# A Self-Adaptive Auto-Scaling System in Infrastructure-as-a-Service Layer of Cloud Computing

by

Seyedali Yadavar Nikravesh

A thesis submitted to the Faculty of Graduate and Postdoctoral Affairs in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Computer Engineering

Carleton University
Ottawa, Ontario

## Abstract

The National Institute of Standards and Technology (NIST) defines scalability, resource pooling and broad network access as the main characteristics of cloud computing which provides highly available, reliable, and elastic services to cloud clients. In addition, cloud clients can lease compute and storage resources on a pay-as-you-go basis. The elastic nature of cloud resources together with the cloud's pay-as-you-go pricing model allow the cloud clients to merely pay for the resources they actually use. Although cloud's elasticity and its pricing model are beneficiary in terms of cost, the obligation of maintaining Service Level Agreements (SLAs) with the end users necessitates the cloud clients to deal with a cost/performance trade-off. Auto-scaling systems are developed to balance the trade-off between the cost and the performance by automatically provisioning compute and storage resources for the cloud services.

Rule-based systems are currently the most popular auto-scaling systems in the industrial environments. However, rule-based systems suffer from two main shortcomings: a) reactive nature, and b) the difficulty of configuration. This thesis proposes an auto-scaling system which overcomes the shortcomings of the rule-based systems. The proposed auto-scaling system consists of a *self-adaptive prediction suite* and a *cost driven decision maker*. The prediction suite remedies the first shortcoming (i.e., the reactive nature) of the rule-based systems by forecasting the near future workload of the cloud service. In addition, the cost driven decision maker uses a genetic algorithm to automatically configure the rule-based decision makers. The evaluation results show that the proposed system reduces the total auto-scaling cost up to 25% compared with the Amazon auto-scaling system.

## Acknowledgements

I would like to express my sincere appreciation and thanks to my supervisors Dr. Samuel A. Ajila and Dr. Chung-Horng Lung for their support, patience, motivation, and immense knowledge. This thesis would not have been completed without their guidance and help.

I would also like to thank my family. Words cannot express how grateful I am to my mother, father, brother, and sister for their support and encouragement.

Last but not the least, my deepest appreciation is expressed to my beloved wife Saloumeh for her unconditional love, patience, and understanding.

# Table of Contents

**Abstract** ..... **2**

**Acknowledgements** ..... **3**

**Table of Contents** ..... **4**

**List of Tables** ..... **9**

**List of Illustrations** ..... **12**

**List of Acronyms** ..... **14**

**List of Symbols** ..... **16**

**Chapter 1 - Introduction** ..... **18**

1.1 Research Motivation ..... 18

1.2 Research Scope ..... 19

1.3 Problem Statement ..... 22

1.4 Formal Problem Definition ..... 23

1.5 Research Goal and Objectives ..... 25

1.6 Research Approach ..... 28

1.7 Thesis Contributions ..... 29

1.8 Outline of the Thesis ..... 31

**Chapter 2 - Literature Review** ..... **32**

2.1 Reducing the Virtual Machine Boot-Up Time ..... 32

2.2 Background Concepts ..... 33

2.2.1 Infrastructure-as-a-Service Providers ..... 34

2.2.1.1 Amazon Elastic Compute Cloud (Amazon EC2) ..... 36

2.2.2 Cloud Service Benchmarks ..... 37

2.2.2.1 TPC-W ..... 39

2.2.3 Service Level Agreements ..... 40
2.2.4 Workload ..... 41
2.3 Existing Auto-Scaling Techniques ..... 44
2.3.1 Target Tier ..... 45
2.3.2 Prediction Method ..... 46
2.3.3 Decision Making Approach ..... 47
2.3.4 Performance Indicator ..... 48
2.3.5 Workload Pattern ..... 50
**Chapter 3 - Theoretical Foundation of Machine Learning ..... 51**
3.1 Formal Definition of the Machine Learning Process ..... 52
3.2 Empirical Risk Minimization ..... 54
3.2.1 VC-dimension ..... 56
3.2.2 Uniform Convergence Rates for the ERM ..... 57
3.3 Structural Risk Minimization ..... 58
3.4 Effect of the Workload Pattern on the Prediction Accuracy of the Empirical and the Structural Risk Minimizations ..... 59
3.5 Artificial Neural Networks ..... 62
3.5.1 Multi-Layer Perceptron with Empirical Risk Minimization ..... 62
3.5.2 Multi-Layer Perceptron with Structural Risk Minimization ..... 63
3.6 Support Vector Machine ..... 64
**Chapter 4 - Proposed Predictive Auto-Scaling System ..... 68**
4.1 Self-Adaptive Workload Prediction Suite ..... 69
4.2 Cost Driven Decision Maker ..... 75
4.2.1 The Reason to Choose Genetic Algorithm ..... 79
4.2.2 Genetic Algorithm ..... 80
4.2.2.1 Individual Presentation ..... 81

4.2.2.2 Fitness Function ..... 82
4.2.2.3 Genetic Operators ..... 83
4.2.2.3.1 Selection Operator ..... 84
4.2.2.3.2 Crossover and Mutation Operators ..... 85
4.2.3 The Proposed Decision Maker Algorithm ..... 86
4.3 Evaluation ..... 90
4.3.1 Evaluation of the Cost Driven Decision Maker ..... 90
4.3.2 Evaluation of the Proposed Predictive Auto-Scaling System ..... 93
4.3.2.1 Amazon Auto-Scaling System Details ..... 93
4.3.2.2 Evaluation Results ..... 94
**Chapter 5 - Experiments and Results ..... 98**
5.1 Experiment I: Cloud Resource Auto-Scaling Based on Hidden Markov Model (HMM) ..... 98
5.1.1 Hidden Markov Models (HMM) ..... 98
5.1.2 Experimental Setup ..... 99
5.1.3 Evaluation Metrics ..... 103
5.1.4 Results ..... 104
5.2 Experiment II: The Influence of the Training Duration on the Accuracy of the Prediction Models ..... 106
5.2.1 Experimental Setup ..... 107
5.2.2 Results ..... 108
5.3 Experiment III: The Influence of the Workload Patterns and the Sliding Window Size on the Accuracy of the MLP, MLPWD, and SVM Prediction Models ..... 109
5.3.1 Experimental Setup ..... 110
5.3.2 Results ..... 112

5.4 Experiment IV: The Sensitivity of the Auto-Scaling Decisions to the Prediction Results ..... 118
5.4.1 Experimental Setup ..... 119
5.4.2 Results ..... 122
5.5 Experiment V: The Virtual Machine Boot-Up Time Effect on the Auto-Scaling Cost factors ..... 124
5.5.1 Experimental Setup ..... 124
5.5.2 Results ..... 126
5.6 Experiment VI: The Smart Kill Effect on the Auto-Scaling Cost factors ..... 129
5.7 Experiment VII: The Impact of the Rule-Based Configuration Parameters on the Auto-Scaling Cost Factors ..... 130
5.7.1 The Upper Threshold (thrU) ..... 131
5.7.2 The Lower Threshold (thrL) ..... 132
5.7.3 The Duration Parameters (durU and durL) ..... 134
5.7.4 The Freezing Periods Durations ..... 135
5.7.5 Section Summary ..... 136
5.8 Experiment VIII: Finding the Optimal Values of the Genetic Algorithm for the Auto-Scaling Problem ..... 137
5.8.1 Population Size ..... 137
5.8.2 Stop Condition ..... 139
5.8.3 Crossover Rate ..... 140
5.8.4 Mutation Rate ..... 142
5.9 Experiment IX: Applying the Empirical Risk Minimization (ERM) to Train the Rule-Based Decision Makers ..... 143
5.10 Experiment X: The Influence of the Training Duration on the Accuracy of the Decision Maker ..... 146

5.11 Summary ..... 148

**Chapter 6 - The Impact of the Database Capacity on the Business tier Auto-Scaling ..... 150**

6.1 Analytical Investigation ..... 151

6.2 Experiments and Results ..... 158

**Chapter 7 - Conclusions and Future Work ..... 166**

7.1 Contributions ..... 167

7.2 Threats to Validity ..... 169

7.3 Future Work ..... 170

**Appendix ..... 172**

Auto-Scaling Simulation Package ..... 172

A.1 Simulation Package Design ..... 172

**References ..... 178**

## List of Tables

Table 2-2. QoS attributes for web services ..... 43

Table 2-1. The existing auto-scaling mechanisms ..... 49

Table 3-1. SVM and MLP comparison [69] ..... 67

Table 4-1. The genetic algorithm behavior in an environment with the periodic workload ..... 91

Table 4-2. The genetic algorithm behavior in an environment with the growing workload ..... 92

Table 4-3. The genetic algorithm behavior in an environment with the unpredictable workload ..... 92

Table 5-1. The hardware specification (experiment I) ..... 102

Table 5-2. The HMM parameters and values ..... 102

Table 5-3. The evaluation of HMM (training phase) ..... 105

Table 5-4. The evaluation of HMM (testing phase) ..... 105

Table 5-5. The hardware specification of the virtual machines (experiment II) ..... 108

Table 5-6. The influence of the training duration on the prediction accuracy ..... 108

Table 5-7. MLP and MLPWD configurations ..... 111

Table 5-8. SVM configuration ..... 112

Table 5-9. The MAE and the RMSE values in the training phase (periodic pattern) ..... 112

Table 5-10. MAE and RMSE values in the testing phase (periodic pattern) ..... 112

Table 5-11. The MAE and The RMSE values in the training phase (growing pattern) ..... 115

Table 5-12. The MAE and The RMSE values in the testing phase (growing pattern) ..... 115

Table 5-13. The MAE and the RMSE values in the training phase (unpredictable pattern) ..... 116
Table 5-14. The MAE and the RMSE values in the testing phase (unpredictable pattern) ..... 117
Table 5-15. The identical scaling decisions ..... 122
Table 5-16. The identical scaling decisions for the reasonable threshold values ..... 123
Table 5-17. Simulation parameters and values ..... 125
Table 5-18. The cost factor values for the different VM boot-up times (iteration 1 – no database tier) ..... 127
Table 5-19. The cost factor values for the different VM boot-up times (iteration 2 – with database tier) ..... 127
Table 5-20. The parameters of the rule-based decision maker ..... 129
Table 5-21. The impact of the smart kill on the cost factors ..... 129
Table 5-22. The impact of the *thrU* on the cost factors ..... 132
Table 5-23. Impact of increasing *thrL* on the cost factors ..... 133
Table 5-24. Impact of decreasing *thrL* on the cost factors ..... 133
Table 5-25. Impact of increasing *durU* on the cost factors ..... 134
Table 5-26. The impact of increasing the *durL* on the cost factors ..... 135
Table 5-27. Impact of increasing *inU* on the cost factors ..... 135
Table 5-28. Impact of increasing *inL* on the cost factors ..... 136
Table 5-29. Impact of population size on the genetic algorithm accuracy ..... 138
Table 5-30. The impact of the number of the generations on the genetic algorithm accuracy ..... 140

Table 5-31. The impact of the crossover rate on the genetic algorithm accuracy ..... 141
Table 5-32. The impact of the mutation rate on the genetic algorithm accuracy ..... 143
Table 5-33. The fitness values for the training and the testing datasets (periodic pattern) ..... 144
Table 5-34. The fitness values for the training and the testing datasets (growing pattern) ..... 145
Table 5-35. The fitness values for the training and the testing datasets (unpredictable pattern) ..... 145
Table 5-36. The influence of the training duration on the auto-scaling accuracy (periodic pattern) ..... 147
Table 5-37. The influence of the training duration on the auto-scaling accuracy (growing pattern) ..... 147
Table 5-38. The influence of the training duration on the auto-scaling accuracy (unpredictable pattern) ..... 147
Table 6-1. The experiment parameters and their values ..... 159
Table 6-2. The resource cost and the SLA violations (periodic pattern) ..... 164
Table 6-3. The resource cost and the SLA violations (growing pattern) ..... 164
Table 6-4. The resource cost and the SLA violations (unpredictable pattern) ..... 164

## List of Illustrations

Figure 1-1. The IaaS environment stakeholders and their relationships ..... 21

Figure 1-2. The architectural overview of the predictive auto-scaling systems [6] ..... 26

Figure 1-3. Quantitative research process [8] ..... 28

Figure 2-1. The effect of the monitoring interval on the under-provisioning condition ..... 32

Figure 2-2. The IaaS environment components ..... 34

Figure 2-3. The customized script to generate the different workload patterns ..... 40

Figure 4-1. Architecture of the proposed predictive auto-scaling system ..... 68

Figure 4-2. A classical autonomic system ..... 70

Figure 4-3. The cloud workload prediction autonomic system ..... 71

Figure 4-4. Mapping the classical autonomic system to the cloud workload prediction autonomic system ..... 72

Figure 4-5. Projection of the cloud auto-scaling autonomic element ..... 73

Figure 4-6. The design of the autonomic elements (the cloud workload pattern and the predictor) by using the strategy design pattern ..... 75

Figure 4-7. The genetic algorithm to configure the decision maker ..... 87

Figure 4-8. The decision maker algorithm ..... 88

Figure 4-9. Resource cost comparison ..... 95

Figure 4-10. SLA violation count comparison ..... 96

Figure 4-11. Total cost comparison ..... 97

Figure 5-1. Experimental setup (experiment I) ..... 101

Figure 5-2. The HMM and the Amazon scaling actions ..... 105

Figure 5-3. The architectural view of the experiment setup (experiment II) ..... 107

Figure 5-4. MAE and RMSE values (periodic pattern) ..... 113
Figure 5-5. MAE and RMSE values (growing pattern) ..... 116
Figure 5-6. MAE and RMSE values (unpredictable pattern) ..... 117
Figure 5-7. The workload patterns ..... 120
Figure 5-8. VM thrashing factor ..... 122
Figure 5-9. The workload patterns, the ceiling capacity, and the floor capacity ..... 126
Figure 5-10. The population size effect on the experiment duration and the fitness value ..... 138
Figure 5-11. The number of generation's effect on the fitness value and the experiment duration ..... 139
Figure 5-12. The crossover rate's effect on the fitness value and the experiment duration ..... 141
Figure 6-1. Typical cloud service with 3-tier architecture ..... 151
Figure 6-2. Business tier scaling actions (periodic pattern) ..... 161
Figure 6-3. Business tier scaling actions (growing pattern) ..... 162
Figure 6-4. Business tier scaling actions (unpredictable pattern) ..... 163
Figure A-1. Class diagram of the simulation package ..... 173
Figure A-2. Class diagram of the simulation package ..... 174
Figure A-3. Simulation package sequence diagram ..... 177

## List of Acronyms

| | |
|---|---|
| Amazon EC2 | Amazon Elastic Compute Cloud |
| AMI | Amazon Machine Image |
| ANN | Artificial Neural Network |
| AR | Auto Regression |
| AWS | Amazon Web Services |
| BT | Business Tier |
| DT | Database Tier |
| durL | Lower scaling duration |
| durU | Upper scaling duration |
| EB | Emulated Browser |
| GA | Genetic Algorithm |
| GUI | Graphical User Interface |
| HMM | Hidden Markov Model |
| IaaS | Infrastructure as a Service |
| inL | Lower cool down duration |
| inU | Upper cool down duration |
| IT | Information Technology |
| MA | Moving Average |
| MAE | Mean Absolute Error |
| MLP | Multi-Layer Perceptron |
| MLPWD | Multi-Layer Perceptron with Weight Decay |
| MOA | Massive Online Analysis |

| | |
|---|---|
| LQNM | Layered Queueing Network Model |
| LQNS | Layered Queueing Network Simulator |
| LR | Linear Regression |
| PaaS | Platform as a Service |
| QNM | Queueing Network Model |
| QoE | Quality of Experience |
| QoS | Quality of Service |
| RMSE | Root Mean Square Error |
| SaaS | Software as a Service |
| SLA | Service Level Agreement |
| SUT | System Under Test |
| SVM | Support Vector Machine |
| thrL | Lower Threshold |
| thrU | Upper Threshold |
| VM | Virtual Machine |
| WEKA | Waikato Environment for Knowledge Analysis |

