overall level of SLO violations</td>
<td>$K$</td>
<td>0 - 10$^5$</td>
</tr>
</tbody>
</table>

5.4.1 Workload

In the experiments presented in the next sections, starting from the model described in section 5.1 we will generate and use two workloads that are characterized by the same intensity (see Table 5.3) but are different for length and shape (see Fig. 5.8). To characterize the burstiness of arrival times, in Table 5.3 we also report the index of dispersion for counts [109], which is calculated by considering the number of arrivals in each 5 minute long time interval for the entire workload traces.

To evaluate the heuristic policies we used 20 instances of a long workload lasting 168 hours (see Fig. 5.8(a)), while to compare the performance of the optimal allocation
5.4. VM Allocation Experimental Evaluation

policy with the heuristics we used 20 instances of a short workload lasting 4 hours (see Fig. 5.8(b)). The 20 workload instances are obtained by changing the random number sequences used to generate the stochastic variables. Each short workload instance has been extracted from a long workload instance.

Table 5.3: System workload characteristics

<table>
<thead>
<tr>
<th>Workload type</th>
<th>Characteristic</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Long</td>
<td>Duration (hours)</td>
<td>168</td>
</tr>
<tr>
<td></td>
<td>Granularity (sec)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Intensity (req/sec)</td>
<td>$48.71 \pm 1.78$ (avg.); 114 (95-perc); 229 (max); 5518.2</td>
</tr>
<tr>
<td></td>
<td>Index of dispersion</td>
<td></td>
</tr>
<tr>
<td>Short</td>
<td>Duration (hours)</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>Granularity (sec)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Intensity (req/sec)</td>
<td>$48.17 \pm 2.33$ (avg.); 136 (95-perc); 225 (max); 1892.4</td>
</tr>
<tr>
<td></td>
<td>Index of dispersion</td>
<td></td>
</tr>
</tbody>
</table>

Figure 5.8: The 20 workload instances used in the experiments: vertical dotted lines separate the runs, being the length of each run 168 and 4 hours, respectively.
Chapter 5. Single Provider VM Allocation

5.4.2 Performance Metrics

To evaluate the performance of the proposed VM allocation heuristic policies and the optimal policy we consider the following metrics:

- the *allocation cost* $C$ (measured in $\$\$ and over $T$) defined by Eq. 5.5. In our experiments we have observed that the randomly generated workload can have a very different intensity (see Fig. 5.8) that results in resource allocation costs with a high variance. Therefore, to take the worst cases into account we will consider the 95-percentile of $C$.

- The *SLA satisfaction factor* (SSF) over the time horizon $T$ defined as follows:

$$SSF = 1 - Pr\{V_i > V_{max}\}, \ 0 \leq SSF \leq 1$$

where $Pr\{V_i > V_{max}\} = \frac{1}{M-M_{wa}+1} \sum_{i=M_{wa}+1}^{M} 1\{V_i > V_{max}\}$, and $V_i$ is the fraction of SLO violations defined in Eq. 5.1.

- The *fraction of SLO violations* $V_i$ at time slot $i$. This latter allows us to evaluate the behavior of the optimal allocation policy under different combinations of tuning parameters. While the optimal allocation policy achieves $SSF = 1$ by definition, the number of SLO violations can range from 0 to $V_{max}$ with a non-negligible impact on the allocation cost.

5.4.3 Optimal VM Allocation

We first analyze some experimental results to show the behavior of the Optimal VM allocation policy based on the MILP optimization problem we have presented in Sec. 5.1.2. The SLA is $\langle R_{max} = 0.5 \text{ sec}, W = 1 \text{ h}, V_{max} = 0.25 \rangle$. The time horizon $T$ is 4 hours and the maximum number of VMs that can be allocated is $X_{max} = 2 \cdot \frac{R_{max} \max_{i=1}^{M} \lambda_i}{\mu R_{max} - 1}$. 

98
The first step is to identify the value of the parameter $K$ that minimizes the VM over-provisioning. To this purpose, we have conducted a set of experiments where $K$ ranges from 0 to 100.

The experiments confirm that for small values of $K$ the optimal strategy operates an under-provisioning of VMs that results in a higher percentage of SLO violations, always satisfying the SLA constraint $V_{\text{max}}$. For $K \geq 10$, the number of SLO violations goes toward zero, because resources are no under-provisioned. As example, Figs. 5.9 and 5.10 show the number of allocated VMs and the observed fraction of SLO violations for three different values of $K$. The cost and the average fraction of observed SLO violations are plotted in Fig. 5.11. The performance results for $K = 0, 0.1,$ and 1 are quite similar due to the large confidence interval. Furthermore, the confidence intervals for $K = 1$ and 5 overlap and thus the provider should not take the risk to have the same fraction of SLO violations while paying a higher allocation cost. Therefore, in the experiments that follow we will use $K = 1$.

Figure 5.9: VM allocation for $K = 0$, $K = 5$, and $K = 10$. The $x_i$ and $z_i$ values are rounded to the nearest integer.

As regards the sensitivity to the SLA time window length $W$, we have considered three different cases: $W/W_a = 0.5$, 1 and 2, where $W_a$ is the VM allocation time.
Chapter 5. Single Provider VM Allocation

Figure 5.10: Fraction of SLO violations for $K = 0$, $K = 5$, and $K = 10$

Figure 5.11: Optimal VM allocation cost and average fraction of SLO violations for different values of $K$

length which is fixed to 1 hour. The corresponding results are shown in Tab. 5.4 where we have set $V_{\text{max}} = 0.25$ to catch SLO violations also for $W/W_a = 0.5$, that is $W = 30$ min. The optimal allocation policy determines an under-provisioning of resources for values of $W \geq 1$ hour (see the 95-percentile for $V_i$). Therefore, in the remaining experiments we will set $W = W_a = 1$ hour.
5.4. VM Allocation Experimental Evaluation

Table 5.4: Optimal allocation: sensitivity to W (W_a = 1 h, V_{max} = 25%)

<table>
<thead>
<tr>
<th>W</th>
<th>95-percentile of C</th>
<th>Avg. V_i</th>
<th>95-percentile of V_i</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5W_a</td>
<td>6.27</td>
<td>0.0868 ±3.1·10^{-4}</td>
<td>0.1667</td>
</tr>
<tr>
<td>W_a</td>
<td>5.78</td>
<td>0.2083 ±2.7·10^{-3}</td>
<td>0.25</td>
</tr>
<tr>
<td>2W_a</td>
<td>6.16</td>
<td>0.1172 ±3.7·10^{-3}</td>
<td>0.25</td>
</tr>
</tbody>
</table>

5.4.4 Heuristic VM Allocation

To evaluate the behavior of the heuristic policies we set: T = 168 h, R_{max} = 0.5 sec, W_a = 1 h, W/W_a = {0.5, 1, 2}, and V_{max} ∈ {0.1, 0.25, 0.35}. We first analyze the sensitivity of the EK and r-l heuristics to the α parameter. Table 5.5 shows the corresponding results. The EK strategy, that assumes to know exactly the workload at time slot i, obtains no violation at the same cost of the r-l policy, because the same values of the α parameter are used to over-estimate the request arrival rate. For r-l, α = 0.1 turns out the best setting because it allows to achieve the lowest cost-performance ratio, i.e., C/(1 − Pr{V > V_{max}}); therefore, we will use it for the comparison.

Table 5.5: r-l heuristic: sensitivity to α

<table>
<thead>
<tr>
<th>Heu.</th>
<th>α</th>
<th>95-percentile of C</th>
<th>SLA satisfaction factor</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>V_{max}=10%</td>
</tr>
<tr>
<td>r-l</td>
<td>0.1</td>
<td>181.4</td>
<td>0.9791</td>
</tr>
<tr>
<td></td>
<td>0.2</td>
<td>196.55</td>
<td>0.962</td>
</tr>
<tr>
<td></td>
<td>0.3</td>
<td>212.47</td>
<td>0.9984</td>
</tr>
<tr>
<td>EK</td>
<td>0.1</td>
<td>181.31</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0.2</td>
<td>196.51</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0.3</td>
<td>212.34</td>
<td>1</td>
</tr>
</tbody>
</table>
Chapter 5. Single Provider VM Allocation

Second, we evaluate the sensitivity of the \( pN-95 \) and \( pN-99 \) heuristics to the SLA parameter \( W \). The corresponding results are shown in Tab. 5.6 where we set \( N = 12 \) corresponding to a prediction period of 1 hour. Setting \( W/W_a > 1 \) results in a significant increase in the fraction of SLA violations. For \( W/W_a \leq 1 \) the fraction of SLA violation is below 2\% and there are no significant differences in term of allocation cost. Therefore, in the remaining experiments we set \( W/W_a = 1 \) to have a fair comparison with the optimal allocation policy.