## List of Symbols

| | |
|---|---|
| $C_R$ | Resource cost |
| $c_{VM}$ | Hourly rate of leasing a virtual machine |
| req | Incoming request |
| $v_t$ | Total number of SLA violations at time *t* |
| $C_{SLA}$ | SLA violation cost |
| $c_b$ | Cost of a SLA violation |
| $C_T$ | Total auto-scaling cost |
| E(w) | Empirical risk |
| R(w) | Functional (actual) risk |
| L(y, f(x,w)) | Loss function, *y* is the actual value and f(x,w) is the predicted value |
| P(x) | Workload distribution |
| $C_o$ | Maximum value of the confidence interval |
| $CapU_{vm}$ | Upper threshold of the number of the requests that can be handled by each of the VMs per second |
| $Cap_{ceiling}$ | maximum number of the requests that can be handled by the cloud service |
| $Cap_{floor}$ | maximum number of the requests that can be handled by the cloud service after being scaled in |
| $CapL_{vm}$ | Lower threshold of the number of the requests that can be handled by each of the VMs per second |
| ISD | Identical scaling decisions |

| | |
|---|---|
| $R(\lambda)$ | Response time |
| $D_k$ | Service demand of service center *k* |
| $U_k$ | Utilization of service center *k* |

# Chapter 1 - Introduction

The elasticity characteristic of cloud computing, conjointly with cloud's pay-as-you-go pricing model, reduces the cloud clients' cost by allowing them to merely pay for the resources they actually use. However, maintaining Service Level Agreements (SLAs) with the end users obliges the cloud clients to deal with the cost and the performance trade-off. The trade-off can be balanced by finding the minimum amount of resources the cloud clients need to fulfill their SLAs obligations. Since the cloud clients' workload varies with time, the cost/performance trade-off needs to be justified in accordance with the incoming workload. Auto-scaling systems are developed to automatically balance the cost/performance trade-off. This thesis proposes a predictive auto-scaling system which minimizes the cloud clients' resource cost as well as prevents SLAs violations. This chapter presents the research problem of this thesis and outlines the approach to address the research problem. The first half of this chapter (i.e., sections 1.1, 1.2, 1.3, 1.4 and 1.5) describes the research problem, the research motivation, the research scope, and the research goal. The second half of this chapter (i.e., sections 1.7, 1.8, and 1.9) introduces the research approach, the thesis contributions, and the thesis outline.

## 1.1 Research Motivation

Deciding the optimal amount of resources in the cloud computing environment is a double-edged sword which may lead to either under-provisioning or over-provisioning conditions. Under-provisioning condition is the result of saturation of the resources and may cause SLAs violation. In contrast, over-provisioning condition occurs when the provisioned resources are wasted which results in excessive energy consumption and

high operational cost [1]. Auto-scaling systems are presented to automatically balance the cost/performance trade-off and prevent the under-provisioning and the over-provisioning conditions.

Existing auto-scaling systems can be divided into reactive and predictive categories. The reactive auto-scaling systems are the most widely used systems in the industrial environments that scale out/in a cloud service according to its current performance condition. Although the reactive auto-scaling systems are easy to use and understand, they neglect virtual machine (VM) boot-up time which is reported to be between 5 and 15 minutes [1][2][3]. Neglecting VM boot-up time results in the under-provisioning condition which causes SLAs violation. The predictive auto-scaling systems forecast the future workload of the cloud service and adjust the resource capacity in advance to meet the future needs. Even though the predictive systems can remedy the shortcoming of the reactive approaches, existing predictive approaches suffer from poor accuracy (see Section 2.2). The poor accuracy of the existing predictive auto-scaling systems persuades the cloud clients to choose the reactive systems over the predictive systems [4]. This thesis focuses on improving the accuracy of the predictive auto-scaling systems. The results of this thesis can be used to decrease the cloud clients' resource cost as well as to reduce the cloud clients' SLAs violations.

## 1.2 Research Scope

There are three fundamental models associated with cloud computing [5]:

- Infrastructure-as-a-Service (IaaS): provides virtualized computing resources such as processor, storage, and bandwidth over the Internet. Examples are Amazon EC2 and Google Compute Engine.

- Platform-as-a-Service (PaaS): accommodates programming environment and tools for the cloud clients to build and deploy cloud services onto the cloud infrastructure. Examples include Amazon Elastic Beanstalk, Google App Engine, and Microsoft Windows Azure.
- Software-as-a-Service (SaaS): provides hosted vendor cloud services. Examples are Google Apps, Salesforce.com, and Microsoft Office 365.

These services can be made accessible to public (i.e., public cloud), restricted for private uses (i.e., private cloud), or be hosted on a hybrid cloud which is a combination of both public and private clouds [6]. In the public cloud, clients have little or no control over the cloud environment. In addition, because the public clouds are reachable through the Internet, they are vulnerable to potential security risks. In the private cloud, clients benefit from a high degree of control over the environment as well as low security risks. The private clouds typically are internal data centers of an organization and they are not reachable for the public clients. This research is interested in the auto-scaling of the IaaS services in the public clouds.

Resource allocation in the IaaS layer of the public clouds can be either horizontal or vertical. In the horizontal scaling (i.e., scaling out/in), the resource unit is a server replica running on a VM, and VMs are added or released to scale the cloud service. In contrast, the vertical scaling (i.e., scaling up/down) is managed by changing the power (such as RAM size, or CPU power) of an existing VM. The most common operating systems do not allow on-the-fly changes to the virtual machines. For this reason, the most cloud providers only offer the horizontal scaling [4]. This research considers the horizontal

scaling as the auto-scaling approach. Figure 1-1 illustrates the typical stakeholders and their relationship in the IaaS environment:

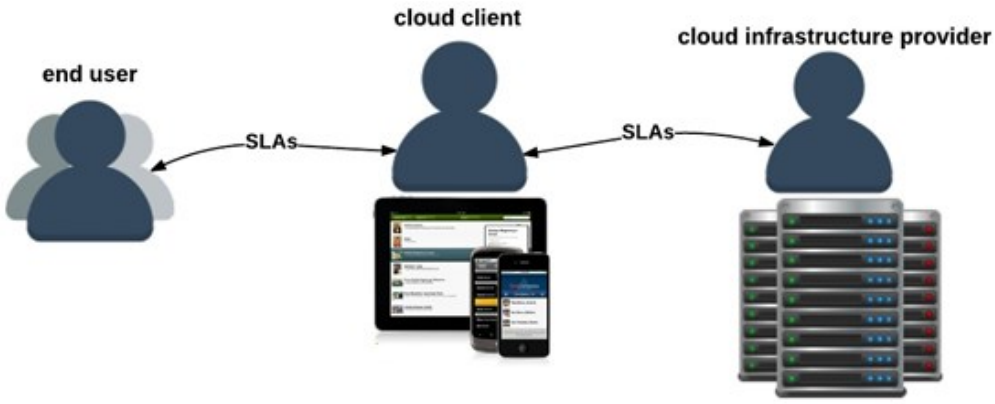


**Figure 1-1. The IaaS environment stakeholders and their relationships**

The three stakeholders in the IaaS environment are [4]:

- Cloud infrastructure provider: refers to the IaaS provider who offers logically unlimited virtual resources in the form of VMs, virtual networks, etc.
- Cloud client: is the customer of the IaaS provider who uses the infrastructure for hosting the cloud service. The cloud client also is referred to as the cloud service provider.
- End user: is the user that accesses the cloud service and generates the workload that drives the cloud service's behavior.

There are two types of SLAs in a cloud computing environment: SLAs between the *end user* and the *cloud client*, and SLAs between the *cloud client* and the *cloud infrastructure provider*. In this research we investigate the cost/performance trade-off from the cloud clients' perspective. A cloud client owns a web service which is deployed

in an IaaS infrastructure. This thesis considers a typical web service with 3-tier architecture, which includes the load balancer, the business tier, and the database tier. Although to scale in/out the web service, all of its tiers should be scaled in/out, this thesis focuses on the business tier scaling and investigates the impact of the database tier on the business tier.

## 1.3 Problem Statement

The auto-scaling process matches MAPE loop of the autonomous systems [4]. The MAPE loop consists of Monitoring (M), Analysis (A), Planning (P), and Execution (E) phases. This is a repetitive loop which in each repetition the auto-scaling system monitors the cloud service's performance and its incoming workload (i.e., Monitoring phase), uses the retrieved information to estimate the future resource need (i.e., Analysis phase), plans a suitable resource modification action (i.e., Planning phase), and finally uses the IaaS provider's APIs to execute the scaling actions (i.e., Execution phase). This thesis focuses on the *Analysis* and the *Planning* phases of the predictive auto-scaling systems with the goal of increasing the accuracy of the predictive auto-scaling systems.

A comprehensive auto-scaling system covers various technologies; some of which are beyond the scope of this thesis. In addition, a few assumptions have been made to scope the thesis:

**Assumption 1:** The goal of the predictive auto-scaling system is to lease the minimum amount of VMs from a given IaaS provider to provision a certain level of Quality-of-Service (QoS) for the cloud end users.

**Assumption 2:** There is only one type of VM provisioned by the IaaS provider. The provisioned VM type has a fixed hourly rate.

**Assumption 3:** In each of the MAPE iterations, only one VM can be added to or removed from the set of provisioned resources.

**Assumption 4:** The auto-scaling system gets the following characteristics of the incoming workload as input from the cloud client:

- Mean service time of the incoming requests on a business tier and a database tier VM.
- Database access rate, which indicates the percentage of the incoming requests that need to access the database.

**Assumption 5:** There is an unlimited number of VMs provided by the IaaS provider.

**Assumption 6:** The load balancing service is supplied by the IaaS provider and has an unlimited capacity.

**Assumption 7:** There is only one class of SLAs violation cost. In other words, the entire SLA violations are treated the same and have an equal cost.

## 1.4 Formal Problem Definition

This thesis addresses the predictive auto-scaling problem in the IaaS layer of the cloud computing environments. To formulate the problem, a few definitions are provided:

**Definition 1 (resource cost):** Resource cost refers to the cost of the leased VMs. This thesis assumes that the IaaS provider supplies only one type of VM with a fixed hourly rate. The resource cost is:

$$C_R = \sum_{t=0}^{T} n_t \times c_{vm} \quad (1\text{-}1)$$

where $T$ is the total hours the auto-scaling system is running, $c_{vm}$ is the hourly rate of leasing a VM, and $n_t$ is the number of the leased VMs between hour $t$ and $t+1$.

**Definition 2 (SLAs violation):** SLAs violation refers to any act or behavior that does not comply with the SLAs document. In this thesis, response time is considered to be the main QoS (see Section 2.2.3) and any request with a response time more than the maximum response time (which is defined in the SLAs document) is recognized as a SLAs violation. Total number of SLAs violations at time $t$ is:

$$v_t = \sum_{req=1}^{N} v_{t,req} \tag{1-2}$$

$$v_{t,req} = \begin{cases} 1 & if\ (r_{req} - R) > 0 \\ 0 & otherwise \end{cases}$$

where $req$ represents an incoming request, $N$ is the total number of requests at time $t$, $r_{req}$ is the response time of the request $req$, and $R$ is the maximum response time defined in the SLAs document.

**Definition 3 (SLAs violation cost):** Measuring SLAs violation cost is not a trivial task and depends on the different factors, such as the downtime duration of the cloud service, the number of affected end users, and even the sociological aspects of the end users' behaviors. In this thesis a constant penalty $c_b$ is assigned to each SLAs violation. The value of $c_b$ is defined by the cloud client who provides the cloud service. The SLAs violation cost is:

$$C_{SLA} = \sum_{t=o}^{T} v_t \times c_b \tag{1-3}$$

**Definition 4 (Workload):** Workload is the consequence of the end users accessing the cloud service or the consequence of the jobs that need to be handled

automatically. Workload of a cloud service in a given time unit is the number of the users who access the cloud service in that time unit.

**Definition 5 (Auto-scaling system):** The auto-scaling system provides a mechanism that generates scaling actions to accommodate the incoming users' requests. A scaling action can be scale out, scale in, or no scale. Scale out action increases the resource cost (i.e., $C_R$), while scale in or no scale actions may increase SLAs violation cost (i.e., $C_{SLA}$). The auto-scaling system is a function from the workload (i.e., $W$) to the cost (i.e., $C_T$).

$$f: W \rightarrow C_T \tag{1-4}$$

**Definition 6 (Auto-scaling accuracy)**: The auto-scaling accuracy is closely related to the cost incurred by the cloud clients. The more accurate the auto-scaling system, the lower the cost incurred by the cloud clients. Therefore, cost is the main metric that measures the accuracy of the auto-scaling systems. The total cost of a given auto-scaling system $C_T$ is the summation of its resource cost and its SLAs violation cost:

$$C_T = C_R + C_{SLA} = \sum_{t=0}^{T}((n_t \times c_{vm}) + (v_t \times c_b)) \tag{1-5}$$

## 1.5 Research Goal and Objectives

The goal of this thesis is to improve the accuracy of the predictive auto-scaling systems. According to Section 1.4, the most accurate auto-scaling system is the auto-scaling system which minimizes the cloud clients' resource and SLAs violation costs.

Reducing the resource cost (i.e., acquiring fewer resources from the cloud provider) decreases the cloud service's compute power. This reduces the cloud service's

performance and causes SLAs breach (i.e., increases the SLAs violation cost). Therefore, the resource cost and the SLAs violation cost cannot be minimized at the same time. The goal of this thesis is to propose a predictive auto-scaling system that finds the balance point between the cloud client's resource cost and the SLAs violation cost. The proposed auto-scaling system minimizes the total auto-scaling cost.

The predictive auto-scaling systems consist of *Monitor*, *Predictor*, and *Decision Maker* components (see Figure 1-2). The *Monitor* component scans one or more performance indicators. The performance indicators values are sent to the *Predictor* component. The *Predictor* component uses the current and the historical values of the performance indicators to forecast their future values. This prediction is used by the *Decision Maker* to generate scaling actions.

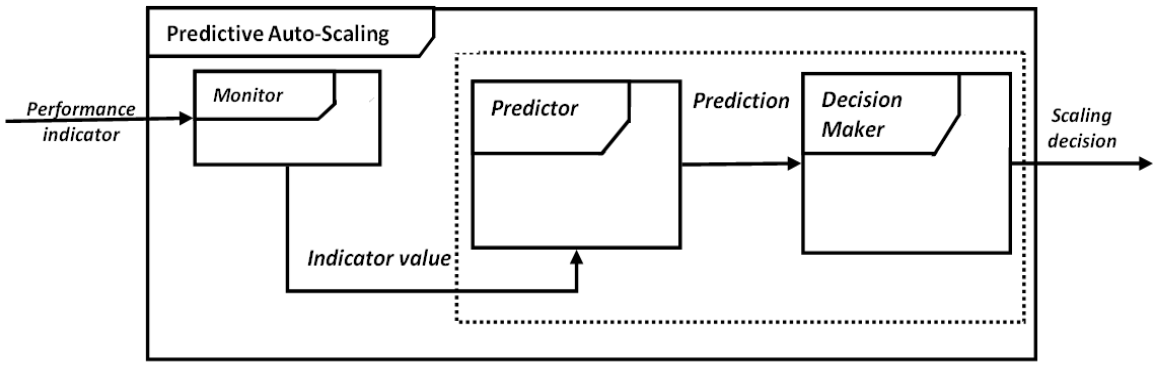


**Figure 1-2. The architectural overview of the predictive auto-scaling systems** [6]

To improve the accuracy of the predictive auto-scaling systems, the accuracy of each of their components (i.e., *Monitor, Predictor*, and *Decision Maker*) should be investigated and improved. Since the existing *Monitor* components are highly accurate

[7], this thesis focuses on investigating the accuracy of the *Predictor* and the *Decision Maker* components.