Table 5.6: Proactive allocation: sensitivity to \( W \) (\( W_a = 1 \) h, \( V_{max} = 25\% \))

<table>
<thead>
<tr>
<th>( W )</th>
<th>Heuristic</th>
<th>95-percentile of ( C )</th>
<th>SLA satisfaction factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 0.5W_a )</td>
<td>( pN-95 )</td>
<td>218.5</td>
<td>0.9854</td>
</tr>
<tr>
<td></td>
<td>( pN-99 )</td>
<td>238.9</td>
<td>0.9886</td>
</tr>
<tr>
<td>( W_a )</td>
<td>( pN-95 )</td>
<td>228.3</td>
<td>0.9819</td>
</tr>
<tr>
<td></td>
<td>( pN-99 )</td>
<td>250.4</td>
<td>0.9858</td>
</tr>
<tr>
<td>( 2W_a )</td>
<td>( pN-95 )</td>
<td>232.2</td>
<td>0.9604</td>
</tr>
<tr>
<td></td>
<td>( pN-99 )</td>
<td>254.6</td>
<td>0.9683</td>
</tr>
</tbody>
</table>

Having chosen the tuning parameters values, we can now compare the different heuristic policies, as shown in Fig. 5.12. The \( EK \) policy, that at the beginning of each time slot knows the average arrival rate for that slot, outperforms all the other heuristics in terms of SLA satisfaction and allocation cost, while the \( r-1 \) is the second best allocation policy. The proactive heuristics (\( p1-95 \), \( p1-99 \), \( pN-95 \), and \( pN-99 \)) suffer in predicting both the arrival and termination of bursts. When the workload is characterized by intense fluctuations, the slow convergence of the prediction algorithm first results in SLO violations (when the burst of requests arrives) and then in resource over-provisioning (when the burst terminates). This behavior determines a lower SLA satisfaction factor and a higher allocation cost compared to the reactive heuristic policy.
5.4. VM Allocation Experimental Evaluation

![Graphs showing SLA satisfaction factor and allocation cost](image)

Figure 5.12: Comparison of heuristic allocation policies: (a) SLA satisfaction factor and (b) 95-percentile of VM allocation cost, for different values of $V_{max}$ (main parameters settings: $V_{max} \in \{0.1, 0.25, 0.35\}$, $N = 12$, $W/W_a = 1$, and $\alpha = 0.1$)

5.4.5 Heuristic versus Optimal VM Allocation

In the last set of experiments we compare the heuristic versus the optimal allocation policies. In this evaluation, we reduce the length of the workload in order to be able to compute the optimal solution. Specifically, we focus our attention on the short workload, which lasts only 4 hours and is extracted from the traces used in the previous experiments (see Fig. 5.8(b)). Figure 5.13 shows the results in terms of SLA satisfaction factor and allocation cost. As expected, with the optimal allocation policy ($Opt$) the SLA is never violated and the allocation cost is minimized. Except the suboptimal $EK$ strategy, none of the heuristic policies is capable to provide 100% SLA satisfaction for the most demanding case corresponding to $V_{max} = 10\%$ and only the $p1-95$ and $p1-99$ heuristics are able to satisfy a more relaxed SLA, corresponding to $V_{max} \geq 25\%$.

The lesson learned from the last set of experiments is that the performance of the
Chapter 5. Single Provider VM Allocation

proposed heuristic policies is comparable in term of SLA satisfaction, but not with respect to the allocation cost. The right choice of the VM allocation strategy will strictly depend on the balance between the penalty the ASP should pay in case of SLA violation and the cost to avoid SLA violations.

![Graph showing SLA satisfaction factor and allocation cost](image)

(a) SLA satisfaction factor  
(b) Allocation cost

Figure 5.13: Optimal versus heuristic VM allocation (main parameters settings: \(V_{max} \in \{0.1, 0.25, 0.35\}\), \(N = 12\), \(T = 4\) h, \(W/W_a = 1\), \(\alpha = 0.1\), and \(K = 1\))

5.5 Prototype Implementation and Experimental Evaluation

In this section a prototype of an autonomic service level provisioning system for the single provider scenario is presented. The goal is to evaluate the advantages and drawbacks of implementing and running the Partial ASP control architecture (equipped with the \(r-I\) allocation algorithm) versus the Limited ASP control architecture. The evaluation has been done instrumenting the systems with the allocation policies described in section 5.5.2 and comparing the system capability to satisfy a given Service Level Objective, the system responsiveness, and the allocation costs of the implemented policies.
As responsiveness we define the capability of a system to promptly react to unexpected load variations. We do not define any specific metric for the responsiveness but it is estimated through the analysis of the the response time and allocated VMs time series. Metrics are collected using Amazon CloudWatch [32], a web service that allows to collect, analyze and view system and application metrics.

The metrics used for comparison are the following:

- **Latency (or Response time)**, collected by the CloudWatch monitor at the load balancer. The response time is collected (and averaged) every minute. We used both the time series and the empirical CDF to compare performance and to give indications on system responsiveness.

- **Number of Allocated VMs**, evaluated by CloudWatch. We consider both the time series and the total value. We remarks that the number of allocated VMs is a cumulative value that does not take into account the identity of the allocated instances.

- **VMs Allocation Cost**, evaluated analyzing the log of VMs allocation and deallocation actions. In the specific, we compute the total and hourly cost. We considered only variable costs, determined by the VMs we run. The cost for all the other resources and services (CloudWatch, Elastic Load Balancer, back-end server and Database) has not be considered since it is the same for both the considered implementations and it is constant over all the experiments.

Both system implementations are evaluated under the same load conditions (see Sec. [5.5.3]), that are characterized by:

- a smooth variation of the workload intensity, and
• burst of requests of unpredictable and heavy intensity (unpredictability has been modeled alternating burst of requests with periods of low intensity).

The fairness of our comparison is given by the fact that both architecture are implemented using the Amazon EC2 infrastructure and that we made an high number of run to hide the unpredictability and unrepeatability of a real environment.

From the experimental results emerged that (see Sec. 5.5.5):

• Having full control over the resource management plan have its advantages in term of achievable performances and reduced resource utilization costs.

• The proposed allocation policy, r-1, outperforms the threshold based policies implemented using the Auto Scaling service.

• Latency and responsiveness benefit of short performance and workload evaluation periods if the allocation/deallocation decision is taken periodically rather than triggered by events.

5.5.1 Partial ASP Control Implementation

In this section we describe how we implemented the Partial ASP control architecture using Amazon EC2 services. In particular, the Limited ASP control implementation, that we take as reference point in the remaining of the chapter, uses the Amazon Auto Scaling service [110].

Single VMs are EC2 on-demand instances of the same size (i.e., EC2 m1.small instances) placed behind an Elastic Load Balancer in one or more availability zones inside the same EC2 Region (at the moment, Amazon Load Balancers cannot span multiple EC2 Regions). The target web application is completely replicated on every
5.5. Prototype Implementation and Experimental Evaluation

VM and data are centralized. This simplifying assumption allows us to consider the application as a whole without taking account of interactions between its layers (otherwise resource replication and distribution have to be considered for every layer) and data synchronization issues.

Incoming requests are managed by the Elastic Load Balancer that distributes them among active VMs using the Least Loaded policy. The Performance Monitoring Service is realized through Amazon CloudWatch, that allows to collect system and application metrics and to aggregate both VM and LoadBalancer metrics for a period that can range from one minute to two weeks.

The Performance Monitor collects, using CloudWatch API GetMetricStatistics, the Average Response Time seen by the Load Balancer (i.e., the average value of the Latency metric) every minute.

In the same way the Workload Monitor gets from CloudWatch the Total number of incoming requests processed by the LoadBalancer (i.e., the sum of the RequestCount metric). Moreover, the Workload Monitor uses the collected information to forecast the arrival rate in the next hour. The forecast, depending on the prediction method used, may be performed directly by a custom software module in the Performance Monitor or by an dedicated external component. We implemented the second solution using Matlab®.

The SLA Analyzer collects observed and predicted metric values from the monitors and triggers the Provisioning Manager at fixed time intervals (e.g., every 5 minutes).

The Provisioning Manager is instrumented with the performance model of the system, that is a M/M/1 queueing network (as described in section 5.1). When the provisioning manager is triggered by the SLA Analyzer it solves the model and determines
Chapter 5. Single Provider VM Allocation

the number of VMs that have to be allocated in the short-medium term (from 1-5 minutes to one hour) to guarantee SLAs.

The various components of the Autonomic Architecture we implemented (Performance and Workload Monitor, SLA Analyzer and Provisioning Manager) have been realized as Java modules running under JRE 1.6 and using AWS SDK for Java 1.1.8 APIs to interact with EC2 services. We did not test the scalability of the autonomic architecture itself because, even if it is an important issue, it does not represent an immediate limitation. While the SLA analyzer and the provisioning manager work on a time scale of minutes, the performance monitor and workload monitor work on the scale of seconds. Even if in the workload scenarios considered these modules are not the system bottleneck, an optimized and scalable implementation should be investigated in the future.

Once the Provisioning manager determines the allocation/deallocation actions that should be taken, it invokes the VM Allocation Service to perform them using LaunchInstances and TerminateInstances APIs.

Once allocated/deallocated, the VM instances are added to/removed from the Load Balancer using RegisterInstancesWithLoadBalancer and DeregisterInstancesFromLoadBalancer APIs.

5.5.2 Limited ASP Control Autoscaling Policies

In this section will be presented the implementation of the limited ASP control solution using the Amazon Auto Scaling service [110] and the autoscaling policies adopted. Amazon Auto Scaling offers the possibility to define allocation/deallocation strategies based on CloudWatch metrics values. In particular, Amazon provide mechanisms to
Set-up alarms on every CloudWatch metric and to decide what action to take when the threshold specified by an alarm is violated. We use these mechanisms defining the threshold values, determined on the basis of a tuning campaign, and what adaptation action to take (i.e. how many VMs have to be allocated/deallocated) when a threshold violation is detected.

In the specific we set up the following four policies:

- **Utilization-based, One alarm** (UT-1Al). This policy scales up and down VMs using a threshold on the the value of the average CPU utilization of all the VMs registered within the load balancer. In the specific, if the utilization exceeds $U_{max}^1$ an instance is allocated while, if it falls under $U_{min}^1$ an instance is deallocated.

- **Utilization-based, Two alarms** (UT-2Al) This policy uses a double threshold on average CPU utilization to scale up/down. If the utilization is greater than $U_{max}^1$ a single instance is launched while, if exceeds the second threshold of $U_{max}^2$ two instances are launched. In a similar way one VM is deallocated if utilization goes below $U_{min}^1$ and two VM are deallocated if the utilization value is less than $U_{min}^2$.

- **Latency-based, One alarm** (LAT-1Al) This policy is similar to the UT-1Al described above except for the fact that, for scaling up, a threshold on the average latency is set. In particular, a new EC2 instance is launched if the average latency is greater than $L_{max}^1$. Since the latency value is not appropriate to determine a deallocation threshold (it never falls under a minimum value of about 0.08 seconds), to scale down the threshold on the minimum utilization $U_{min}^1$ is still used.
• **Latency-based, Two alarms (LAT-2Al)** This policy is similar to the UT-2Al but uses two thresholds on the maximum latency value to scale up. A VM is allocated if the average latency is greater than $L_{max}^1$, two if it is greater than $L_{max}^2$. The deallocation policy is the same as in UT-2Al.

In the experiments we used the following values for the autoscaling thresholds:

- Maximum Utilization thresholds: $U_{max}^1 = 62\%$; $U_{max}^2 = 70\%$;
- Minimum Utilization thresholds: $U_{min}^1 = 50\%$; $U_{min}^2 = 25\%$;
- Maximum Latency thresholds: $L_{max}^1 = 0.2$ sec; $L_{max}^2 = 0.5$ sec;

In all the policies we defined, the possibility to take an adaptation action is periodically evaluated. The evaluation periods we consider are equal to 1 and 5 minutes. For the threshold based policies we used a cool down interval (i.e., minimum time interval between two consecutive adaptation actions) equal to the evaluation period.

### 5.5.3 Workload generation

Right now, benchmarking cloud services is an open issue and no standard solutions, neither largely accepted solutions are available. One of the main reason is that the the benchmark depends on the type of target service (e.g., IaaS, PaaS, SaaS) and on the goal of the evaluation. For example, CloudHarmony [37] provides benchmarking services to test performance (CPU, I/O, Applications), network and uptime of public IaaS providers. Of our interest is a benchmark capable to reproduce the behavior of a typical Web 2.0 application and to test the elasticity, responsiveness and performance of an autonomic cloud manager. At the best of our knowledge, CloudStone [47] as been the first Web 2.0 benchmark for cloud systems. CloudStone is a toolkit consisting of an
5.5. Prototype Implementation and Experimental Evaluation

open-source Web 2.0 social calendar application (Olio [111]) and a set of automation tools, based on Faban [112], for generating load and measuring its performance in different deployment environments. Today, the benchmark is not more maintained and we experienced that the current available version suffers from various implementation issues. Another useful benchmark for distributed large scale applications is WikiBench [54], a Web hosting benchmark allowing to stress-test systems designed to host Web applications, from single components (e.g., application servers, database, load balancers) to Cloud computing platforms. WikiBench uses MediaWiki (the application used to host wikipedia.org) as stress test target application, manipulates real data (actual database dumps of the Wikipedia website), and generates real traffic by replaying real Wikipedia traces. However, WikiBench is not completely useful for the purpose of our performance analysis because the Wikipedia traces [53] do not represent a very bursty workload. Wikipedia workload intensity varies in a very periodical way and, usually, increases not more than of 100% in a 12 hours period. For this reason we think that WikiBench it is not appropriate to evaluate elasticity, stability and responsiveness of a cloud application subject to a very bursty traffic.

Therefore, we decided to create our own workload generator assembling: MediaWiki as the Web 2.0 target application, scaled Wikipedia traces to generate the requests [53], and httperf [113] to control the request generation rate and statistics collection. In this way we have been able to generate not only traffic with a smooth and periodical variation of intensity (smooth ramp workload) but also controlled bursts of requests ranging from the 33% to the 300% of the base traffic (bursty workload), as shown in Figure 5.14.
Figure 5.14: Two examples of stress-test workload generated using MediaWiki and httperf. On the left the smooth ramp workload and on the right the bursty workload.

5.5.4 Testbed setup

As previously introduced we implemented our testbed by means of the Amazon EC2 infrastructure (see Fig. 5.15). In the specific we used:

Figure 5.15: Architecture of the implemented testbed
5.5. Prototype Implementation and Experimental Evaluation

- From 1 to 10 Amazon EC2 m1.small instances (Linux 32 bit machines having one virtual core with 1 EC2 Compute Unit and 1.7 GB of memory). Each VM runs, as application server, Apache 2.2.16 with PHP module installed (PHP version 5.3.6). Each VM replicates and executes the front-end part of the MediaWiki web application (version 1.16.4). The MediaWiki back-end database is centralized and runs on a dedicated node.

- One Amazon EC2 m1.large instance (a Linux 64 bit machine with 2 virtual cores with 2 EC2 Compute Units each, 7.5 GB of memory and MySQL 5.1.52 DBMS) implementing the database server that runs the MediaWiki back-end. This solution avoid multi-tier load balancing and data consistency problems.

- The Amazon Elastic Load Balancer to distribute incoming traffic among the active VMs.

- One EC2 m1.small instance, located in the same availability zone of the application and back-end servers, to run the workload generator.

- One EC2 m1.small instance to run the components of the Partial ASP control architecture we implemented (Performance and Workload Monitor, SLA Analyzer and Provisioning Manager).

The VMs, the database, the load balancer and the workload generators run all in the same availability zone (us-east-1a). Placing the workload generators and the servers in the same availability zone we tried to isolate the components of the total response time due to queuing and computation reducing to the minimum the effects of network latency.
Once installed the MediaWiki servers, the DB has been populated with the Wikipedia database dump used in Wikibench (snapshot of January, 3 2008) that contains about 7 millions of wiki pages.

As previously introduced, we generated the workload starting from real traces. We used a sample (10\%) of the real Wikipedia access traces during the period from September 17th to October 1st, 2007. The trace contains only requests for the English version of Wikipedia web pages. (The access log has been pruned using the Tracebench tool provided with WikiBench). Finally, we generated the desired load intensity through Httperf, a workload generator that iterates requests from a list of URLs addresses from a real trace at the rate specified by the user.

To evaluate the capacity of a single VM and the workload intensity it is capable to handle in normal load conditions we proceeded as follows.

First, empirically, we evaluated that an EC2 m1.small serves four requests per second with an utilization of about 62\%. These results has been obtained averaging the results of multiple runs of the same workload at different hours of the day and day of the week. This evaluation has been necessary because we observed that the performance isolation for the VMs is not complete and that the service rate of the single VM is influenced by the utilization level of the physical host it runs on and by live VM migrations (factors that are not under the control of the Amazon EC2 infrastructure user).

Second, considering a simple M/G/1 model and that the system is stable ($\rho = \lambda/\mu < 1$) we evaluated the service rate of a single VM as 6.45 requests per second.