According to [4], the most dominant prediction technique in the cloud auto-scaling area is time-series prediction. In recent years, many time-series prediction techniques have been proposed for the auto-scaling systems in various disciplines. The first objective of this thesis is:

1. To investigate the existing time-series prediction techniques, identify the mathematical foundation of the prediction techniques, and propose a prediction suite which is tailored to the needs of the IaaS layer of the cloud computing environment.

In addition, most of the existing *Decision Maker* components only consider the business tier of the cloud services for the auto-scaling purposes (see Section 2.3). However, most of the cloud services include a database tier, as well. The impact of the database tier capacity is not known on the business tier auto-scaling decisions. Thus, the second objective of this thesis is:

2. To investigate the impact of the database tier capacity on the business tier auto-scaling decisions.

The rule-based systems are currently the most popular auto-scaling systems in the industrial environments [4]. Configuration is significantly important for the accuracy of the rule-based systems. However, finding the optimal configuration of the rule-based systems is not a trivial task. Therefore, the third objective of this thesis is:

3. To propose an approach to automatically configure the rule-based decision makers. The configuration should reduce the total auto-scaling cost based on the cloud clients' cost preference.

Eventually, after enhancing the accuracy of the *Predictor* and the *Decision Maker* components, the fourth objective of this thesis is:

4. To propose an accurate predictive auto-scaling system and to evaluate the proposed system against a popular industrial auto-scaling system.

## 1.6 Research Approach

In this research we use a quantitative research process that is formal, objective, and systematic. In the quantitative research process, numerical data are utilized to obtain information about the research problem [8]. The quantitative research process follows a deductive reasoning approach and its steps are shown in Figure 1-3.

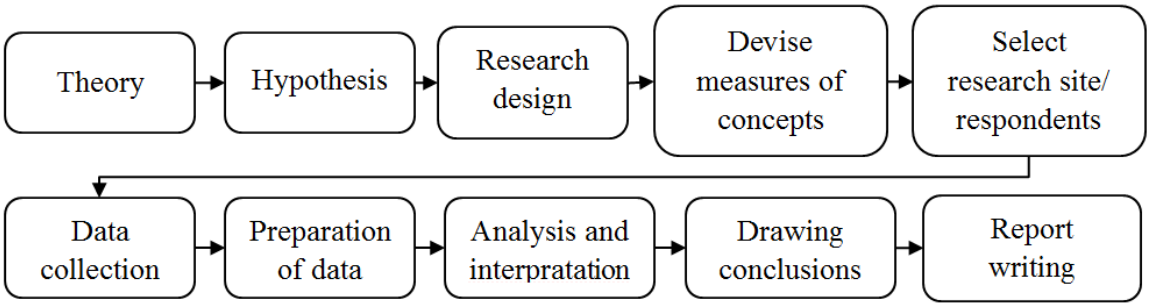


**Figure 1-3. Quantitative research process** [8]

The quantitative research process starts with investigation of a *theory* which serves as a standardized principle to explain the relationships between two or more concepts. *Hypothesis* is introduced based on the theory investigation and the research is designed

according to the hypothesis. *Research design* includes identifying research questions and objectives. The next step is to devise *measures of concepts*. This step identifies the measurable and non-measurable indicators, which serve to validate the hypothesis. In the fifth step (i.e., *select research site*), an evaluation method of the theory is selected. In the rest of the steps, evaluation data will be gathered and analyzed [8].

To improve the accuracy of the predictive auto-scaling systems, this thesis proposes a set of hypotheses for the research and evaluates them mathematically and experimentally. Chapters 3, 4, and 5 present the hypotheses of this thesis, the evaluation methods to asses each hypothesis, and the results.

## 1.7 Thesis Contributions

The main contributions of this thesis are:

1. A proposal of a self-adaptive prediction suite. The prediction suite includes a set of the prediction methods, and chooses the most accurate prediction method based on the pattern of the incoming workload. This contribution resulted into the following publications:

    A. Y. Nikravesh, S. A. Ajila, and C.-H. Lung, "Towards an Autonomic Auto-Scaling Prediction System for Cloud Resource Provisioning," Proceedings of the 10th International Symosium on Software Engineering for Adaptive and Self-Managing Systems (*SEAMS*), pp. 35-45, 2015

A. Y. Nikravesh, S. A. Ajila, and C.-H. Lung, "Using MLP, MLPWD, and SVM to Forecast Mobile Network Traffic," Proceedings of IEEE International Congress on Big Data, 2016

2. A comprehensive evaluation of the sensitivity of the rule-based decision makers to the prediction results in the predictive auto-scaling systems. This contribution resulted into the following publication:

   A. Y. Nikravesh, S. A. Ajila, and C.-H. Lung, "Measuring Prediction Sensitivity of a Cloud Auto-Scaling System", Proceeding of the 7th IEEE International Workshop on Service Science and Systems, in collaboration with the 38th IEEE International Computers, Software & Applications Conference, pp. 690-695, 2014

   A. Y. Nikravesh, S. A. Ajila, and C.-H. Lung, "Evaluating Sensitivity of Auto-Scaling Decisions in an Environment with Different Workload Patterns," Proceedings of the 39th International Computer Software and Applications Conference, pp. 415-420, 2015

3. An investigation of the impact of the database tier capacity on the business tier auto-scaling decisions.
4. A proposal of a cost driven decision maker to reduce the total auto-scaling cost based on the cloud client's preferences. This contribution resulted into the following publication:

   A. Y. Nikravesh, S. A. Ajila, and C.-H. Lung, "Cloud Resource Auto-scaling System Based on Hidden Markov Model (HMM)," Proceedings of

IEEE International Conference on Semantic Computing, pp. 124-127, 2014

5. An evaluation of the proposed predictive auto-scaling system against the Amazon auto-scaling system.

## 1.8 Outline of the Thesis

This thesis contains seven chapters. The rest of the thesis is organized as follows:

Chapter 2 presents a literature review as well as an overview of the research background. Chapter 3 describes an introduction to the machine learning algorithms used in this thesis and their mathematical analysis. The proposed prediction suite and the cost driven decision maker are introduced in Chapter 4. In addition, an evaluation of the proposed predictive auto-scaling system is presented in Chapter 4. The conducted experiments that lay out the groundwork for the proposed auto-scaling system are presented in Chapter 5. The impact of the database capacity on the business tier's auto-scaling is discussed mathematically and experimentally in Chapter 6. Finally, Chapter 7 concludes the thesis and recommends the possible directions for the future research work.

# Chapter 2 - Literature Review

This chapter presents the existing work on the cloud resource provisioning. Section 2.1 discusses approaches that focus on reducing the VM boot-up time. Section 2.3 overviews the most dominant auto-scaling techniques and highlights their shortcomings. Section 2.2, introduces the background concepts that are used in this thesis.

## 2.1 Reducing the Virtual Machine Boot-Up Time

In the horizontal auto-scaling, VMs are building blocks of the underlying IaaS infrastructure and VM instantiation time (i.e., boot-up time) plays a pivotal role in the accuracy of the auto-scaling systems. VM boot-up time is the time it takes for the acquired VMs to be ready for use [9]. The longer is the VM boot-up time, the higher is the risk of the under-provisioning condition. Therefore, researchers have strived to speed up the VM instantiating process to reduce the under-provisioning risk [10]. However, speeding up the VMs instantiation process does not completely prevent the under-provisioning issue.

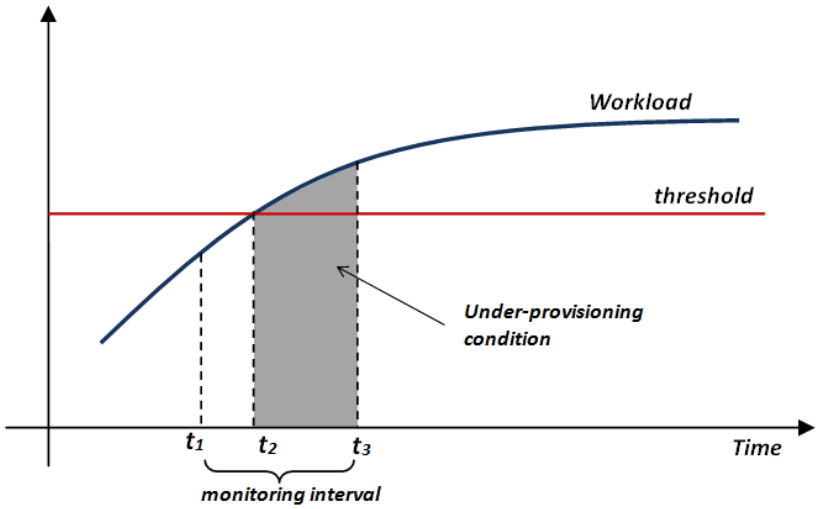


**Figure 2-1. The effect of the monitoring interval on the under-provisioning condition**

Auto-scaling systems monitor the workload of the cloud service and scale out the underlying infrastructure if the workload exceeds a certain threshold. The workload value is being retrieved at certain points of times (e.g., at times $t_1$ and $t_3$ in Figure 2-1). However, it is possible that the workload exceeds the threshold during the monitoring interval (e.g., at time $t_2$ in Figure 2-1). In this case, even if the VM boot-up is instantaneous, the cloud service becomes under-provisioned until the next monitoring time (e.g., between times $t_2$ and $t_3$ in Figure 2-1). Therefore, shortening the VM boot-up time does not prevent the under-provisioning condition. It can be argued that reducing the monitoring interval mitigates the aforementioned issue. But reducing the monitoring interval only shortens (i.e., does not completely prevent) the under-provisioning condition. Furthermore, reducing the monitoring interval downgrades the auto-scaling system's performance.

## 2.2 Background Concepts

Figure 2-2 illustrates components of a typical IaaS environment. The auto-scaling systems work as an intermediate between the cloud service and the IaaS provider to carry out the resource provisioning task.

To setup an experimental environment to investigate the auto-scaling systems, it is necessary to select an appropriate IaaS provider, a performance indicator, QoS metrics, and a cloud service benchmark. Section 2.3.4 introduces the performance indicators and their corresponding target tier. In this section the other components of the IaaS environment (i.e., the IaaS providers, the QoS metrics, and the cloud service benchmark options) are presented and the elements that are used in this thesis are introduced.

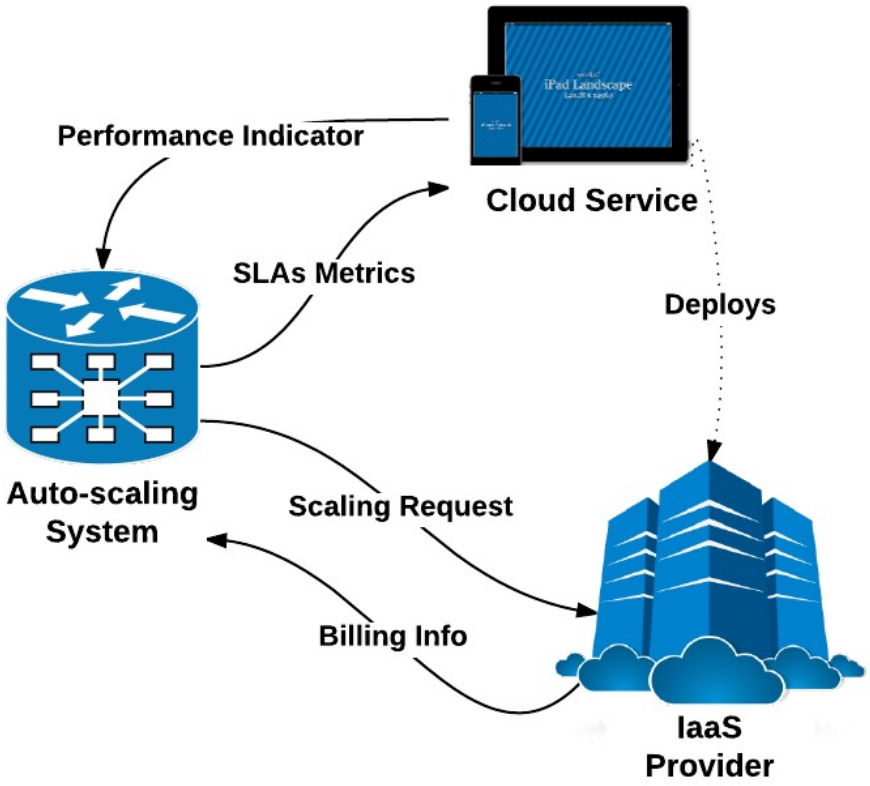


**Figure 2-2. The IaaS environment components**

### 2.2.1 Infrastructure-as-a-Service Providers

There are different infrastructure options to be used to conduct the auto-scaling experiments. The first option is to use the industrial IaaS environments such as Amazon EC2 or Google Compute Engine. The industrial IaaS providers give few or no control to the clients to manage the cloud environment [2]. Therefore, to reduce the experimentation cost and to have a more controlled environment, the IaaS auto-scaling experiments can be done by using a custom test-bed. For the custom test-beds, virtualization at the server level is needed. Server level virtualization is commonly referred to as *Hypervisor* or

*Virtual Machine Monitor* (VMM) [4]. Some of the popular hypervisors are: Xen [11], VMWare ESXi [12], and Kernel Virtual Machine (KVM) [13].

The custom test-beds can be deployed in the open-source (e.g., Eucalyptus [14] and OpenStack [15]) or the commercial (e.g., vCloud Director [16]) platforms. Eucalyptus enables the creation of on-premises Infrastructure-as-a-Service clouds, with support for Xen, KVM, ESXi, and Amazon EC2 API. Similarly, OpenStack is another open-source initiative that is supported by many of the enterprises such as Cisco, RackSpace, HP, and Intel. VCloud Director is a commercial platform that is developed by VMWare.

In addition, software simulations can be used to mimic the cloud platform behaviors, including resource allocation and de-allocation, VM execution, and monitoring. There are multiple advantages in using the software simulators. The software simulators test multiple algorithms without having to reconfigure the infrastructure each time [4]. Moreover, experiments that are carried out in the real infrastructures may last for hours, whereas in an event-based simulator, this process takes only few minutes. Simulators are highly configurable and allow the clients to gather any information about the system state or the performance metrics. On the other hand, a simulated environment is still an abstraction of the physical machine clusters, thus the reliability of the software simulators results depends on the level of the implementation details considered during the development [4].

In this thesis too much control over the infrastructure is not needed, therefore the industrial IaaS environments were used to conduct the experiments. Amazon EC2 has comprehensive online documentations and includes helpful management tool sets. Thus, Amazon EC2 is used as the IaaS provider in our experiments.

#### 2.2.1.1 Amazon Elastic Compute Cloud (Amazon EC2)

Amazon Elastic Compute Cloud (Amazon EC2) is a web-service that provides scalable compute capacity in the cloud and is designed to facilitate the web-scale computing for the developers [17]. Amazon EC2 is a central part of the Amazon Web Services (AWS) platform and provides the developers with a web service which allows the developers to boot an Amazon Machine Image (AMI) to create the virtual machines. These virtual machines are called instances and the developers can create, launch and terminate them as needed. An AMI is a template that contains a software configuration, including an operating system, which defines the client's operating environment.

There are three options to purchase the Amazon EC2 instances: On-demand, reserved, and spot instances. The on-demand instances let the clients to hourly pay for the compute capacity with no long-term commitments. The reserved instances allow the clients to make a low, on-time payment for each of the instance that they want to reserve. The reserved instances are significantly cheaper than the on-demand instances, but to use the reserved instances the clients have to know the amount of the instances they are going to use in advance, which sometimes is not a straightforward task. Finally, the spot instances allow the clients to bid on the unused Amazon EC2 capacity and run those instances for as long as their bid exceeds the current spot price. In addition to the purchasing options, Amazon EC2 has a range of instance types, including: standard instances, micro instances, high-memory instances, and high-CPU instances. For more details about the Amazon EC2 instances and their specifications see [17].