Finally, supposing the ASP has to guarantee a maximum response time (SLO) of 0.5 seconds, we were capable to determine the minimum number of VMs needed to
5.5. Prototype Implementation and Experimental Evaluation

handle a given load and to satisfy the SLO. The minimum number of required VMs is evaluated using Equation 5.3.

Moreover, we evaluated empirically that the EC2 m1.large instance used to host the MediaWiki database is capable to guarantee that, for the level of traffic we submit to it (up to 40 req/s), the back-end server never represents the system bottleneck.

5.5.5 Performance and Responsiveness analysis

The first set of experiments shows the behavior of the $r - 1$ allocation policy, the UT-1Al policy and the LAT-1Al policy for the smooth ramp workload (Fig. 5.14) when the allocation/deallocation action is taken every minute. We chose these three policies as representative of the whole set we consider, because we observed that the double threshold based policies have essentially the same behavior of the single threshold in the case of smooth ramp workload.

Figure 5.16 shows the allocated VMs and the response time. Comparing the trend of the workload (request per seconds) and the trend of allocated VMs, is clear that both Partial ASP control and Limited ASP control are capable to adapt the system capacity according to the workload fluctuations. In the ramp up phase of the workload (0-70 minutes), the latency measured for the LAT-1Al algorithm is higher than the latency measured using the r-1, therefore the LAT-1Al policy pay the use of less VMs. The strange behavior at the beginning of the experiment (5-20 min) is due to an apparent inefficiency of the load balancer to manage the incoming workload when only two VM are allocated. Indeed, after the new VM are allocated, the latency time goes down.

Moreover, in this scenario all the policies are capable to maintain the response time below 0.5 seconds, the specified SLO. The $r - 1$ VM allocation profile is different from
Chapter 5. Single Provider VM Allocation

the others because VMs are deallocated, if needed, only when the 1 hour billing period is expiring. This allows to have a more stable response time in the ramp up phase of the workload (from 0 to 70 minutes). The only exception, common also to the other policies, is the spike of the response time at time 20, when 2 VMs are not enough to manage 10 requests per second. The allocation policy reacts after a couple of minutes and therefore the system is not capable to guarantee the proper service rate. After a new VMs is allocated the response time jump back to lower values.

Therefore, in the worst case, the responsiveness of the allocation policies can be quantified in 2 minutes given by the time to recognize a change in the workload intensity or system state, and the time to actuate the allocation of new VMs.

![Figure 5.16: The allocated VMs time series (left) and the Latency time series (right) for the smooth ramp workload](image)

Table 5.7 compares the allocation costs. The total hours column shows the total number of hours billed to the ASP. \( r^L \), as expected, outperforms the threshold based policies. The difference of the total cost over a three hour experiment, with a peak usage of 9 VMs, is negligible but, saving about the 15% of the cost is, of course, significant over a longer utilization period and for an higher number of VMs.

Table 5.7 compares the allocation costs. The total hours column shows the total number of hours billed to the ASP. \( r^L \), as expected, outperforms the threshold based policies. The difference of the total cost over a three hour experiment, with a peak usage of 9 VMs, is negligible but, saving about the 15% of the cost is, of course, significant over a longer utilization period and for an higher number of VMs.
5.5. Prototype Implementation and Experimental Evaluation

Table 5.7: Allocation cost for the smooth ramp workload

<table>
<thead>
<tr>
<th></th>
<th>Total hours</th>
<th>Total Cost</th>
<th>Cost/hour</th>
</tr>
</thead>
<tbody>
<tr>
<td>r-1</td>
<td>12</td>
<td>1.02</td>
<td>0.408</td>
</tr>
<tr>
<td>UT-1Al</td>
<td>17</td>
<td>1.445</td>
<td>0.578</td>
</tr>
<tr>
<td>LAT-1Al</td>
<td>16</td>
<td>1.36</td>
<td>0.544</td>
</tr>
</tbody>
</table>

In the second set of experiments we compared all the allocation policies under the bursty workload (see Fig. 5.14). We considered two cases, namely 1-minute and 5-minutes differing for the duration of the allocation/deallocation decision interval.

Results for the first case are shown in Figures 5.17 and 5.18. Comparing Figures 5.14 and 5.17 (allocated VMs) is evident that also in this case the system (both architectures) is capable to self-adapt its configuration to contrast the workload fluctuation. However the higher intensity of the workload impact the system responsiveness that ranges from 2 to 10 minutes when the burst exceeds the 130% of base workload (i.e., starting from time \( t = 70 \) minutes). The empirical CDF plot shows that r-1 is capable to satisfy the SLO in more than 95% of the observations. The UT-1Al policy follows with a SLO satisfaction in more than the 87% of the cases. All the other policies offer worse performances (from 79% to 82% of SLO satisfaction).

The allocation cost is reported in Table 5.8. The r-1 allocation strategy allows to save about the 32% versus the UT-2Al, LAT-2Al and LAT-1Al policies, and about the 17% compared to the UT-1Al policy.

When allocation/deallocation decisions are taken every 5 minutes the system is more stable (i.e., VMs are not continuously added and removed), as shown by the time series of VM allocation (see Fig. 5.19) and by the lower allocation cost (see Tab. 5.9).
Chapter 5. Single Provider VM Allocation

Figure 5.17: The allocated VMs time series (left) and the Latency time series (right) for the bursty workload (1 minute evaluation period)

However, for the threshold based policies, the lower allocation cost impacts negatively on the system latency, responsiveness and, therefore, capability to satisfy the SLO (see Fig. 5.20). Only the $r-1$ ($a = 0.1$) policy performs better than in the 1-minute case, with a probability to satisfy the SLO equal to 1. The $r-1$ ($a = 0.05$) had almost the same behavior. All the other policies are characterized by the probability of the system
Table 5.8: Allocation cost for the bursty workload (1-minute case)

<table>
<thead>
<tr>
<th></th>
<th>Total hours</th>
<th>Total Cost</th>
<th>Cost/hour</th>
</tr>
</thead>
<tbody>
<tr>
<td>r-1, a=0.1</td>
<td>19</td>
<td>1.615</td>
<td>0.538</td>
</tr>
<tr>
<td>r-1, a=0.05</td>
<td>19</td>
<td>1.615</td>
<td>0.538</td>
</tr>
<tr>
<td>UT-2Al</td>
<td>28</td>
<td>2.38</td>
<td>0.793</td>
</tr>
<tr>
<td>UT-1Al</td>
<td>23</td>
<td>1.955</td>
<td>0.651</td>
</tr>
<tr>
<td>LAT-2Al</td>
<td>29</td>
<td>2.465</td>
<td>0.821</td>
</tr>
<tr>
<td>LAT-1Al</td>
<td>27</td>
<td>2.295</td>
<td>0.765</td>
</tr>
</tbody>
</table>

Table 5.9: Allocation cost for the bursty workload (5 minutes evaluation time)

<table>
<thead>
<tr>
<th></th>
<th>Total hours</th>
<th>Total Cost</th>
<th>Cost/hour</th>
</tr>
</thead>
<tbody>
<tr>
<td>r-1, a=0.1</td>
<td>18</td>
<td>1.53</td>
<td>0.51</td>
</tr>
<tr>
<td>r-1, a=0.05</td>
<td>17</td>
<td>1.445</td>
<td>0.481</td>
</tr>
<tr>
<td>UT-2Al</td>
<td>17</td>
<td>1.445</td>
<td>0.481</td>
</tr>
<tr>
<td>UT-1Al</td>
<td>18</td>
<td>1.955</td>
<td>0.51</td>
</tr>
<tr>
<td>LAT-2Al</td>
<td>15</td>
<td>1.275</td>
<td>0.425</td>
</tr>
<tr>
<td>LAT-1Al</td>
<td>14</td>
<td>1.19</td>
<td>0.39</td>
</tr>
</tbody>
</table>

to satisfy the SLO ranging from 0.7 to 0.83 (see Fig. 5.20), while in the 1-minute case the same probability was greater than 0.79.

Moreover, observing Latency CDF (Fig. 5.20), it is immediate to note that the distribution of the response time has a longer queue than in the 1-minute case (Fig. 5.18). This means that the system did not react quickly to workload fluctuations and, when a burst of request arrived, ASP users experimented a higher response time for a longer time period.

Finally, we remark that the allocation cost decreases for all the threshold based allocation policies and, in the specific, the Latency based allocation policies allow to
save about the 20% of the total allocation cost (compared to the other strategies).

Figure 5.19: The allocated VMs time series (left) and the Latency time series (right) for the bursty workload (5 minutes evaluation period)

Figure 5.20: The empirical CDF of the Latency for the bursty workload (5 minutes evaluation time period). On the left is shown the whole plot and on the right a zoom for Latency values lower than 1 second
5.6 Annex: Recursive Least Square Based Prediction

5.6.1 Autoregressive Process Prediction

An autoregressive (AR) process \( x \) of order \( m \) is a stationary Gaussian process which takes the form

\[
x(n) = \theta_0 + \theta_1 x(n-1) + \ldots + \theta_m x(n-m) + a(n)
\]  \hspace{1cm} (5.16)

where \( \theta = (\theta_0, \theta_1, \ldots, \theta_m) \) is a set of weights and \( a \) is a white noise Gaussian process with zero mean and variance \( \sigma_a^2 \).

The prediction for AR processes is straightforward. The “best” predictors (in least mean square error sense) of the future values \( x(n+1), x(n+2), \ldots, x(n+L) \), given the past \( x(n), x(n-1), \ldots \) are obtained by setting to zero the future values of the white noise \( a \). Thus, the best predictor \( \hat{x}(n+j) \) of \( x(n+j) \) can be recursively computed for \( j = 1, \ldots, L \), as follows:

\[
\hat{x}(n+j) = \theta_0 + \theta_1 \hat{x}(n+j-1) + \theta_2 \hat{x}(n+j-2) + \ldots + \theta_m \hat{x}(n+j-m)
\]  \hspace{1cm} (5.17)

where we set \( \hat{x}(k) = x(k) \) for \( k \leq n \).

Let \( e(l) = \hat{x}(n+l) - x(n+l) \) denote the \( l \)-ahead forecast error. \( e(l) \) captures how \( x(n+l) \) deviates from the predicted value \( \hat{x}(n+l) \). \( e(l) \) is normally distributed with zero mean, and its variance is \( \sigma^2_{e(l)} = \sigma^2_a \cdot (\sum_{i=0}^{l} \psi^2(i)) \leq \sigma^2_x \), where \( \psi(.) \) is the impulse response of the Infinite Impulse Response filter with parameter \( \theta \). The error variance increases with \( l \) and converges to the variance of the process \( \sigma^2_x \) as \( l \) grows to infinity.

It is important to observe that the equations above are nothing but determining the conditional distribution of the future values of the process \( x \), given knowledge of the past values. Thus, in the following, to stress that the predicted values \( \hat{x}(n+l) \) and...

\footnote{In other words, \( \psi(0), \psi(1), \ldots \) can be computed via \( \text{(5.16)} \) using the “impulse” input: \( a(0) = 1 \) and \( a(k) = 0, k > 0 \).}
Chapter 5. Single Provider VM Allocation

the error variance $\sigma^2_{e(l)}$ are the mean and the variance of the (Gaussian) conditional
distribution of the future values, we will denote them by $\mu(l)$ and $\sigma^2(l)$, respectively.
We will also denote as $\hat{x}(l) \sim \mathcal{N}(\mu(l), \sigma^2(l))$ a Gaussian random variable with mean
$\mu(l)$ and variance $\sigma^2(l)$.

5.6.2 Recursive Least Square Process Prediction

In this paper, we use the Recursive Least Square (RLS) based process prediction. The
idea behind the RLS-based prediction corresponds to: 1) model a process as a (time
varying parameter) AR process; 2) use this model to predict future behavior; and 3)
adopt a recursive form for the estimation of the model parameters to reduce the com-
putational complexity.

To model the user requests we need to: 1) choose the model order $m$; 2) estimate
the unknown parameter $\theta$ and the variance $\sigma^2_a$ of the white noise. The RLS approach
proceeds recursively as follows. Hereafter, for simplicity, we will assume the model
order $m$ as given.

Let $x(n) = (x(n-1), \ldots, x(n-m+1))$ be the most recent $m$ observed values
of the process (excluding the current value $x(n)$), and $\theta(n) = (\theta_1(n), \ldots, \theta_m(n))$ the
current estimate of $\theta$. The RLS estimation of $\theta$ is then recursively expressed as:

$$k(n) = \frac{\lambda^{-1}P(n-1)x(n)}{1 + \lambda^{-1}x^TP(n-1)x(n)} \quad (5.18)$$

$$\theta(n) = \theta(n-1) + k(n)(x(n) - \theta(n-1)x(n)) \quad (5.19)$$

$$P = \lambda^{-1}P(n-1) - \lambda^{-1}k(n)x^T(n)P(n-1) \quad (5.20)$$

where $\lambda$ is a forgetting factor, $P(n)$ denotes the inverse of the input correlation matrix,
and $k(n)$ is a gain vector.
5.6. Annex: Recursive Least Square Based Prediction

Given \( \theta(n) \), the estimation of the future values of the process \( x(n+1), x(n+2), \ldots \), is carried out as in (5.17) with \( \theta \) replaced by \( \theta(n) \).

For \( l = 1, 2, \ldots \), the variance estimation \( \hat{\sigma}^2_{e(l)}(n) \) of the error is

\[
\hat{\sigma}^2_{e(l)}(n) = \sum_{k=0}^{l} \psi^2(k; n) \hat{\sigma}^2_a(n)
\]

(5.21)

where \( \psi(\cdot; n) \) is the impulse response of the filter \( \theta(n) \), and

\[
\hat{\sigma}^2_a(n) = \frac{1}{n - 2m - 1} \sum_{j=m+1}^{n} e^2(j)
\]

is the current estimation of the white noise process variance.

As before, we observe that the predicted values and the error variances characterize the conditional distribution of the future values of the process. We remark that these conditional distributions converge to the actual conditional distribution only in the case of Gaussian processes. In all other cases, these conditional distributions are only approximations.
Inter-Cloud VM Allocation

In this chapter the problem of resource outsourcing in a cloud federation is modeled. After the definition of a SLA that considers availability, performance, security and cost, a formulation as an optimization problem that considers network latency and multiple service classes is proposed and evaluated.

Contents

6.1 Cloud Federation Outsourcing Model ........................................ 125
   6.1.1 Model ........................................................................... 127
   6.1.2 Problem formulation ....................................................... 130

6.2 Experimental Evaluation .......................................................... 132
   6.2.1 Assumptions and experimental settings ............................. 132
   6.2.2 Metrics ........................................................................ 133
   6.2.3 Evaluation .................................................................... 134

6.1 Cloud Federation Outsourcing Model

As introduced in section 4.4, we consider the perspective of a service provider ($SP_0$ hereafter) joining a cloud federation. $SP_0$ is capable to allocate resources (i.e., VMs) in all the federation providers ($\{SP_1, SP_m\}$) and to distribute incoming load among all the allocated VMs inside the federation. In this scenario, illustrated in fig 6.1, both providers and customers are partitioned into geographical zones. We suppose that
the latency experienced by users depends on the zone where the provider serving the request resides (i.e., depends on the “distance” from the provider).

$SP_0$ is equipped with a cloud manager component (see section 4.4.1) that is capable to allocate resources and distribute the incoming traffic inside the federation. The cloud manager performs multi-cluster dispatching using DNS redirection: (i) clients query the dispatcher for the server IP address; (ii) the dispatcher answers with the IP address of the cluster that will serve the request (i.e., $SP_i$); (iii) the client and the chosen server communicate directly for all the session duration without passing through the dispatcher again. Intra-cluster dispatching is demanded to the IaaS provider.

Figure 6.1: Cloud federation operation scenario.
6.1. Cloud Federation Outsourcing Model

6.1.1 Model

Starting from the cloud federation scenario described above, in what follows we define a general outsourcing model for the provider $SP_0$ joining the cloud federation $\{SP_1, \ldots, SP_M\}$. The outsourcing model is designed to guarantee the SLAs agreed by the customers and the service provider, and to minimize the resource allocation cost for the service provider.

We suppose that $M$ service providers are dispersed among $n$ distinct zones $Z_1, \ldots, Z_n$ ($M \geq n$) and that in zone $Z_j$, $j \in [1, n]$, are located $m_j$ different service providers (i.e., IaaS providers). If a service provider has its infrastructure replicated in many zones, then each replica is considered as a different provider.

The customers of $SP_0$ are partitioned into the same $n$ zones and, moreover, are clustered into $c$ service classes. (Partitioning of clients and SPs should follow the same rules.)