### 2.2.2 Cloud Service Benchmarks

One of the significant components to test the auto-scaling systems is the workload generator. Synthetic workloads, real traces and benchmarks can be used to mimic the cloud service behavior. The synthetic workloads can be generated with different patterns. There is a broad range of the workload generators, such as:

- Httpref [18]: generates various HTTP workloads for measuring the server performance.
- SWAT [19]: a workload generator that is typically used for the stress testing. SWAT uses Httperf for the request generation.
- Faban [20]: a Markov-based workload generator, which is used in the CloudStone [21] stack.
- Rain [22]: a statistics-based workload generation toolkit that uses parameterized and empirical distributions to model the different classes of the workload variations.
- SURGE [23]: uses an offline trace generation engine to create the traces of the requests. This generator is mainly used for testing the network and the server performance.
- Apache Jmeter [24]: a Java workload generator that tests the performance both on the static and the dynamic resources.

The synthetic workloads are widely used for the controlled experiments. However, the workload traces that are generated by the synthetic workload generators are not realistic [4]. To have more realistic workloads, the real trace files can be used to mimic the cloud service behavior.

According to [4], there is no publicly available real trace file from the cloud providers. However, the general real trace files – not the cloud specific trace files – have been used by some of the researchers to investigate the auto-scaling performance. ClarkNet [25] and World Cup 98 [26] trace files are the most widely used real trace files in the IaaS auto-scaling area.

In addition to the synthetic workloads and the real trace files, benchmarks can be used to evaluate the server's performance. Typically, a benchmark includes a web service and a workload generator. Therefore, the benchmarks inherit the innate shortcomings of the synthetic workload generators. The commonly used benchmarks are TPC-W [27], CloudStone [21], RUBiS [28], and WikiBench [29]. In this thesis the benchmark approach is used to model the behavior of a cloud service in the IaaS environment. The reasons for choosing the benchmarks over the synthetic workload and the real trace files are:

- Contrary to the real trace files, the benchmarks can generate the different workload patterns
- Contrary to the synthetic workload generators, the benchmarks benefit from the embedded cloud services which makes the benchmarks be easier to deploy and to use.

Among the aforementioned benchmarks, Java implementation of TPC-W is used as the benchmark in our experiments. TPC-W is simple use and has extensive online documentations. Section 2.2.2.1 introduces the TPC-W benchmark.

#### 2.2.2.1 TPC-W

Transactional Processing Performance Council (TPC) is a non-profit organization that is founded to define the transaction processing and the database benchmarks. TPC-W is a benchmark provided by TPC. TPC-W is a complex e-commerce application that models an online bookstore and is commonly used for the resource provisioning, the scalability and the capacity planning for the traditional e-commerce websites. The components of TPC-W can be logically divided into three tiers: a set of emulated web browsers, a web server, and a database tier. The web browsers are emulated by using RBE (Remote Browser Emulator). RME mimics a number of simultaneous end user sessions and allows a single node to emulate several users.

TPC-W uses the Emulated Browsers (EB) concept to generate requests for the System-Under-Test (SUT). An EB emulates a user communication with the SUT by using a browser to send and receive HTML content through HTTP and TCP/IP protocols over a network connection. The number of EBs that are used for a given test is determined by the size and scaling factor of the SUT. A RBE creates and manages one EB for each of the emulated users. The number of EBs is constant throughout an experiment.

In this thesis TPC-W is used to generate the different workload patterns (i.e., growing, periodic, and unpredicted). Since the number of EBs specifies the number of the user requests per second, adjusting the number of EBs throughout the experiment can result in the different workload patterns. This idea was used to create a customized script for generating the different workload patterns by using TPC-W benchmark (cf. Figure 2-3).

```
#!/bin/bash
counter=0
bound=1000
duration=14400
workloadPattern=INPUT
trend=1
max=1000
min=10
while [$counter -lt $duration]
do
      let counter=counter+1
      if [$workloadPattern = "Unpredicted"]
      then
            load=$RANDOM
            load=$((load%bound))
      elif [$workloadPattern = "Growing"]
      then
            if [[$counter -gt 5000] && [$counter -lt 14000]]
            then
                  load=load*10
            else
                  load=load+10
            fi
      else
            if [ $trend = 1]
            then
                  load=$load+10
                  if [ $load -gt $max ]
                  then
                        $trend=0
                  fi
            else
                  load=$load-10
                  if [ $load -lt $min ]
                  then
                        $trend = 1
                  fi
            fi
      fi
      java -mx512M rbe.RBE -EB rbe.EBTPCW1Factory $rand  -OUT run1.m
-RU 1 -MI 3 -RD 1 -WWW {machine ip address}:{port}/tpcw/ -CUST 144000
-ITEM 10000 -TT 1.0
done
```

**Figure 2-3. The customized script to generate the different workload patterns**

### 2.2.3 Service Level Agreements

A service level agreement (SLA) is a contract between a service consumer and a service provider that specifies how the interactions between the two parties are carried out and which contractual parties are involved. An important aspect of a contract for IT services

is a set of QoS guarantees [30]. A SLA describes the entire aspects of a service including the overall cost, the financial aspects of the service delivery such as the fees and the penalties, the service conditions, and the specific performance metrics governing the compliant service delivery. The individual performance metrics are called Service Level Objectives (SLO). This thesis is interested in the SLO section of the SLAs. In this thesis terms SLAs and SLOs are used interchangeably.

SLOs are built around one or more constrained QoS objects. In this thesis the auto-scaling problem is investigated from the cloud clients' (i.e., the cloud service provider) point of view. The cloud clients, on the one hand, try to minimize the cost by decreasing their cloud resource usage, and on the other hand, are obligated to maintain the SLOs with the end users. The SLOs between the cloud clients and their end users include a list of the QoS attributes and the values that the cloud client guarantees to provision for each of the QoS attributes. Authors in [31] have grouped the web service QoS attributes into the four categories. Table 2-2 shows these categories and their corresponding quality attributes.

In this thesis response time is considered as the SLO that is guaranteed by the cloud client. Response time is one of the most important metrics that has been monitored in lots of the existing auto-scaling mechanisms (see Table 2-2).

### 2.2.4 Workload

Workload is the consequence of the end users accessing the cloud service or the consequence of the jobs that need to be handled automatically [32]. According to [6] and [33] there are four workload patterns:

- **Stable workload** is characterized by the constant number of the requests per minute. This means that there is normally no explicit necessity to add or remove the processing power, the memory or the bandwidth for the workload changes.
- **Growing workload** represents a load that rapidly increases.
- **Cycle/Bursting workload** represents regular periods (i.e., seasonal changes) or regular bursts of the load in a punctual date
- **On-and-Off workload** shows the batch processing.

Authors in [32] categorize the application workload patterns as:

- **Static workloads** are characterized by flat utilization profile over time within the certain boundaries.
- **Periodic workload** represents a peaking utilization at reoccurring time intervals experience
- **Once-in-a-lifetime workload** is a special case of the periodic workload. In this type of workload, the peaks of the periodic utilization can occur only once in a very long timeframe. Often, this peak is known in advance as it correlates to a certain event or task.
- **Unpredictable workloads** are generalization of the periodic workloads as they require elasticity but are not predictable. This class of workload represents the constantly fluctuating loads.
- **Continuously changing workloads** represent a constant growth or decrease in the application load over time

Resource allocation for the batch applications (i.e., on-and-off pattern) is usually referred to as scheduling which involves meeting a certain job execution deadline [4].

Scheduling has been extensively studied in the grid environments and also is explored in the cloud environments. Scheduling is outside of the scope of this thesis. Similarly, cloud services with the stable (or static) workload pattern do not require an auto-scaling system for the resource allocation. Therefore, in this thesis only cloud services with the following workload patterns are investigated:

**Table 2-1. QoS attributes for web services**

| Category | Attribute | Description |
|---|---|---|
| Runtime related | Scalability | The ability of increasing the computing capacity of the service provider's computer system and the system's ability to process more operations or transactions in a given period |
| | Capacity | Limit of the concurrent requests for the guaranteed performance |
| | Response time | The guaranteed maximum (or average or min) time required to complete a service request |
| | Latency | Time taken between the service request arrives and the request is being serviced |
| | Throughput | The number of completed service requests over a time period |
| | Reliability | The ability of a service to perform its required functions under the stated conditions for a specified period of time. It can be measured by: Mean Time Between Failure (MTBF), Mean Time to Failure (MTF), and Mean Time To Transition (MTTT) |
| | Availability | The probability the system is running. Availability is related to reliability |
| | Robustness/ Flexibility | The degree to which a service can function correctly in the presence of invalid, incomplete or conflicting inputs |
| | Exception handling | Ability of the service to handle the exceptions. |
| | Accuracy | The error rate produced by the service |
| Transaction related | Integrity | Transactions can be grouped into a unit to guarantee the integrity of the data operated on by the transactions. |
| Configuration management and Cost related | Regulatory | A measure of how well the service is aligned with the regulations |
| | Supported Standard | A measure of whether the service complies with the standards |
| | Stability/change cycle | A measure of the frequency of change related to the service in terms of its interface and/or implementation |
| | Cost | A measure of the cost involved in requesting the |

| | | |
|---|---|---|
| | | service |
| | Completeness | A measure of the difference between the specified set of features and the implemented set of features |
| Security related | Authentication | Indicates how the service authenticates the principals |
| | Authorization | Indicates how the service authorizes the principals |
| | Confidentiality | Indicates how the service treats the data, so that only the authorized principals can access or modify the data |
| | Accountability | Indicates whether the suppliers are hold accountable for their services? |
| | Traceability and Auditability | Indicates whether it is possible to trace the history of a service when a request was serviced |
| | Data encryption | Indicates how the service encrypts the data |
| | Non-Repudiation | A principal cannot deny requesting a service or data after the fact. |

- **Growing workload**: represents workloads with the increasing trend. This class of workload covers the growing workload of [4] and the continuously changing workload of [32].
- **Periodic workload:** represents workloads with the seasonal changes. This class of workload covers the cyclic/bursting workload of [4] and the periodic and the once-in-a-lifetime workloads of [32].
- **Unpredictable workload:** represents the fluctuating workloads. This class of workload covers the unpredictable workload of [32].

## 2.3 Existing Auto-Scaling Techniques

To date, cloud practitioners have pursued different mechanisms to solve the auto-scaling problem. This section proposes a set of criteria to evaluate the existing auto-scaling mechanisms. The evaluation criteria are selected based on the scope of this thesis.

**Criterion 1 (Scaling method):** indicates the scaling approach of the method under the study. Scaling method can be reactive, predictive, or hybrid. The hybrid

auto-scaling methods use the reactive and the predictive mechanisms to decide about the scaling action.

**Criterion 2 (Target tier):** represents the software tier that is scaled by the method under the study. The target tier can be the business tier, the database tier, or a set of the software tiers.

**Criterion 3 (Prediction method):** specifies the prediction technique that is used by the method under the study.

**Criterion 4 (Decision making method):** indicates the mechanism that is used by the method under the study to generate the scaling decisions.

**Criterion 5 (Performance indicator):** represents the performance indicators that are monitored by the method under the study.

**Criterion 6 (Workload pattern):** defines the workload pattern that is used to evaluate the method under the study. Workload pattern can be growing, periodic, and unpredictable. See Section 2.3 for more details on the workload patterns.

The existing auto-scaling techniques are presented in Table 2-2. Since this thesis is scoped to the IaaS layer of the public clouds, research works that focus on the other cloud layers (i.e., PaaS or SaaS auto-scaling, such as [34]) or the other cloud types (i.e., private or hybrid clouds auto-scaling, such as [35][36]) fall out of the scope of this thesis. The following subsections discuss each of the evaluation criteria.

### 2.3.1 Target Tier

According to Table 2-2, most of the existing auto-scaling systems only scale the business tier and neglect the impact of the other software tiers (such as the database tier and the

load balancer) to the business tier scaling actions. However, adding VMs to the business tier shifts the bottleneck to a downstream tier (i.e., the database tier) [37]. Furthermore, most of the cloud services have multi-tier architecture. Because the software tiers of a given cloud service are closely related to each other, capacity of each tier can affect the scaling actions of the other tiers. Therefore, to increase the accuracy of the business tier auto-scaling, the impact of the database tier capacity on the business tier scaling actions should be investigated.

### 2.3.2 Prediction Method

The goal of the prediction methods in the auto-scaling domain is to forecast the future value of a performance indicator (i.e., $y_{t+1}$) based on the last $w$ observations of the indicator value (i.e., $[y_t, y_{t-1}, y_{t-2}, \dots, y_{t-w+1}]$). According to Table 2-2, prediction techniques that are used for this purpose in the literature are Moving Average, Auto-Regression, exponential smoothing, ARMA, and machine learning algorithms.

Moving Average (MA) method forecasts the future value by calculating the weighted average of the last historical values. In other words, $MA(q)$ assumes the future value of the time-series is the arithmetic mean of the last $q$ values (i.e., MA assigns an equal weight of $\frac{1}{q}$ to all of the observations). In contrast, Weighted Moving Average $WMA(q)$ assigns a different weight to each of the observations. Typically, more weight is given to the most recent observations in the time series, and less weight is assigned to the older observations. MA method generally generates poor results for the time-series analysis [4] and is usually applied to remove the noise from the time-series.

Auto-Regression (AR) is largely used in the IaaS resource provisioning field. AR prediction formula is determined as the linear weighted sum of the previous historical observations in the time series. The results in [38] show that the performance of AR highly depends on the monitoring interval length, the size of the history window, and the size of the adaptation window.

ARMA is a combination of Moving Average and Auto-Regression algorithms. According to [4], ARMA is a suitable model for the stationary processes (i.e., the mean and the variance of the time-series stay constant over time). Therefore, to increase the ARMA's prediction accuracy, the time-series must not show any trend (the variations of the mean) or any seasonal changes.

Machine learning algorithms have been used by many researchers (such as [1][5]) to handle the resource provisioning task in the IaaS layer. In contrast to the ARMA models, machine learning algorithms are suitable for the time-series with trends or seasonal changes. Chapter 3 presents the machine learning concept and introduces the machine learning algorithms that are used in the resource provisioning field.

### 2.3.3 Decision Making Approach

According to Table 2-2, the existing auto-scaling approaches can be grouped into five categories: rule-based policies, reinforcement learning, queuing theory, control theory, and time-series analysis. Authors in [4] have introduced the same categories for the existing auto-scaling systems. Among these categories, the time-series analysis focuses on the prediction side of the resource provisioning task and is not a "decision making" technique per se. In contrast, the rule-based technique is a pure decision making

mechanism. The rest of the auto-scaling techniques play the “predictor” and the “decision maker” roles at the same time.

Queuing theory models each VM as a queue of requests and uses the established methods to calculate the performance metrics’ values. The calculated values are used to generate the scaling actions. Reinforcement learning algorithms cope with the auto-scaling task without any a priori knowledge or system model. However, the time for the reinforcement learning methods to converge to an optimal policy can be unfeasibly long. Control theory creates a reactive or a predictive controller to automatically adjust the required resources to the cloud service’s demand. Readers are encouraged to see [4] for more details about the different decision making approaches.