$SP_0$ stipulates with its customers service level agreements based on four attributes: maximum response time, minimum level of availability, minimum level of security, and maximum cost. To guarantee such SLA $SP_0$ must allocate resources to satisfy the following constraints:

- $R^h_{\text{max}}$, the maximum response time accepted by the customers of class $h$;
- $A^h_{\text{min}}$, the minimum level of availability acceptable for class $h$ customers;
- $U^h_{\text{min}}$, the minimum level of security acceptable for the customers of class $h$;
- $C^h_{\text{max}}$, the maximum cost the provider is willing to pay to serve class $h$ users requests.
Chapter 6. Inter-Cloud VM Allocation

We suppose that class $h$ customers from zone $Z_i$ generate a traffic $\Lambda^h_i$. The response time perceived by these customers depends on their service class $h$, on the source zone $Z_i$ and the provider $k$ that serves the request from zone $Z_j$ and on the latency between zone $Z_i$ and zone $Z_j$. Therefore, the average response time $r^h_{ijk}$ is given by

$$r^h_{ijk} = l_{ij} + s^h_{ijk} \quad (6.1)$$

where: $l_{ij}$ is the average latency between zone $Z_i$ and zone $Z_j$; and $s^h_{ijk}$ is the average service time of provider $k$ in zone $Z_j$ to serve a class $h$ request from zone $Z_i$. The average latencies between different zones are specified in a latency matrix $L = \{l_{i,j}\}^{n \times n}$.

Modeling each service provider as a network of $x$ $M/G/1/PS$ queues we have that

$$s^h_{ijk} = \frac{1}{\mu^h_{ijk} - \lambda^h_{ijk}/x^h_{ijk}} \quad (6.2)$$

where: $\lambda^h_{ijk}$ is the rate of class $h$ requests originated from zone $Z_i$ and dispatched to provider $k$ in zone $Z_j$; $x^h_{ijk}$ is the number of VMs allocated on provider $k$ in zone $Z_j$ to serve zone $Z_i$ class $h$ users; and $\mu^h_{ijk}$ is the service rate of VMs. Here we assume that the service rate is independent from the users service class and origin zone, therefore $\mu^h_{ijk} = \mu_{jk} \forall h, i$.

The resource allocation operated by $SP_0$ must be such that $r^h_{ijk} \leq R^h_{max}$, therefore, from equation 6.2 it’s possible to determine $x^h_{ijk,min}$, the minimum number of VMs required to guarantee a response time not greater than $R^h_{max}$:

$$x^h_{ijk,min} = \frac{\lambda^h_{ijk}}{\mu^h_{ijk} - 1/R^h_{max}} \quad (6.3)$$

The availability is defined as the ability of an item (a VM in our case) to be capable to perform a required function at a given instant of time or at any instant of time within
a given time interval, assuming that the external resources, if required, are provided.
We assume that each service provider $k$ in zone $j$ exposes the availability level $a_{jk}$ it can guarantee. The availability level of a service provider is independent from the source zone of requests and from the class of customers requests.

Given the rate $\Lambda^h = \sum_{i=1}^{n} \Lambda^h_i$ of class $h$ requests and the partitioning $\lambda^h_{ijk}$ of that flow among service providers, the average availability perceived by the class $h$ customers is given by

$$A^h = \frac{1}{\Lambda^h} \sum_{jk} \left( a_{jk} \sum_{i} \lambda^h_{ijk} \right)$$  (6.4)

and the traffic partitioning decision operated by $SP_0$ must be such that $A^h \geq A^h_{min}$.

Equation 6.4 and the inequality above introduced do not guarantee that all the class $h$ users will experience an availability greater than $A^h_{min}$. For example, let us suppose class $h$ customers have $A^h_{min} = 0.98$, that are available three servers with availability 0.99, 0.99, 0.97 and that the incoming traffic is partitioned among the servers as follow: 30%, 67% and 3%. Applying equation 6.4 we obtain $A^h = 0.9894 > A^h_{min}$. However, the 3% of customers will experiment an availability that is lower than the one agreed in the SLAs.

Therefore, if $SP_0$ want to guarantee $A^h_{min}$ to all customers of class $h$, it must enforce not only that $A^h \geq A^h_{min}$ but also that $a_{jk} \geq A^h_{min}$, for the selected servers. Otherwise, if $SP_0$ could accept the risk of violating the SLA for some users, the weak constraint can be used.

Concerning security, there is not a standard metric that can be used. Measuring security is a big challenge and typically security metrics depend on the context observed and on the specific goal of the analysis [114]. Since our model is general, we do not consider any specific metric but we propose a meta-metric $U$ that assumes values...
between 0 and 1. For example, $U$ could represent the Attack surface, the Strength of the Authentication method or any other security metric or a combination of them. The proposed formulation is independent from the specific metric. Each service provider $k$ in zone $j$ is characterized by a security level $u_{jk}$ and a customer of class $h$ agrees to experience a minimum security level $U_{h_{min}}$.

Finally, the cost for running a VM for an hour on provider $j$ in zone $k$ to serve requests from user class $h$ in zone $Z_i$ is $c^h_{ijk}$. Similarly with the hypothesis on service rate variability, we also restrict the cost differentiation to service providers, therefore $c^h_{ijk} = c_{jk}$.

### 6.1.2 Problem formulation

We assume that the federation policies allow $SP_0$ to allocate on every provider of the federation a dedicated pool of VMs to serve requests from different geographical zones and different service classes. This assumption is not unrealistic since we can imagine that users from different service classes might have different security, availability and performance requirements and that different types of VM images might be used to satisfy these requirements.

Considering the above described formalization of the provisioning problem, $SP_0$ must determine:

1. how to distribute the incoming traffic from class $h$ and zone $Z_i$ between the available service providers of the federation, i.e., what is the value for $\lambda^h_{ijk}$,

2. how many resources are needed to serve $\lambda^h_{ijk}$ and where these resources have to be allocated, i.e., what is the value for $x^h_{ijk}$.
6.1. Cloud Federation Outsourcing Model

$$\min C(x) = \sum_{i,j,k,h} c_{jk} x^h_{ijk}$$

subject to:

$$\sum_{j,k} \lambda^h_{ijk} = \Lambda^h_i, \ \forall i, h$$ \hfill (6.5)

$$x^h_{ijk} - \lambda^h_{ijk} \frac{1}{\mu_{jk} - 1/(R^h_{max} - l_{ij})} \geq 0, \forall i, j, k, h$$ \hfill (6.6)

$$(c^h_{max} - c_{jk}) x^h_{ijk} \geq 0, \forall i, j, k$$ \hfill (6.7)

$$\sum_{i,h} x^h_{ijk} \leq X_{jk,max}, \forall j, k$$ \hfill (6.8)

$$x^h_{ijk} = 0, \forall i, j, k, h : \mu_{jk} \leq \frac{1}{R^h_{max} - l_{ij}} \text{ or } R^h_{max} \leq l_{ij}$$ \hfill (6.9)

$$(u^h_{jk} - U^h_{min}) x^h_{ijk} \geq 0, \forall i, j, k$$ \hfill (6.10)

$$(a_{jk}^h - A_{min}^h) x^h_{ijk} \geq 0, \forall i, j, k$$ \hfill (6.11)

$$x^h_{ijk} \in \mathbb{N}, \forall i, j, k, h$$ \hfill (6.12)

$$\lambda^h_{ijk} > 0, \forall i, j, k, h : x^h_{ijk} > 0$$ \hfill (6.13)

$$\lambda^h_{ijk} = 0, \forall i, j, k, h : x^h_{ijk} = 0$$ \hfill (6.14)

Figure 6.2: Optimization problem formulation
Decisions 1 and 2 are strictly correlated and have a common goal: to minimize the total cost $C(x)$ satisfying SLAs and serving all the incoming service requests. Decisions 1 and 2 are determined by the solution of the optimization problem shown in figure 6.2. Equation 6.5 assures that all the incoming traffic is routed to the IaaS providers, Eqs. 6.6 and 6.7 guarantee response time and cost constraints satisfaction, Eq. 6.9 assures that no request is routed to providers that cannot satisfy the constraint on the maximum response time, and Eq. 6.8 guarantees that, for every provider, the number of allocated VMs is not greater than the maximum value $X_{jk,max}$. Eq 6.10 is the security level constraint and Eq 6.11 is the availability constraint. Finally, Eq. 6.14 guarantees that no traffic is redirected to a service provider if it has not been selected (i.e., $x_{hijk} = 0$).

6.2 Experimental Evaluation

In this section the experiments conducted for the scenario of VM allocation in a cloud federation are illustrated and discussed. Starting from the scenario and the architecture described in section 4.4 and from the model proposed in section 6.1, different cases are considered to show how the variation of one or more model parameters impacts on allocation cost and traffic distribution.

6.2.1 Assumptions and experimental settings

In this set of experiments we considered a scenario consisting of four distinct geographical zones ($n = 4$). In every zone there are up to three distinct cloud providers ($m = 3$). Users belong to two different service classes ($c = 2$) and are dispersed among the four zones. The generated traffic for class 1 and 2 from zones 1-4, expressed
6.2. Experimental Evaluation

Table 6.1: Service Level Agreements

<table>
<thead>
<tr>
<th>Class</th>
<th>$R_{max}$</th>
<th>$C_{max}$</th>
<th>$A_{min}$</th>
<th>$S_{min}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.5</td>
<td>10</td>
<td>0.950</td>
<td>0.7</td>
</tr>
<tr>
<td>2</td>
<td>1.0</td>
<td>1.5</td>
<td>0.900</td>
<td>0.5</td>
</tr>
</tbody>
</table>

in req/s, is defined respectively in $\Lambda^1 = \{40, 150, 100, 70\}$ and $\Lambda^2 = \{20, 75, 50, 35\}$ (i.e. $\Lambda^h = \{\Lambda^h_i\}_{i=1..n}$).

The SLAs for class 1 and 2 users are shown in Tab. 6.1. We consider that service rates are the same for all the providers (i.e. $\mu_{jk} = 10$ req/s $\forall j, k$). In the considered scenario the value of the maximum number of VMs allocable on a single provider is equal for all the providers ($X_{jk, max} = X_{max} \forall j, k$) and its default value is $X_{max} = 20$.

Latencies between zones are defined in the latency matrix $L$.

$$L = \begin{pmatrix}
0.0 & 0.1 & 0.5 & 0.9 \\
0.1 & 0.0 & 0.1 & 0.2 \\
0.3 & 0.1 & 0.0 & 0.1 \\
0.6 & 0.2 & 0.1 & 0.0
\end{pmatrix}$$

In the considered scenario there are three different providers in zone $Z_1$ and $Z_2$, two providers in zone $Z_3$ and one provider in zone $Z_4$. Hourly VM allocation costs and availability and security levels for the various providers are fixed and their values are shown in table 6.2.

6.2.2 Metrics

The metrics considered in the experiments are:

- Total allocation cost $C$.
- Traffic distribution among the various cloud federation providers $\lambda^h_{ijk}$.
Table 6.2: Service providers cost, availability and security level

<table>
<thead>
<tr>
<th>Zone</th>
<th>( (c_{ij}, a_{ij}, s_{ij}) )</th>
<th>( (c_{ij2}, a_{ij2}, s_{ij2}) )</th>
<th>( (c_{ij3}, a_{ij3}, s_{ij3}) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( Z_1 )</td>
<td>(1.0, 0.950, 0.7)</td>
<td>(1.2, 0.990, 0.8)</td>
<td>(1.4, 0.990, 0.8)</td>
</tr>
<tr>
<td>( Z_2 )</td>
<td>(1.5, 0.990, 0.9)</td>
<td>(1.7, 0.999, 0.9)</td>
<td>(1.9, 0.999, 0.9)</td>
</tr>
<tr>
<td>( Z_3 )</td>
<td>(1.4, 0.990, 0.7)</td>
<td>(1.6, 0.999, 0.8)</td>
<td>-</td>
</tr>
<tr>
<td>( Z_4 )</td>
<td>(1.1, 0.950, 0.7)</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Figure 6.3: Traffic distribution among Zones for the different value of \( X_{max} \)

- Traffic distribution among the various geographical regions for each class: \( \lambda^h = \sum_{i,k} \lambda^h_{ijk} \) and aggregated for all classes: \( \lambda_j = \sum_h \lambda^h_j \).

We do not consider \( x^h_{ijk} \) in the performance metrics because in the considered scenario, being the service rate fixed to \( \mu = 10 \) for all the providers, the number of VMs allocated on the various cloud federation providers is directly proportional to \( \lambda^h_{ijk} \) (see Eq.3).

6.2.3 Evaluation

In this section we present the five types of sensitivity analysis that have been performed:

- sensitivity to \( X_{max} \);
- sensitivity to latency;
6.2. Experimental Evaluation

- sensitivity to the increment of the arrival rate in a single zone;
- sensitivity to availability constraint;
- sensitivity to security constraint.

**Sensitivity to** $X_{\text{max}}$

In the first set of experiments we varied the value of the maximum number of VMs allocable on a single cloud provider from 10 to 40 leaving the other system parameters unchanged.

Fig. 6.3 shows that, for small values of $X_{\text{max}}$, requests are distributed in a quite uniform way among the various zones. This happens because the resources of providers located in the less expensive zones (i.e., $Z_1$ and $Z_4$) are fully utilized and many incoming requests have to be served by the providers in other zones. Traffic routing to different zones is performed mostly for class 2 users that have strict cost constraints, while class 1 user requests are routed to other zones only if there are not enough resources available to serve them in the origin zone. Increasing $X_{\text{max}}$ we see that all the traffic is distributed between zones $Z_1$ and $Z_4$ while no requests is routed to $Z_2$ and $Z_3$. Since the optimization problem objective is the cost minimization this behavior allows to minimize costs routing requests from zones with higher provider prices to near cheaper zones. (We consider a concept of proximity between zones that is based on the latency value. Lower is the latency among two zones and higher is their proximity). Obviously this behavior implies that the total allocation cost $C$ decreases as $X_{\text{max}}$ increases (see Fig. 6.4).
Sensitivity to latency

In this set of experiments we analyzed the system performance sensitivity to variation of latencies between zones. In the first case we consider the latency matrix \( L \) defined above, then we considered the no-latency case (all latency values are set to zero), and two different symmetric latency matrices, \( L_{0.1} \) and \( L_{0.2} \) that have the same latency value for all the zones, i.e., \( l_{i,j} = 0.1 \) (0.2 respectively) \( \forall i \neq j \) and \( l_{i,i} = 0 \) \( \forall i \).

Figure 6.5 shows that, as latency increases, incoming requests are less often routed to other zones even if the providers in these zones offer cheaper services. Obviously, if there is no latency, cheaper service providers are fully utilized (i.e., zone \( Z_1 \) and \( Z_4 \) providers are full) and only exceeding requests are routed to other providers (i.e., zone \( Z_3 \) providers), while providers of the most expensive zone (\( Z_2 \)) are not utilized at all.

Allocation costs, as expected, increase as latencies increase (see Fig. 6.6).
Sensitivity to arrival rate (scalability)

In the third set of experiments we evaluate the scalability property of the model increasing the arrival rate from a specific zone \((Z_1)\) for both the considered user classes leaving arrival rates from the other zones unchanged. \(\Lambda^h_1\) was incremented by 3, 6, 9 and 12 times its default value. From Fig. 6.7 it is possible to see how the traffic from...

Figure 6.5: Traffic distribution \((\lambda_j)\) sensitivity to latency

Figure 6.6: Total allocation cost \(C\) sensitivity to latency
zone $Z_1$ is served by providers in the same zone until they are saturated and then how is gradually split in the other zones. It’s interesting to see that, when the request rate is very high, exceeding requests from zone $Z_1$ are preferably routed to zone $Z_2$ rather than $Z_3$ and $Z_4$ even if the prices in this zone are higher because of the lower latency between $Z_1$ and $Z_2$. Similarly, no requests is routed from $Z_1$ to $Z_4$ because the high latency between these zones would have made impossible to satisfy the SLA constraint on the maximum response time.

![Figure 6.7: Traffic distribution ($\lambda_j$) sensitivity to zone $Z_1$ arrival rate](image)

**Sensitivity to availability constraint**

While in the previous sets of experiments the availability and security constraints were always satisfied ($a_{ij} \geq A^h_{\text{min}} \forall i,j$ and $s_{ij} \geq S^h_{\text{min}} \forall i,j$), in the following experiments we varied these constraints to observe their impact on the considered metrics. Starting from availability, we considered the following constraints: Very Low (VL), that is
$A_{\text{min}} = 0.900$; Low (L) with $A_{\text{min}} = 0.950$; Medium (M) with $A_{\text{min}} = 0.990$; and High(H) that correspond to $A_{\text{min}} = 0.995$. In the five considered cases, the constraints for class 1 and 2 are respectively set to (L, VL), (M, L), (H, L), (M, M), (H, M). From Fig. 6.8 it is possible to observe how, as availability requirements becomes higher, more request are routed from cheaper zones ($Z_1$ and $Z_4$) to zones hosting the providers that guarantees higher availability levels ($Z_2$ and $Z_3$). In particular, in the case (H, M), no request is routed to zone $Z_4$ and only class 2 requests are routed in zone $Z_1$.

![Figure 6.8: Traffic distribution ($\lambda_j$) sensitivity to availability constraint](image)

Using, instead of (6.11), the following weaker availability constraint

$$\sum_{i=1}^{n} \frac{1}{A_i^h} \sum_{ijk} \lambda_{ijk}^h a_{jk} \geq A_{\text{min}}^h \forall h \quad (6.15)$$

that guarantees that the average availability for class $h$ users is equal or greater than $A_{\text{min}}^h$, we observed a similar traffic distribution (see Fig. 6.9) and a lower allocation cost. From Fig. 6.10 we can see that, when a high availability level is not required the
cost is the same in both cases since the minimum availability level is guaranteed by all the providers while, when the requested availability level is medium-high, in the weak constraint case the total allocation cost is about 15% lower.