### 2.3.4 Performance Indicator

Performance indicators are sampled by the monitoring component (see Figure 1-2) to measure the performance of the cloud service. Performance of the cloud service can be measured by various indicators that are retrieved from the different software tiers. Authors in [38] propose the following list of the performance indicators:

- From the hardware: CPU utilization, disk access, network interface access, memory usage.
- From the operating system: CPU time, page faults, real memory
- From the load balancer: size of the request queue length, session rate, number of the current sessions, transmitted bytes, number of the denied requests, number of errors, number of the incoming requests (i.e., workload)

**Table 2-2. The existing auto-scaling mechanisms**

| Ref. | Scaling method | Target tiers | Prediction method | Decision making | Performance indicator | Workload pattern |
|---|---|---|---|---|---|---|
| [2] | Predictive | BT | ANN, LR[1], SVM[2] | Static threshold | Response time, CPU utilization, Throughput | Periodic |
| [39] | Predictive | BT[3] | MA[4] | Deadline calculation | CPU usage | Unpredictable |
| [40] | Predictive | BT | DES[5] | Dynamic thresholds | System utilization + response time | None[6] |
| [41] | Predictive | DT[7] | ANN[8] | Static thresholds | Response time + throughput | Unpredictable |
| [42] | Hybrid | All | Ensemble | Capacity calculation | CPU utilization | None |
| [43] | Reactive | All | None | Scheduling | Response time | None |
| [44] | Reactive | BT + DT | None | Capacity calculation | Response time | Periodic |
| [45] | Reactive | BT | None | Static threshold | Turnaround time | None |
| [46] | Reactive | BT | None | Capacity calculation | Response time | Unpredictable |
| [47] | Predictive | BT and DT | ARMA | QNM[9] and control theory | Response time | Periodic |
| [48] | Reactive | BT | EAP[10] | None | Workload | Unpredictable |
| [49] | Predictive | BT | KMP[11] | None | CPU usage | None |
| [50] | Reactive | BT | None | QNM and Threshold based | Response time, CPU utilization | None |
| [51] | Reactive | All | None | Threshold | BT, DT, and network utilization | None |
| [52] | Reactive | BT | None | Static threshold | Response time, CPU utilization | None |
| [53] | Predictive | BT | KSwSVR[12] | None | System utilization | None |

[1] LR: linear regression
[2] SVM: support vector machine
[3] BT: business tier
[4] MA: moving average
[5] DES: double exponential smoothing
[6] None values indicate that the reviewed paper does not provide enough information about the criterion
[7] DT: data tier
[8] ANN: artificial neural networks
[9] QNM: queueing network model
[10] EAP: a prediction framework tailored for specific workload
[11] KMP: string matching technique
[12] KSwSVR: combination of SVR and Kalman smoother

| | | | | | | |
|---|---|---|---|---|---|---|
| [54] | Hybrid | BT | LR | QNM | Workload | Periodic |
| [55] | Reactive | All | None | Threshold + schedule | Not specified | None |
| [56] | Predictive | BT | ANN | Cost Calculations | Workload | Unpredictable |
| [57] | Predictive | BT | ARMA | Static threshold | CPU utilization | None |
| [58] | Predictive | BT | ARMA | QNM | Workload | Unpredictable |
| [59] | Reactive | DT | None | Schedule + threshold | Response time | Growing |
| [60] | Reactive | BT | None | QNM | CPU Utilization | None |

### 2.3.5 Workload Pattern

Workload is the consequence of the end users accessing the cloud service or the consequence of the jobs that need to be handled automatically [32]. Among the existing workload patterns, periodic, growing, and unpredictable patterns are the usual patterns of the cloud services. Section 2.3 provides a comprehensive study on the workload patterns. According to Table 2-2 none of the existing auto-scaling systems provides a comprehensive study on the effect of the workload patterns on the accuracy of the auto-scaling system.

# Chapter 3 - Theoretical Foundation of Machine Learning

To improve the accuracy of the predictive auto-scaling systems, the first step is to investigate the accuracy of the results that are generated by the predictor component. Time-series analysis is the dominant mechanism to carry out the prediction task in the auto-scaling problem domain [4]. The auto-scaling systems periodically sample a performance indicator (such as the incoming workload) at fixed intervals. The result is a time-series $X$ which contains a sequence of the last $s$ observations of the performance indicator's value[4].

$$X = x_t, x_{t-1}, x_{t-2}, \dots, x_{t-s+1} \tag{3-1}$$

The job of the predictor component is to receive the time-series and forecast its future value. Different time series analysis algorithms have been used for the cloud resource provisioning (see Table 2-2). Machine learning is the most suitable mechanism to forecast the future performance condition of a cloud service [2][3]. Machine learning is the study of algorithms which can learn complex relationships or patterns from the empirical data and make accurate decisions [61]. Machine learning can be classified into supervised learning, semi-supervised learning, and unsupervised learning. The supervised learning deduces a functional relationship from the training data that generalizes well to the whole dataset. In contrast, the unsupervised learning has no training dataset. The goal of the unsupervised learning is to discover the relationships between the samples or reveal the latent variables behind the observations [62]. The semi-supervised learning falls between the supervised and the unsupervised learnings by utilizing both of the labeled and the unlabeled data during the training phase [61]. Among the three categories of the machine learning, the supervised learning is the best fit to solve the prediction

problem in the auto-scaling domain [62]. Therefore, this thesis uses the supervised learning prediction approach.

This chapter presents the formal definition of the machine learning process and analyses the risk minimization as the core function of the learning theory. Moreover, mathematical foundation of Artificial Neural Networks (ANNs) and Support Vector Machines (SVMs) are introduced in this chapter. These algorithms (i.e., ANN and SVM) are extensively used in this thesis to carry out the practical experiments.

## 3.1 Formal Definition of the Machine Learning Process

Vapnik [63] describes the machine learning process through the following components:

1. A *generator* of random vectors $x$. The generator uses a fixed but unknown distribution $P(x)$ to independently produce the random vectors.
2. A *supervisor* which is a function that returns an output vector $y$ for every input vector $x$, according to a conditional distribution function $P(y|x)$. The conditional distribution function is fixed but unknown.
3. A *learning machine* that is capable of implementing a set of functions $f(x, w), w \in W$, where $x$ is a random input vector, $w$ is a parameter of the function, and $W$ is a set of abstract parameters that are used to index the set of functions $f(x, w)$ [64].

According to Vapnik's explanation, the goal of the machine learning technique is to select the best available approximation to the supervisor's response. The selection is based on a training set of $l$ independent observations:

$$(x_1, y_1), (x_2, y_2), \dots, (x_l, y_l) \tag{3-2}$$

To choose the best approximation to the supervisor`s response, a loss function $L\big(y, f(x,w)\big)$ should be measured. The loss function calculates the difference between the response $y$ of the supervisor with respect to a given input $x$ and the response $f(x,w)$ that is provided by the learning machine. The value of the loss function is defined as the "functional risk" $R(w)$.

$$R(w) = \int L\big(y, f(x,w)\big) dP(x,y) \tag{3-3}$$

The goal of the learning theory is to minimize the functional risk $R(w)$ over the class of functions $f(x,w), w \in W$. Minimizing the functional risk is equal to finding the learning machine that has the closet prediction values to the supervisor's responses. The problem in minimizing the functional risk is that the joint probability distribution $P(x,y) = P(y|x)P(x)$ is unknown and the only available information is contained in the training set. In other words, only the supervisor's response for the training set is available, and the supervisor's response for the testing data set is unknown.

In the predictive auto-scaling problem domain, the predictor component corresponds to the learning machine of the leaning process. The goal is to find the most accurate predictor which is the learning machine with the minimum functional risk. The components of the formal learning process can be mapped to those of the predictive auto-scaling problem as follows:

- The *input vector x* maps to the workload vector which is randomly generated by a workload generator.
- The *supervisor's response* is analogous to the future workload value which is determined by $P(x,y)$.

- The *independent observations* are equivalent to the training dataset and indicate the historical values of the workload.
- The *learning machine* maps to the predictor component.

In the auto-scaling problem domain, $P(x, y)$ refers to the workload distribution. Suppose that there is a set of the candidate predictor functions $f(x, w), w \in W$ and the goal is to find the most accurate function among them. Given that only the workload values for the training duration are known, the functional risk $R(w)$ cannot be calculated for the candidate predictor functions $f(x, w), w \in W$. Therefore, the most accurate prediction function cannot be found. To address this issue, Empirical Risk Minimization (ERM) and Structural Risk Minimization (SRM) approaches are introduced.

### 3.2 Empirical Risk Minimization

To solve the functional risk problem, Vapnik [63] proposes an induction principle of replacing the functional risk $R(w)$ with an empirical risk function:

$$E(w) = \frac{1}{l}\sum_{i=1}^{l} L\big(y_i, f(x_i, w)\big) \tag{3-4}$$

The induction principle of ERM assumes that the function $f(x_i, w_l^*)$, which minimizes $E(w)$ over the set $w \in W$, results in a risk $R(w_l^*)$ which is close to the minimum $R(w)$. ERM suggests finding the learning machine that minimizes the functional risk over the training dataset (not the whole dataset). Aidu et al. [65] have shown that an evaluation of the robustness of the ERM principle requires to answer the following questions:

1. Does the empirical risk $E(w)$ converge uniformly to the functional risk $R(w)$ over the full set of $f(x, w), w \in W$?

The uniform convergence is defined as

$$Prob\{sup_{w\in W}|R(w) - E(w)| > \varepsilon\} \to 0 \; as \; l \to \infty \tag{3-5}$$

where $sup_{w\in W}|R(w) - E(w)|$ is the supremum of the set of the differences between the functional risk (i.e., $R(w)$) and the empirical risk (i.e., $E(w)$) for different weights in $W$. Equation (3-5) indicates that as the size of the training dataset increases (i.e., $l \to \infty$), the probability of the difference between the functional risk and the empirical risk becomes very small (i.e., the difference between the empirical risk and the functional risk is not greater than a small value $\varepsilon$). Vapnik et al. [66] stress that the uniform convergence in equation (3-5) is a necessary and sufficient condition that shows the consistency of the ERM principle [63]. Therefore, to solve the machine learning problem, if the condition in equation (3-5) is valid, then the prediction function with the minimum ERM is the best approximation to the supervisor's response.

2. What is the convergence rate?

Vapnik et al. in [66] have developed a theory of uniform convergence of the empirical risk to the functional risk. According to that theory, the bounds of the convergence rate are independent of the distribution function $P(x, y)$ and are based on the capacity of the set of the functions that are implemented by the learning machine. The capacity of the learning machine is called VC-dimension (for Vapnik-Chervonenkis dimension). VC-dimension represents the complexity of the learning machine. According to the Vapnik theory of uniform convergence, the convergence rate bounds in the auto-scaling domain are independent of the workload distribution and are based on the complexity (i.e., VC-dimension) of the regression model that is used in the predictor component.

### 3.2.1 VC-dimension

VC-dimension is a complexity metric which measures the expressive power, the richness or the flexibility of a set of functions by assessing how sinuous (i.e., sinuosity - continuously differentiable curve) the functions can be [67]. The formal definition for the VC-dimension of a set of the real functions is [67]:

Let $A \leq f(x,w), w \in \mathrm{W} \leq B$, be a set of real functions that are bounded by constraints A and B (A can be $-\infty$ and B can be $\infty$), where $x$ is the input, $w$ is a parameter of the function, and $W$ is a set of abstract parameters that are used to index the set of functions $f(x,w)$ [64]. The indicator of level $\beta$ of function $f(x,w)$ shows the $x$ values for which function $f(x,w)$ exceeds $\beta$. Function $f(x,w)$ can be described by a set of indicators. Consider the set of the indicators that is [67]:

$$I(x,w,\beta) = \theta\{f(x,w) - \beta\}, w \in \mathrm{W}, \beta \in (\mathrm{A},\mathrm{B}) \tag{3-6}$$

where $\theta(x)$ is the step function.

$$\theta(x) = \begin{cases} 0 & if\ x < 0, \\ 1 & if\ x \geq 0. \end{cases} \tag{3-7}$$

The VC-dimension of a set of the functions $f(x,w), w \in \mathrm{W}$ is defined to be the VC-dimension of the set of the corresponding indicators in equation (3-6) with parameters $w \in \mathrm{W}, \beta \in (\mathrm{A},\mathrm{B})$.

The VC-dimension of a set of indicator functions is the maximum number $h$ of the input vectors which can be shattered[13] in all of the possible $2^h$ ways by using functions in the set. For instance, for the linear decision rules in an $n$-dimensional space the VC-dimension is $n+1$, because the linear decision rules can at most shatter $n+1$ points.

[13] Model $f$ with some parameter vector θ shatters a set of data points ($x_1$,$x_2$,…,$x_n$ ), if for all of the assignments of the labels to the data points there exists a θ such that the model $f$ makes no error evaluating that set of the data points.

### 3.2.2 Uniform Convergence Rates for the ERM

Vapnik's theory of uniformed convergence [66] proves that the VC-dimension of the regression model determines the bounds of the uniform convergence rate. In the binary classification problem, the supervisor's response is either 0 or 1 (i.e., $y \in \{0,1\}$) and $f(x,w), w \in W$ is the class of the indicator functions. In the binary classification problem, the loss function takes only two values: $L\big(y, f(x,w)\big) = 0$ if $y = f(x,w)$, or $L\big(y, f(x,w)\big) = 1$ if $y \neq f(x,w)$. In this case, the functional risk is the error probability that is denoted by $R(w)$. In addition, the empirical risk is the frequency of error in the training data set which is shown by $E(w)$.

According to the theory of uniformed convergence, for a set of the indicator functions with VC-dimension of $h$ the following inequality is valid [63]:

$$Prob\{sup_{w\in W}|R(w) - E(w)| > \varepsilon\} < \left(\frac{2le}{h}\right)^h \exp\{-\varepsilon^2 l\} \tag{3-8}$$

With probability 1 – ŋ for all w ∈ W [63]:

$$R(w) < E(w) + C_0\left(\frac{l}{h}, \eta\right) \tag{3-9}$$

And the maximum value of the confidence interval (i.e., $C_0$) is [63]:

$$C_0\left(\frac{l}{h}, \eta\right) = \sqrt{\frac{h\left(ln\frac{2l}{h}+1\right) - \ln \eta}{l}} \tag{3-10}$$

where $l$ is the size of the training dataset, $h$ is the VC-dimension of the regression model, $e$ is the Euler's number, and $(1 - \eta)$ is the probability of the validity of equation (3-9) for all w ∈ W.

Based on equation (3-9), the error probability of the regression model (i.e., the functional risk) is less than the error frequency in the training dataset (i.e., the empirical

risk) plus the confidential interval. According to equation (3-9) ERM is suitable for training the regression models when the confidence interval is small (i.e., the functional risk is bounded by the empirical risk).

### 3.3 Structural Risk Minimization

In equation (3-9) and equation (3-10), $l$ is the size of the training dataset and $h$ is the VC-dimension (or complexity) of the regression model. Authors in [63] show when $\frac{l}{h}$ is large, the confidence interval becomes small and negligible. In this case, the functional risk is bounded by the empirical risk, which means the probability of the regression model's error in predicting the testing dataset is expected to be less than the probability of the regression model's error in predicting the training dataset.

On the other hand, when $\frac{l}{h}$ is small, the confidence interval is not negligible and even $v(w) = 0$ (i.e., the empirical risk is zero) does not guarantee a small $P(w)$ (i.e., functional risk). In this case to minimize the $P(w)$, the empirical risk $v(w)$ and the confidence interval $C_0\left(\frac{l}{h}, \eta\right)$ should be minimized at the same time. To this end, it is necessary to control the VC-dimension (i.e., complexity) of the regression model. In other words, when the training dataset is complex, the learning machine increases the VC-dimension to shatter the training dataset. By increasing the VC-dimension, the regression model becomes strongly tailored to the particularities of the training dataset and does not perform well on the testing dataset (Overfitting situation).

To control the VC-dimension, Vapnik [63] introduces a nested structure of subsets $S_p = \{f(x, w), w \in W_p\}$ such that:

$$S_1 \subset S_2 \subset \cdots \subset S_n \tag{3-11}$$

The corresponding VC-dimensions of the subsets satisfy:

$$h_1 < h_2 < \cdots < h_n \tag{3-12}$$

Structured Risk Minimization describes a general model of the complexity control and provides a trade-off between the complexity of the hypothesis space and the fitting quality of the training dataset. According to [67] there are four steps to implement the SRM:

1. A class of functions is chosen by using à priori knowledge of the domain. Examples for the class of functions are: polynomials of degree $n$ or artificial neural networks with $n$ hidden layers.
2. The class of functions is divided into a hierarchy of nested subsets. The hierarchy is ordered by complexity.
3. The empirical risk of each of the subsets is calculated.
4. The model with the minimum summation of the empirical risk and the complexity is selected.