![Traffic distribution sensitivity to availability constraint](image)

Figure 6.9: Traffic distribution ($\lambda_j$) sensitivity to availability constraint (weak constraint)

**Sensitivity to security constraints**

In the last set of experiments we observed the impact of the security constraints on the performance metrics. As previously done for availability, we considered five constraints combinations: (L,VL), (M,L), (H,L), (M,M), (H,M). In this case the constraints values are: Very Low (VL), i.e. $S_{\text{min}} = 0.5$; Low (L), that is $S_{\text{min}} = 0.7$; Medium (M) with $S_{\text{min}} = 0.8$; and High(H), i.e. $S_{\text{min}} = 0.9$. From the traffic distribution (Fig. 6.11) it is possible to observe that, when the required security level is high, a great number of requests is routed to zone $Z_2$ since only in this zone there are service
providers that guarantee a security level of 0.9. In the case of Medium-Low security level requirements, more requests are routed to cheaper providers in zones $Z_1$, $Z_3$ and $Z_4$. Obviously higher are the security requirements, higher is the allocation cost (see Fig. 6.12).
Figure 6.11: Traffic distribution ($\lambda_j$) sensitivity to security constraint

Figure 6.12: Total allocation cost $C$ sensitivity to security constraints
7 Conclusions

Contents

7.1 Summary .................................................. 143
7.2 Future Research .......................................... 146

7.1 Summary

In this thesis the problem of service level provisioning in cloud systems has been ana-
lyzed and new architectures, models and algorithms to address it have been proposed.

The main issues of service level provisioning and the solutions proposed in litera-
ture to address them have been classified into five dimensions (i.e., where, when, what,
why, and how) and, starting from these dimensions, state of the art solutions have been
organized in a service level provisioning taxonomy (Chapter 3).

The services and features offered by the main IaaS providers have been analyzed in
the perspective of realizing an autonomic system for service level provisioning. From
the IaaS providers features taxonomy illustrated in section 4.2 has been shown that
no efficient ready-to-use solution for service level provisioning in clouds is currently
available. Starting from a general MAPE-K based QoS-aware autonomic architecture
and from the IaaS providers’ useful features identified, four different QoS-aware au-
tonomic architectures have been proposed. The architectures, namely Extreme ASP
Chapter 7. Conclusions

Control, Full ASP Control, Partial ASP Control and Limited ASP Control, differ for the degree of control that the ASP has on the various architectural components of the autonomic system (i.e., for the usage of features and services offered by IaaS providers instead of ad-hoc components). An inter-cloud resource manager has been proposed to extend the autonomic architecture allowing support for resource outsourcing in a cloud federation (Chapter 4).

For what concerns the models and algorithms proposed to perform service level provisioning in clouds, two different scenarios have been considered: (i) the one of an ASP that performs provisioning through allocation/deallocation of VMs from a single IaaS provider; (ii) the scenario of a service provider that has the opportunity to outsource resources inside a cloud federation; in this case, besides allocating/deallocating VMs, the ASP can choose how to distribute the incoming load among available resources inside the federation.

For the VM allocation problem from a single provider a model based on queuing networks and a SLA considering performance requirements (i.e., maximum value of the average response time) have been introduced. In the proposed SLA the maximum number of SLO violations allowed during a time period is contractualized. From the proposed model and SLA, the VM allocation problem has been formulated as a mixed integer linear optimization problem. The proposed formulation has the objective to minimize allocation cost guaranteeing SLAs; to this purpose an objective function that allows to consider the tradeoff between allocation cost and user perceived performance has been introduced. Since the complexity of the proposed problem is exponential in the number of integer variables, the computational cost to solve the problem might be prohibitive for online operation. To this end both reactive and proactive heuristic
7.1. Summary

Policies have been proposed to perform VM allocation. The proactive policies use a Recursive Least Square based prediction algorithm to forecast the future workload level. Simulation experiments have been conducted to evaluate the heuristic policies and compare them to the optimal solution.

A stochastic workload model to generate synthetic workloads reproducing the time dependency and business characteristics of real workloads has been elaborated and used to generate the workload used in the experiments. From the experimental evaluation it can be observed that the better performing heuristics are capable to guarantee a SLA satisfaction very close to 100% at a cost about 10%-20% higher than in the optimal case. The proactive policies tend to over-provision resources, allowing to obtain a slightly higher SLA satisfaction than the reactive ones but a considerably higher cost. From this results is evident that the choice of the best VM allocation policy is strictly dependent on the balance between the SLA violation penalty that the service provider should pay and the cost to avoid SLA violations.

Moreover, a prototype implementation of the Partial ASP Control and Limited ASP Control architectures using the Amazon EC2 public cloud has been presented. The experiments, conducted using MediaWiki as the target application and synthetic workloads based on real Wikipedia traces, have been used to compare the proposed reactive heuristics with simple threshold based autoscaling policies. Experimental results show that the heuristic policies are capable to react to sudden workload changes allowing to obtain lower response times at a cost similar or even lower than threshold based policies (Chapter 5).

For the Inter-cloud scenario a model based on queuing networks considering users and providers dispersed in different geographical zones and taking account of network
latencies between zones has been introduced. In this scenario a SLA considering users from different service classes and constraints on performance, availability, security and cost has been proposed. An optimization problem has been formulated to decide how many resources (i.e., VMs) need to be allocated, where to allocate these resources inside the federation, and how to distribute the incoming load among the allocated VMs.

From the conducted experiments it is possible to see how latency is a predominant factor: clients should be served by ”nearest” providers. Moreover, since the objective is cost minimization, traffic is preferably routed to the cheapest providers that allow to guarantee SLA satisfaction. This behavior is influenced by the ratio between the arrival rate and the available resources: lower is the ratio higher is the possibility to distribute traffic between cheaper providers, while higher is the ratio more balanced is the load distribution (at higher costs) (Chapter[6]).

7.2 Future Research

The research work described in this thesis leaves room for further research in many different branches.

**Service Level Provisioning.** To better evaluate the proposed autonomic architectures also the Extreme ASP Control and Total ASP control should be implemented and tested. Moreover, to compare the performance of the various IaaS providers, the proposed architectures might be implemented using the infrastructure and services of different providers.

The service level provisioning approach proposed, although addressing provision-
ing at a single application level, could be extended to consider multi layer and multi component applications. In this case data consistency and replication issues have to be considered and the queueing network model presented needs to be updated to consider multi level provisioning.

Moreover, for the single provider case, the model could be extended to consider different service classes for users and a SLA contemplating constraints on different metrics such as availability and security. To better evaluate the proposed algorithms, also the proactive heuristics should be implemented on the autonomic system prototype. Finally it would be interesting to stress the prototype with different types of workload and for different types of applications and services.

Cloud Federation. Although the research proposed is a first step towards autonomic service level provisioning in a cloud federation scenario, a lot of work has still to be done to achieve effective provisioning in inter-clouds.

The implementation of an architecture that allows an inter-cloud resource manager has many challenging issues. Efficient mechanisms and protocols should be proposed to support interoperability between different heterogeneous providers. To support efficient distributed resource management inside the federation advanced distributed monitoring mechanisms are needed.

Moreover, the definition of a structured and standardized representation of SLAs would be very useful for both users and providers.

Finally, the proposed model could be extended to consider fog computing and edge computing scenarios where data and computing power are brought away from centralized points to the logical extremes of a network.
Chapter 7. Conclusions

**Cloud Benchmarking and Workload Characterization.** To better evaluate the proposed solutions for service level provisioning specific benchmarks should be developed. Moreover, a detailed workload characterization allowing to reproduce workloads suitable for different types of applications would be useful to properly test the efficiency of the service level provisioning approach proposed.
Bibliography


[34] “Open nebula,” [http://opennebula.org](http://opennebula.org)


[70] “Google cloud platform,” cloud.google.com


[87] “Cloudsim: A framework for modeling and simulation of cloud computing infrastructures and services,” www.cloudbus.org/cloudsim


[90] “Microsoft windows azure,” www.windowsazure.com


[92] “Storm on demand,” www.stormondemand.com


[94] “Gogrid,” www.gogrid.com

[95] “At&t synaptic,” www.synaptic.att.com


[97] “dimensiondata,” cloud.dimensiondata.com


[99] “Hp public cloud,” www.hpcloud.com

[100] “Joyent,” www.joyent.com


[103] “Softlayer,” softlayer.com

[104] “Terremark vcloud express,” vcloudexpress.terremark.com