## 3.4 Effect of the Workload Pattern on the Prediction Accuracy of the Empirical and the Structural Risk Minimizations

According to Section 2.2, there are three workload patterns in the cloud computing environment: periodic, growing, and unpredictable. The periodic and the growing workload patterns follow a repeatable pattern and their trend and seasonality is predictable. However, the unpredictable workload pattern does not follow a repeatable trend. Therefore, the unpredictable workload pattern is more complex than the growing and the periodic patterns. This suggests using a regression model with a higher VC-

dimension to forecast the unpredictable pattern. From the aforementioned discussions, (i.e., Section 3.1 to Section 3.3), the following hypotheses are proposed:

- **Hypothesis 1:** The SRM approach performs better than the ERM approach in the environments with the periodic and the growing (i.e., predictable) workload patterns.
- **Hypothesis 2:** The ERM approach performs better that the SRM approach in the environments with the unpredictable workload pattern.
- **Hypothesis 3:** Increasing the window sizes does not necessarily have a positive effect on the performance of the structural and the empirical risk minimizations.

As shown in Section 3.3, $\frac{l}{h}$ determines when to use the empirical or the structural risk minimizations. If the training dataset size (i.e., $l$) is static, then for the small values of $h$, $\frac{l}{h}$ fraction is large. In this case, the confidence interval is negligible and the functional risk is bounded by the empirical risk.

In the environments with the predictable workload patterns (i.e., periodic or growing), the training and the testing datasets are not complex. Therefore, in these environments the VC-dimension (i.e., $h$) is small and the empirical and the structural risk minimizations are expected to perform well. However, it is possible that the ERM approach becomes overfitted against the training dataset. The reason is that, although the periodic and the growing workloads follow a repeatable pattern, it is highly probable that some of the data points in the training datasets do not follow the main pattern of the time-series (i.e., noisy data). The noise in the data increases the complexity of the regression model. Increasing the complexity (i.e., VC-dimension) increases the confidence interval as well as the probability of error (see equation (3-9)), which reduces the ERM accuracy.

On the other hand, SRM controls the complexity by neglecting the noise in the data, which reduces the confidence interval. Therefore, in the environments with the periodic and the growing workload patterns the SRM approach is expected to outperform the ERM approach.

The same reasoning applies to the environments with the unpredictable workload pattern. However, in this case there is no distinctive workload trend and none of the data points should be treated as the noise. In the unpredictable environments, the ERM approach increases the VC-dimension to shatter all of the training data points, but the SRM approach controls the VC-dimension to decrease the confidence interval. Therefore, the SRM approach cannot capture the fluctuating nature of the unpredictable workload pattern and trains a less accurate regression model compared to the ERM approach.

In the machine learning domain, window size refers to the input size of the prediction algorithm. Increasing the window size provides more information for the prediction algorithm and is expected to increase the accuracy of the prediction model. However, increasing the input size makes the prediction model more complex. Therefore, because the ERM approach cannot control the complexity of the regression model, increasing the window size increases the VC-dimension of the prediction model, which causes a bigger confidence interval, and reduces the accuracy of the prediction model. On the other hand, since the SRM approach controls the complexity of the regression model, increasing the window size does not affect the prediction model's accuracy. Therefore, increasing the window size does not necessarily increase the accuracy of the prediction models that use the ERM or the SRM approaches.

## 3.5 Artificial Neural Networks

There are different variations of the Artificial Neural Networks (ANNs), such as back-propagation, feed-forward, time delay, and error correction [62]. Multi-Layer Perceptron (MLP) is a feed-forward ANN. This section introduces two versions of MLP that use the empirical and the structural risk minimizations to create the regression model.

### 3.5.1 Multi-Layer Perceptron with Empirical Risk Minimization

A MLP is a network of simple neurons that are called perceptron. A perceptron computes a single output from multiple real valued inputs by forming a linear combination to its input weights and putting the output through a nonlinear activation function. The mathematical representation of the MLP's output is [63]:

$$y = \varphi(\sum_{i=1}^{n} w_i x_i + b) = \varphi(W^T X + b) \tag{3-13}$$

where $W$ denotes the vector of the weights, $X$ is the vector of the inputs, $b$ is the bias, and $\varphi$ is the activation function.

The MLP networks are typically used in the supervised learning problems. Therefore, there is a training set that contains an input-output set similar to equation (3-2). Training of MLP refers to adapting all of the weights and biases to their optimal values to minimize the following equation [63]:

$$E = \frac{1}{l}\sum_{i=1}^{l}(T_i - Y_i)^2 \tag{3-14}$$

where $T_i$ denotes the predicted value, $Y_i$ is the actual value, and $l$ is the training set size. Equation (3-14) is a simplified version of equation (3-4) and represents the ERM approach. Therefore, MLP trains the regression model based on the ERM approach.

### 3.5.2 Multi-Layer Perceptron with Structural Risk Minimization

The general principle of the SRM can be implemented in different ways. According to [67], the first step to implement the SRM is to choose a class of functions with hierarchy of nested subsets ordered by complexity. The authors in [63] suggest three examples of the structures that can be used to build the hierarchy of the neural networks.

1. Structure given by the architecture of the neural networks
2. Structure given by the learning procedure
3. Structure given by the preprocessing

The second structure (i.e., given by the learning procedure) uses "weight decay" to create the hierarchy of the nested functions. This structure considers a set of functions $S = \{f(x, w), w \in W\}$ that are implemented by an ANN with a fixed architecture. The parameters {w} are the weights of the ANN. The nested structure is introduced through $S_p = \{f(x, w), ||w|| \leq C_p\}$ and $C_1 < C_2 < \cdots < C_n$, where $C_i$ is a constant value that defines the ceiling of the norms of the ANN weights. For a convex loss function, the minimization of the empirical risk within the element $S_p$ of the structure is achieved through the minimization of:

$$E(w, \gamma_p) = \frac{1}{l}\sum_1^l L(y_i, f(x_i, w)) + \gamma_p ||w||^2 \tag{3-15}$$

The nested structure can be created by appropriately choosing the Lagrange multipliers $\gamma_1 > \gamma_2 > \cdots > \gamma_n$. According to equation (3-15), the well-known weight-decay procedure refers to the structural risk minimization [63].

Training neural networks with the weight decay approach means that during the training phase, updates of the weights are multiplied by a factor which is slightly less

than 1. This prevents the weights from growing too large. The risk minimization equation for the Multi-Layer Perception with Weight Decay (MLPWD) is:

$$E = \frac{1}{l}\sum_{i=1}^{l}(T_i - Y_i)^2 + \frac{\lambda}{2}\sum_{i=1}^{l} {w_i}^2 \quad (3\text{-}16)$$

The authors in [68] have shown that the conventional weight decay technique can be considered as the simplified version of the SRM in the ANNs. Therefore, in this thesis MLPWD algorithm is used to study the accuracy of the SRM for predicting the workload.

## 3.6 Support Vector Machine

Support Vector Machine (SVM) is a learning machine which implements the SRM principle and has two main categories: Support Vector Classification (SVC) and Support Vector Regression (SVR). This thesis investigates SVR algorithm and evaluates the accuracy of SVR algorithm to be used in the predictor component of the auto-scaling systems. The terms SVM and SVR are used interchangeably throughout this thesis.

Similar to MLP and MLPWD algorithms, the goal of SVR is to find the function $f(x)$ that predicts the future values of the time-series. SVR can be applied to the linear and the non-linear prediction problems. Equation (3-17) and equation (3-18) define the SVR prediction functions for the linear and the non-linear regression applications, respectively:

$$f(x) = (w.x) + b \quad (3\text{-}17)$$

$$f(x) = \big(w.\varphi(x)\big) + b \quad (3\text{-}18)$$

where $w$ is the set of the weights, $b$ is the threshold (or biased), and $\varphi$ is the kernel function.

If the training dataset is not linear, SVR uses a kernel function (i.e., $\varphi$) to map the data to a higher dimension feature space and then performs a linear regression in the higher dimensional feature space. The goal is to find the optimal weights $w$ and threshold $b$. SVR attempts to minimize flatness of the weights (i.e., minimize $||w||$) and minimize the error generated by the estimation process of the values (i.e., the empirical risk) at the same time. Flatness of the weights represents the VC-dimension of the regression model. Therefore, the goal of SVR is to minimize the structural risk.

$$R_{reg}(f) = R_{emp}(f) + \frac{\lambda}{2}\left||w|\right|^2 \tag{3-19}$$

The scale factor $\lambda$ is commonly referred to as the regularization constant or the capacity control term. The empirical risk in equation (3-19) is defined as:

$$R_{emp}(f) = \frac{1}{l}\sum_{i=1}^{l} L(y_i, f(x_i, w)) \tag{3-20}$$

where $l$ is the length of the training dataset, $L$ is the loss function, $y_i$ is the actual value, $f$ is the regression function, and $w$ is the regression function parameter.

There are two common loss functions that are used in the literature for SVM algorithm: $\varepsilon$-intensive loss function and quadratic loss function. The quadratic loss function is typically associated with Least-Squares Support Vector Machines (LS-SVM) [69]. The LS-SVM is suitable for prediction of the large datasets, however the LS-SBM prediction accuracy is less than the SVM accuracy [70]. This thesis explores the $\varepsilon$-intensive loss function. Inserting $\varepsilon$-intensive loss function in equation (3-20) results in equation (3-21) that should be solved to find the optimal weights and to minimize the structural risk.

$$\text{Minimize } \frac{1}{2}||w||^2 + C\sum_{i=1}^{l} L\big(y_i, f(x_i, w)\big) \tag{3-21}$$

where

$$L\big(y_i, f(x_i, w)\big) = \begin{cases} |y_i - f(x_i, w)| - \ \varepsilon & if\ |y_i - f(x_i, w)| \geq \ \varepsilon \\ 0 & otherwise \end{cases} \quad (3\text{-}22)$$

Constant $C$ includes the $\frac{1}{l}$ summation normalization factor, the scale factor $\lambda$ is assumed to be 1, and $\varepsilon$ is tube-size which refers to the precision by which the function is to be approximated. The $C$ value and the $\varepsilon$ value are user defined constraints and typically are computed empirically. SVM uses Lagrange multipliers to find the optimal weights and the bias values. The dual optimization problem which is formed by the Lagrange multipliers is:

$$\text{Maximize } -\frac{1}{2}\sum_{i,j=1}^{l}(\alpha_i - \alpha_i^*)\big(\alpha_j - \alpha_j^*\big)\langle x_i, x_j\rangle - \varepsilon\sum_{i=1}^{l}(\alpha_i - \alpha_i^*) + \ \sum_{i=1}^{l} y_i(\alpha_i - \alpha_i^*)$$

$$\text{Subject to: } \sum_{i=1}^{l}(\alpha_i - \alpha_i^*) = 0 : \ \alpha_i, \alpha_i^* \ \in \ [0, C] \quad (3\text{-}23)$$

In equation (3-23) $\alpha_i$ and $\alpha_i^*$ are the Lagrange multipliers. At the point of the optimal solution the product of the variables and the constraints is zero. Therefore, the approximation of the function $f(x)$ is given as the sum of the optimal weights times the dot products between the data points:

$$f(x) = \ \sum_{i=1}^{l}(\alpha_i - \alpha_i^*)\langle x, x_i\rangle + b \quad (3\text{-}24)$$

The data points that are either in or outside of the $\varepsilon$ tube with non-zero Lagrange multipliers $\alpha$ are defined as Support Vectors.

To carry out a non-linear regression, SVM maps the input space $x_i$ into a higher dimension linear feature space $\varphi(x_i)$ by using a kernel function. The kernel functions that are usually used in SVM are:

- Linear kernel: $\varphi(x) = \ x_i^T x$
- Polynomial: $\varphi(x) = \ (\gamma x_i^T x + r)^d, \gamma > 0$
- Radial Basis Function: $\varphi(x) = \ \exp(-\gamma||x_i - x||^2), \gamma > 0$

- Sigmoid: $\varphi(x) = tanh(\gamma x_i^T x + r), \gamma > 0$

where $\gamma$, $r$ and $d$ are the kernel parameters.

This thesis compares SVM and MLP algorithms to investigate which one suits best for the workload prediction task. Table 3-1 compares SVM and MLP [69].

**Table 3-1. SVM and MLP comparison** [69]

| Prediction Method | Advantages | Challenges |
|---|---|---|
| Multi-Layer Perception | Not model dependent | A large number of free parameters |
| | Not dependent on linear, stationary process | Selection of free parameters usually calculated empirically |
| | Can be computationally efficient | Not guaranteed to converge to optimal solution |
| | | Can be computationally expensive (training process) |
| Support Vector Machine | Not model dependent | Selection of free parameters usually calculated empirically |
| | Not dependent on linear, stationary process | |
| | Guaranteed to converge to optimal solution | |
| | A small number of free parameters | Can be computationally expensive (training process) |
| | Can be computationally efficient | |

According to Table 3-1 SVM and MLP have similar advantages and challenges, except that SVM guarantees convergence to the optimal solution, but MLP does not. The reason is that SVM benefits from the SRM, but MLP uses the ERM which causes the overfitting problem. Since, MLPWD uses the SRM to train the regression model, SVM and MLPWD are similar in terms of the advantages and the disadvantages. However, both of the regression models are complex and have different configuration parameters, hence, they cannot be compared mathematically. Therefore, this thesis experimentally compares the regression accuracy of SVM, MLP and MLPWD models. The experiment is described in Chapter 5.

# Chapter 4 - Proposed Predictive Auto-Scaling System

The main goal of this thesis is to increase the accuracy of the predictive auto-scaling systems. Rule-based systems are the most popular auto-scaling systems in the industrial environments [4]. However, according to [4], the rule-based systems suffer from two main shortcomings: a) their reactive nature, and b) the difficulty of selecting a correct set of the parameters.

This thesis investigates the impact of these shortcomings on the accuracy of the rule-based systems, and proposes an auto-scaling system to overcome the aforementioned shortcomings. The proposed auto-scaling system uses the predictive approach to remedy the first shortcoming of the rule-based systems (i.e., the reactive nature). In addition, a genetic algorithm is used to automatically identify the optimal set of the parameters to configure the rule-based systems. Figure 4-1 shows the architecture of the proposed auto-scaling system.

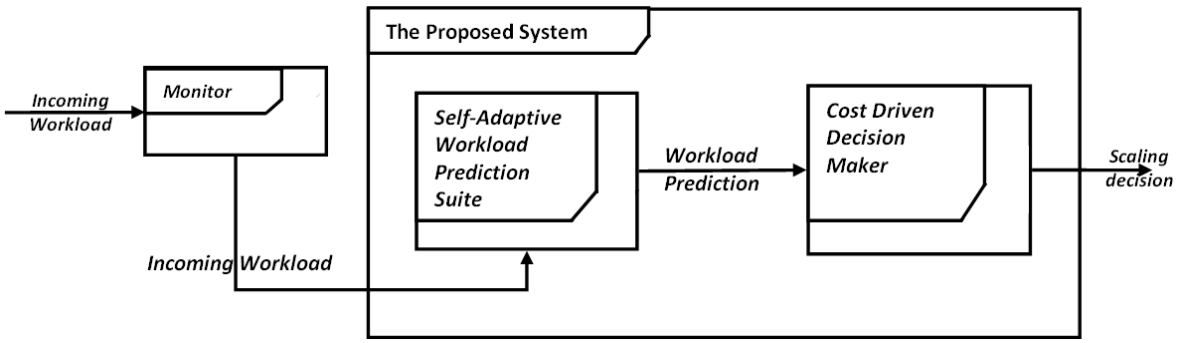


**Figure 4-1. Architecture of the proposed predictive auto-scaling system**

The proposed auto-scaling system consists of a *"self-adaptive prediction suite"* and a *"cost driven decision maker"*. This chapter introduces these components and evaluates the accuracy of the proposed predictive auto-scaling system. The proposed system

accommodates the business tier of the cloud services. However, the proposed auto-scaling mechanism can be applied to scale the database tier, as well.

### 4.1 Self-Adaptive Workload Prediction Suite

The reactive nature is the first issue of the rule-based auto-scaling systems. According to [2], it takes between 5 and 15 minutes for a VM to be started and become fully functional (i.e., the VM boot-up time). The reactive nature of the rule-based systems doesn't allow them to generate the scaling actions ahead of time, which causes the SLA violations during the VM boot-up time. Section 5.5 presents the experimental results to investigate the impact of the VM boot-up time on the accuracy of the auto-scaling systems. According to the results in Section 5.5, the VM boot-up time causes excessive SLA violations and increases the resource cost. To solve the VM boot-up time issue, a predictor component is used in the proposed auto-scaling system. The predictor component forecasts the future workload of the cloud service, and enables the decision maker to generate the scaling actions ahead of time.

Researchers have already used prediction methods to alleviate the reactive nature of the rule-based systems (see Section 2.3). However, the existing predictive auto-scaling systems use only one prediction method to forecast the future performance condition of the cloud service. Section 3.4 hypothesizes that the different prediction algorithms are suitable for the different workload pattern. This hypothesis is proved mathematically in Section 3.4. Furthermore, Section 5.3 experimentally confirms the hypothesis. Therefore, to increase the prediction accuracy, the proposed predictive auto-scaling system identifies the pattern of the incoming workload and chooses the prediction algorithm based on the

detected pattern. In this section a high level design of a self-adaptive workload prediction suite is presented. The self-adaptive suite automatically chooses:

- The MLP prediction model to forecast the workload in the environments with the unpredictable workload pattern
- The MLPWD prediction model to forecast the workload in the environments with the periodic workload pattern
- The SVM prediction model to forecast the workload in the environments with the growing workload pattern.

Readers are encouraged to study Section 3.4 and Section 5.3 for more details about the reasons to choose the aforementioned prediction models and their corresponding environments.

The goal of a classical autonomic system (cf., Figure 4-2) is to make the computing systems self-managed. This is motivated by increasing complexity in the software systems due to the objects change, environmental influence, and ownership cost of the software [71][72]. One of the major requirements for operating a computer system with a minimal human intervention is that the computer system must be able to monitor its activities at runtime based on its internal situation. In addition, the computer system must be able to keep the knowledge about its past, present, and future goals.

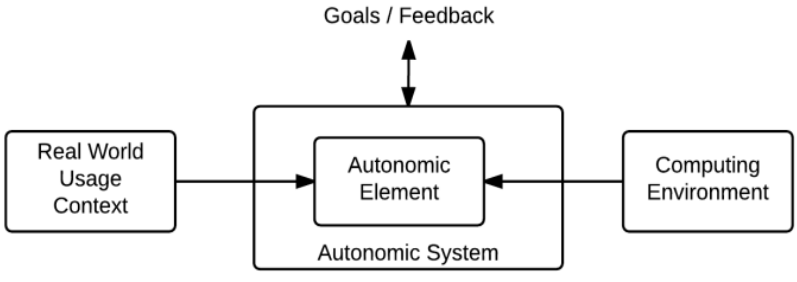


**Figure 4-2. A classical autonomic system**

A self-managed system (i.e., an autonomic system) must be attentive to its internal operations and must adapt to the "behavior" changes in order to produce the "future" actions. A typical autonomic system [73][74][75] consists of a *"context"* autonomic element, and a computing environment (see Figure 4-2). In addition, the autonomic system receives the goals and gives the feedback to an external environment. The autonomic system depends on the autonomic element which is an executable function of the system that exhibits the autonomic properties. An autonomic element regularly senses the sources of change by using "sensors" or "reasons". In this thesis, a reason (or a sensor) is a change in the workload pattern. (cf. Figure 4-3). In some cases an autonomic element can be a complete system. In the auto-scaling domain, the autonomic element and the autonomic system are the same.

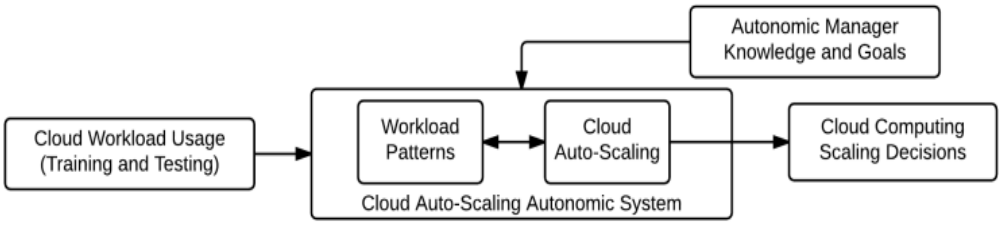


**Figure 4-3. The cloud workload prediction autonomic system**

To create the self-adaptive prediction suite, the classical autonomic system architecture is adapted to the cloud auto-scaling autonomic system architecture. The mapping between the two systems is shown in Figure 4-4.

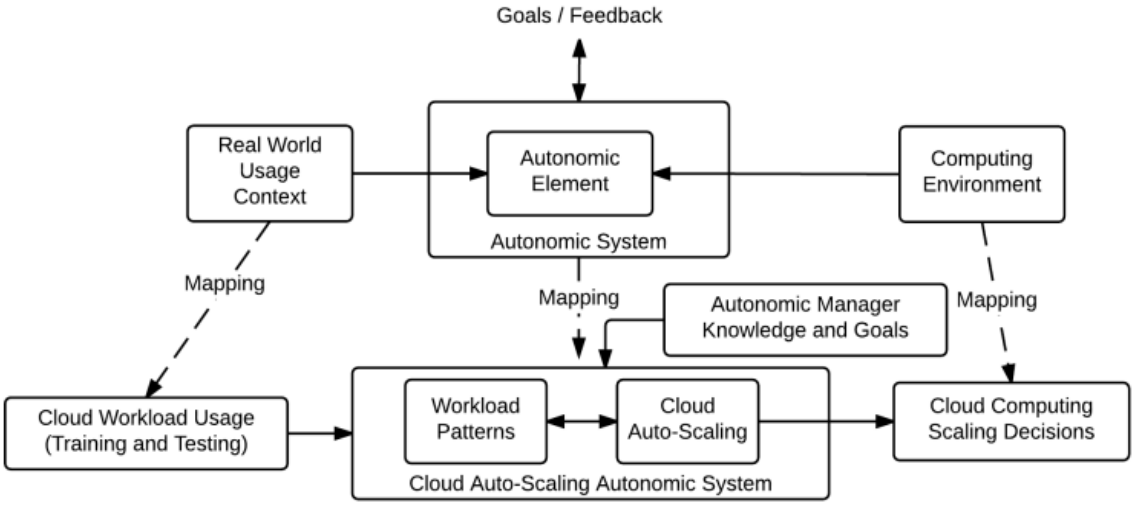


**Figure 4-4. Mapping the classical autonomic system to the cloud workload prediction autonomic system**

The cloud auto-scaling architecture consists of a cloud workload context element; a cloud auto-scaling autonomic system which includes the meta-autonomic elements (the workload pattern and the cloud auto-scaling); and a cloud computing scaling decisions element. In addition, an element for the autonomic manager, knowledge, and goals is added to the architecture.

The *cloud workload usage* represents the *real world usage context* while the *cloud computing scaling decisions* represents the *computing environment* context. It is important to note that an autonomic system always operates and executes within a context. The context in general is defined by the environment as well as the runtime behavior of the system. The purpose of the autonomic manager is to apply the domain specific knowledge which is linked to the cloud workload pattern and apply the appropriate predictor algorithm (cf. Figure 4-5) to predict the future workload. The cloud autonomic manager is constructed around the analyze/decide/act control loop. Figure 4-5 shows the projection of the cloud auto scaling autonomic element.

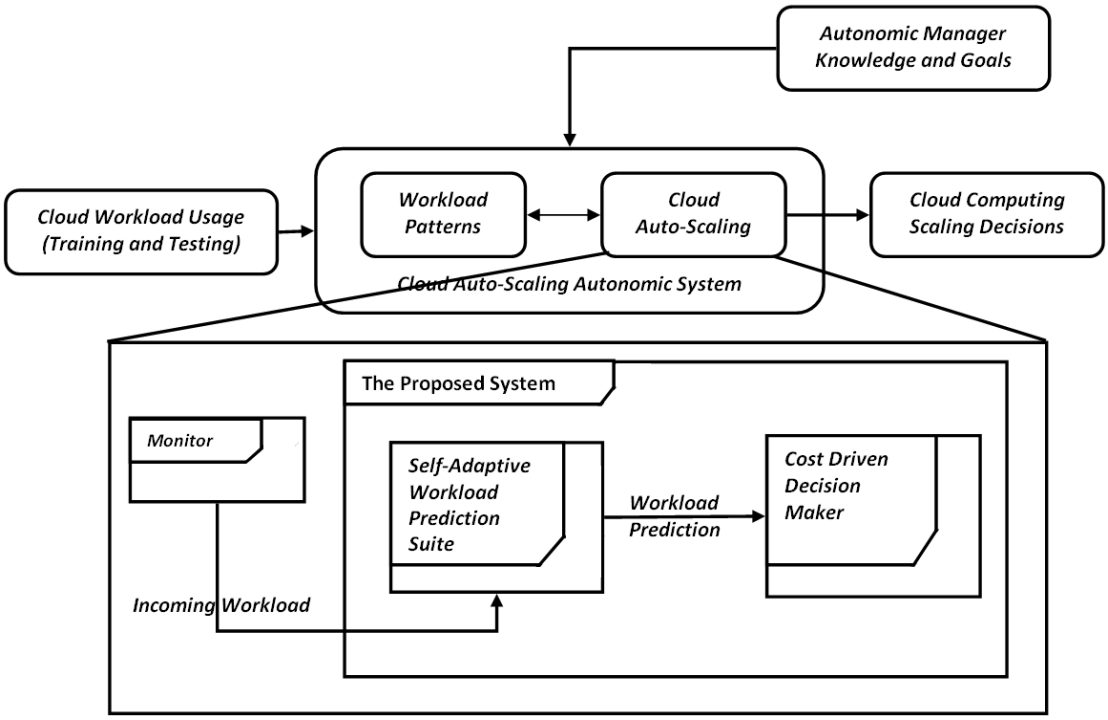


**Figure 4-5. Projection of the cloud auto-scaling autonomic element**

The proposed prediction suite identifies the pattern of the incoming workload and chooses the most accurate prediction algorithm based on the workload pattern. To identify the pattern, the proposed prediction suite decomposes the incoming workload to its components by using Loess package of the R software suite. The Loess component decomposes a workload to its *seasonal*, *trend*, and *remainder* components. If the workload has strong seasonal and trend components which repeat at fixed intervals then the workload has periodic pattern. If the trend of the component is constantly increasing or decreasing then the workload has growing pattern. Otherwise the workload has unpredictable pattern.

The cloud auto scaling autonomic elements (i.e., the workload patterns and the predictor component of the cloud auto-scaling) are designed such that the architecture can be implemented by using the strategy design pattern (cf. Figure 4-6). The strategy design pattern consists of a *strategy* and a *context*. In the cloud auto-scaling domain, the predictor is the strategy and the workload pattern is the context. In general the strategy and the context interact to implement the chosen algorithm. A context passes all of the data (i.e., the workload pattern) that is required by the algorithm to the strategy. In the cloud auto-scaling domain, the context passes itself as an argument to the strategy. This lets the strategy to call on the appropriate sub-context. The way this works is that the context determines the workload pattern and passes its pattern interface to the strategy's interface. The strategy then uses the interface to invoke the appropriate algorithm based on the workload pattern interface. All of this process is done automatically at runtime and this is what makes the auto scaling system to be an autonomic system. A careful examination of the strategy design pattern shows that the context is designed by using the *template method* design pattern (cf. Figure 4-6). The intent of the template method design pattern is to define the skeleton of an algorithm (or function) in an operation to defer some of the steps to the subclasses. In the generic strategy design pattern, the context is simply an abstract class with no concrete subclasses. We have modified this by using the template method to introduce the concrete subclasses to represent the different workload patterns and to implement the workload pattern context as an autonomic element. This way, the cloud workload pattern is determined automatically and the pattern interface is passed on to the predictor element which then automatically invokes the appropriate prediction algorithm for the workload pattern.

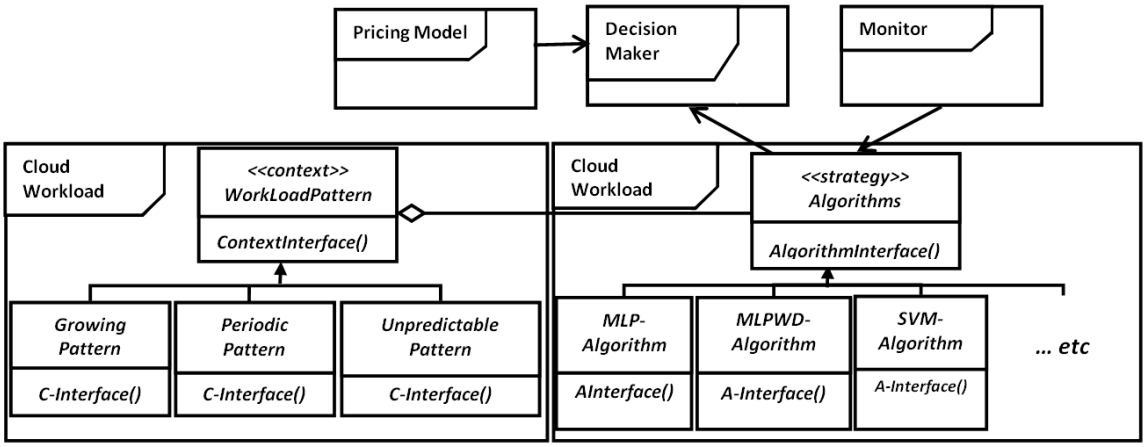


**Figure 4-6. The design of the autonomic elements (the cloud workload pattern and the predictor) by using the strategy design pattern**

It should be noted that, although increasing the prediction accuracy improves the accuracy of the final auto-scaling system, the exiting cloud decision makers are not highly sensitive to the prediction results (see Section 5.4). The experiment that is presented in Section 5.4 investigates the sensitivity of a rule-based decision maker to the different prediction results and concludes that although increasing the prediction results is beneficiary, to improve the total auto-scaling accuracy more precise decision makers are needed.

## 4.2 Cost Driven Decision Maker

The difficulty of selecting a correct set of the parameters is one of the shortcomings of the rule-based decision makers [4]. The correct set of the parameters forms the most accurate rule-based decision maker. The accuracy of the decision maker is defined by the cost that is incurred by the cloud clients, which consists of the SLA violation cost and the resource cost (see Section 1.4). The most accurate rule-based system results in the least

auto-scaling cost. Therefore, the correct set of the parameters is the parameter set which causes the minimum auto-scaling cost.

A highly accurate decision maker prevents the SLA violations as well as reduces the resource costs. However, it is not possible to minimize the number of the SLA violations and the resource cost at the same time. Providing more infrastructural resources reduces the number of the SLA violations, but results in an excessive resource cost. On the other hand, releasing the infrastructural resources saves the resource cost, but increases the number of the SLA violations. Therefore, the decision maker job is to find the balance point between the SLA violations and the resource cost (i.e., the cost/performance trade-off). The optimum solution to this trade-off varies for the different cloud clients. The smaller businesses that don't have too many end users, such as startup companies, usually prefer to reduce the resource cost, while the bigger businesses that have many end users, such as eBay or Netflix, prefer to minimize the SLA violations. Therefore, the cloud client's cost preference is one of the factors that should be considered by the decision makers to solve the cost/performance trade-off. This thesis proposes a cost driven decision maker which finds the correct parameters to configure the rule-based systems. The research question of this section is:

"*How to configure a rule-based decision maker to minimize the total auto-scaling cost based on the cloud clients' cost preferences?"*

In the rule-based auto-scaling, the number of the leased VMs varies according to a set of the scaling rules. A scaling rule has two parts: the condition and the action to be executed when the condition is met. The condition part of a scaling rules uses one or more performance indicator(s), such as the average response time or the average

workload. A performance indicator has an upper (i.e., *thrU*) and a lower (i.e., *thrL*) threshold. If the scaling condition is met for a given duration (i.e., *durU* or *durL*) then the corresponding action will be triggered. After executing a scale action, the decision maker stops itself for a small cool-down period which is defined by *inU* or *inL*. For example, let $x$ be a given performance indicator, a sample of a scale out rule is [4]:

$$if\ x > thrU\ for\ durU\ seconds\ then\ n = n + s\ and\ do\ nothing\ for\ inU\ seconds \quad (4\text{-}5)$$

Some of the researchers have proposed additional parameters to improve the auto-scaling accuracy. For instance the proposed method in [51], uses two upper and two lower thresholds to determine the trend of the performance indicator. Considering the trend of the performance indicator helps to predict the future performance of the cloud service and generate the scaling actions ahead of time. Although the proposed method in [51] generates the scaling actions ahead of time, it does not have a better accuracy compared to the traditional rule-based systems [4].

A typical rule-based decision maker has six configuration parameters: the upper threshold (*thrU*), the lower threshold (*thrL*), the upper scaling duration (*durU*), the lower scaling duration (*durL*), the upper cool-down duration (*inU*), and the lower cool-down duration (*inL*). According to the experimental results in Section 5.7, all of the configuration parameters can influence the SLA violations and the resource cost of the rule-based decision makers. However, some of the parameters, such as the *thrU* and the *durU*, have more influence on the cost factors. In addition, the experimental results show that to decrease the resource cost, the cloud clients should:

1. Increase the *thrU*
2. Increase the *thrL*
3. Increase the *durU*
4. Decrease the *durL*

5. Increase the *inU*
6. Decrease the *inL*

On the other hand, to reduce the SLA violations, the cloud clients should:

1. Decrease the *thrU*
2. Decrease the *thrL*
3. Decrease the *durU*
4. Increase the *durL*
5. Decrease the *inU*
6. Increase the *inL*

Readers are encouraged to see Section 5.7 for more details.

The research problem of this section is to find the best value for each of the parameters such that the configured rule-based decision maker minimizes the final auto-scaling cost. Since the domain of the valid values for each of the parameters is known, the universal set of the possible solutions can be created where each solution is a valid combination of the parameters. Then to find the optimal solution (the optimal solution is a configuration set with the minimal scaling cost), the search space (i.e., the universal set of the solutions) is traversed and the solution with the least auto-scaling cost is found. The main challenge here is to measure the auto-scaling cost of each of the solutions. Similar to the functional risk problem of the learning machines in Section 3.2, suppose there is a set of the candidate decision-maker functions $f(x, w)$ and the goal is to find the most accurate function among them. The parameter $x$ is the incoming workload and the parameter $w$ is the set of the configuration parameters. Similar to the supervised learning mechanism (see Chapter 3), the experimental dataset is split to the training and the testing datasets, and the ERM is applied to identify the most accurate decision maker. According to the ERM principle, it can be hypothesized that the decision maker with the least auto-

scaling cost over the training dataset minimizes the auto-scaling cost over the testing dataset. The experimental results in Section 5.9 confirm this hypothesis.

According to the results of Section 5.9, to measure the auto-scaling cost of a given solution, a decision maker is configured with the parameters of that solution, and an auto-scaling simulation is run over the training dataset to calculate the total auto-scaling cost of the decision maker. In this thesis an in-house simulation package is used to carry out the simulations (see 0). Based on the simulation result, measuring the auto-scaling cost of a given solution averagely takes five minutes. Therefore, for a search space with 100 possible solutions, it takes 500 minutes (more than eight hours) to traverse the search space and find the optimum solution.

For example, assume that in an auto-scaling environment the CPU utilization is considered as the performance indicator. Since the CPU utilization value is always between 0 and 100, the upper threshold can take 100 different values. In addition, the lower threshold can take any value greater than zero and less than the upper threshold. Moreover, suppose that the *inU*, the *inL*, the *durU*, and the *durL* take any values between 0 and 5 minutes. In this environment, the universal set includes 6,413,904 valid solutions. Given that measuring the total cost of a solution takes 5 minutes, traversing the whole search space takes 32,069,520 minutes (i.e., more than 61 years), which is infeasible to perform. This section proposes a genetic algorithm to find an optimal solution within the search space in a shorter time.

### 4.2.1 The Reason to Choose Genetic Algorithm

Section 4.2 models a search problem which its goal is to traverse a given search space and find a set of the configuration parameters which minimizes the rule-based auto-

scaling cost. Since the search space is very big, using the exhaustive search algorithms to find the optimum solution is infeasible (see the example given in Section 4.2). This necessitates using a search heuristic algorithm that finds an optimal solution in a shorter time. There are different search heuristic algorithms that can be used in this regard, such as: hill climbing [76], simulated annealing [77], and genetic algorithm. Performance of these algorithms highly depends on the problem domain. However, unlike the genetic algorithm, the hill climbing and the simulated annealing approaches do not use the random search mechanism, therefore the time which is required by them to converge to an optimal solution can be long [78]. Since the search space in the cloud auto-scaling problem is usually big, the hill-climbing and the simulated annealing approaches cannot find an optimal solution in a short time, which makes the genetic algorithm be a better option. In addition, this thesis assumes that there is only one VM type in the IaaS (see Section 1.3). Relaxing this constraint (i.e., assuming that there is more than one VM type in the IaaS) converts the search problem to become a scheduling problem, which is proved to be NP-complete [79]. The scheduling problem can be solved by the genetic algorithm. Therefore, using the genetic algorithm improves the extensibility of the proposed mechanism.

### 4.2.2 Genetic Algorithm

Genetic algorithm applies the evolution principle to provide a robust search technique that finds a high-quality solution in a large search space in polynomial time. A genetic algorithm combines the exploitation of the best solutions from the past searches with the exploration of the new regions of the solution space [79]. Any solution in the search

space is represented by an individual (or chromosomes). A genetic algorithm maintains a population of the individuals that evolves over the generations (or iterations). The quality of an individual in the population is determined by a fitness function. The fitness value indicates how good an individual is compared to the other individuals in the population [79]. A typical genetic algorithm has following steps:

1. Creation of an initial population consisting of randomly generated solutions.
2. Generation of a new offspring by applying the genetic operators which are: selection, crossover and mutation, one after the other.
3. Calculation of the fitness value of the individuals.
4. Repeat steps 2 and 3 until the algorithm converges to an optimal solution.

In order to use the genetic algorithm concept to solve the auto-scaling problem, the representation of the individuals in the population, the fitness function, and the genetic operations should be determined.

#### 4.2.2.1 Individual Presentation

In the rule-based systems, a feasible solution is required to meet the following conditions:

1. The upper threshold should be less than or equal to the upper limit of the performance indicator. For instance, if CPU utilization is the performance indicator, the upper threshold cannot be greater than 100%.
2. The lower threshold should be greater than or equal to the lower limit of the performance indicator. For instance, if CPU utilization is the performance indicator, the lower threshold cannot be less than 0%.

3. The freezing periods should be greater than or equal to zero (i.e., $inU \geq 0$, $inL \geq 0$).
4. The duration parameters should be greater than or equal to zero (i.e., $durU \geq 0$, $durL \geq 0$).
5. The upper threshold should be greater than the lower threshold (i.e., $thrU > thrL$).

An individual in the population is a feasible set of the configuration parameters, and has the following format:

$$< thrU, thrL, durU, durL, inU, inL > \quad (4\text{-}1)$$

Each of the configuration parameters is referred to as a "gene". First step to create a genetic algorithm is to decide about the encoding of the genes. Encoding represents the demonstration of the genes of the individuals in the population. Different encoding types are used in the genetic algorithm, such as: binary, permutation, value, and tree encoding. In the auto-scaling problem, a gene represents a value of one of the configuration parameters of the rule-based decision maker. Therefore the *value encoding* is used in the genetic algorithm that is presented in this thesis.

#### 4.2.2.2 Fitness Function

A fitness function is used to measure the quality of the individuals in the population according to a given optimization objective. Since the goal of the decision maker is to reduce the total cost, the fitness function measures the total auto-scaling cost of the individuals. According to equation (4-2), the total cost of an individual is the summation of its SLA violations cost and its resource cost:

$$C_{total} = C_R + C_{IC} = \sum_{t=o}^{T} \sum_{req=1}^{N} (c_b \times v_{t,req}) + \sum_{t=0}^{T} n_t \times c_{vm} \quad (4\text{-}2)$$

The values of the constant parameters (i.e., $c_b$ and $c_{vm}$) are determined by the cloud clients. See Section 1.4 for more details about the parameters of equation (4-2). The cloud clients can change the fitness function based on their cost preferences.

#### 4.2.2.3 Genetic Operators

The genetic operators manipulate the individuals in the current population and generate a new population of the individuals. A genetic algorithm has three operators:

1. *Selection operator* equates to the survival of the fittest and gives preference to better individuals to pass on their genes to the next generation. Darwin's theory of evolution suggests that the best ones should survive and create a new offspring. In the auto-scaling problem, the selection operation finds the sets of the configuration parameters (i.e., the individuals) with the least cost to pass their configuration parameter values (i.e., the genes) to the next generation.
2. *Crossover operator* represents the mating between the individuals. In the auto-scaling problem, a crossover operation represents how the selected sets of the configuration parameters are combined to generate a new set of the configuration parameters.
3. *Mutation operator* introduces the random modifications in the genes of the individuals. Purpose of the mutation operator is to maintain the diversity in the population and avoid a premature convergence.

The aforementioned operators indicate how the genetic algorithm explores the search space. The following sections represent the genetic operators that are used in the genetic algorithm that is presented in this thesis.

#### 4.2.2.3.1 Selection Operator

The existing methods that are used as the selection operator are: roulette wheel selection, rank selection, steady state selection, and elitism [80].

In the roulette wheel selection, parents are selected according to their fitness value. The better individuals have more chance to be selected as the parents. The roulette wheel selection method simulates a roulette wheel with all of the individuals on it. An individual's portion of the wheel is calculated according to its fitness value. Then a marble is thrown to select an individual. The individuals with better fitness value are selected more frequently. Although the roulette wheel is the most popular selection method in the genetic algorithm, it doesn't perform well when the fitness values of the individuals differ very much. The rank selection solves this issue by first ranking the population and then giving a fitness value to each of the individuals based on their ranking. The rank selection gives a fair chance to the individuals to be selected, but can lead to a slower convergence.

The third selection method is the "steady state selection". Contrary to the rank selection and the roulette wheel selection methods, the steady state selection does not select the parents. The main idea of the steady state is that a big part of the individuals should survive to the next generation. Therefore, in every generation of the steady state process, a few individuals with high fitness value are selected for creating a new

offspring. Then the individuals with less fitness rate are replaced by the new offspring. The rest of the population survives to the new generation.

One problem with the aforementioned selection methods is that there is a big chance that the best individuals are replaced with the new offspring. The elitism selection method prevents this problem by copying the best individuals to the new population. The rest of the selection process is done by one of the aforementioned methods. The elitism can rapidly increase the performance of the genetic algorithm.

In the proposed genetic algorithm of this thesis, a combination of the elitism and the roulette wheel selection methods is used. This combination is simple to implement and guarantees to keep the individuals with the highest fitness values in the next generation.

#### 4.2.2.3.2 Crossover and Mutation Operators

The type and implementation of the crossover and the mutation operators depend on the encoding of the individuals. In the auto-scaling problem, the *value encoding* (see Section 4.2.2.1) is used. The crossover types for the value encoding are [80]:

- **Single-point crossover:** one crossover point is selected, the chromosome string (i.e., the individual) from the beginning to the crossover point is copied from one of the parents, and the rest is copied from the other parent.
- **Two-point crossover:** two crossover points are selected, the chromosome string from the beginning to the first crossover point is copied from one of the parents, the part from the first crossover point to the second crossover point is copied from the second parent, and the rest is copied from the first parent.

- **Uniform crossover:** the genes are randomly copied from the first or from the second parent.
- **Arithmetic crossover:** some arithmetic operation is performed to make a new offspring.

In the auto-scaling problem, there are some limitations on the values of the genes (see Section 4.2.2.1). For instance, the lower threshold should always be less than the upper threshold. Since the gene values depend to each other, using the single-point or the two-point crossover methods can produce an invalid offspring. Therefore, the uniform crossover method is used in this thesis to create a new generation.

Moreover, for the value encoding, the mutation operator is implemented by adding a small number to the randomly selected genes. In the auto-scaling problem domain one of the genes (i.e., a configuration parameter) is randomly picked and its value is altered.

### 4.2.3 The Proposed Decision Maker Algorithm

The proposed decision maker uses the rule-based mechanism to generate the scaling actions. In addition, a genetic algorithm is used to identify an optimal set of the configuration parameters which minimize the total auto-scaling cost based on the cloud client's cost preferences. This section presents two algorithms. The first algorithm uses the genetic algorithm concept to configure the decision maker. The second algorithm represents how the configured decision maker generates the auto-scaling actions. The genetic algorithm to configure the decision maker is shown in Figure 4-7.

**Inputs:**

- SLA violation cost factor ($c_b$)
- VM hourly rate ($c_{vm}$)

**Output:**

- The optimal parameter set $< thrU, thrL, durU, durL, inU, inL >$

**Steps:**

1. Cloud customer selects $c_b$ and $c_{vm}$ values.
2. The fitness function is formulated based on the selected $c_b$ and $c_{vm}$ values.
3. A random population of $N$ individuals is generated. Every individual is formatted based on equation (4-1).
4. Fitness function (i.e., equation (4-2)) of every individual in the population is calculated.
5. Some of the parents are selected for crossover operation. The number of selected parents is defined by the *crossover rate*.
6. Uniform crossover method is applied to generate two new offspring.
7. Few individuals are randomly selected for mutation operation. The number of genes that are modified during the mutation is defined by the *mutation rate*.
8. Elitism selection method is used to remove the individuals with least fitness value from the population and copy the remaining individuals to the next generation. The number of removed individuals is equal to the number of new offspring, which is defined by the crossover rate.
9. If the experiment duration is over, then stop the algorithm. Otherwise, go to step 4.

**Figure 4-7. The genetic algorithm to configure the decision maker**

To fully implement the algorithm of Figure 4-7, an appropriate *population size*, a *crossover rate*, a *mutation rate*, and a *stop condition* should be identified. These parameters are problem specific, and there is no best global value for these parameters. Section 5.8 presents a set of experiments which find a local optimum *population size*,

*crossover rate*, *mutation rate*, and *stop condition* for the rule-based auto-scaling systems.

The decision maker algorithm to generate the scaling actions is shown in Figure 4-8.

**Inputs:**

- Incoming workload prediction ($\lambda$)
- Average service demand of each request on the VMs ($D$)
- Number of rented VMs
- Start time of each rented VM

**Output:**

- Scaling actions: scale out, scale in, no scaling

**Steps:**

1. Decision maker receives the average service time, number of VMs, and start time of each VM from the IaaS.
2. Decision maker receives the incoming workload prediction from the predictor.
3. Decision maker calculates the *ceiling capacity* (c.f. equation (4-3)) and the *floor capacity* (c.f. equation (4-6)) of the underlying IaaS.
4. If the predicted incoming workload is greater than the ceiling capacity for durU duration (i.e., scale out condition is valid), then decision maker generates scale out action, inhabits for inU duration, and then goes to step 2.
5. If the predicted incoming workload is less than the floor capacity for durL duration and *smart kill* condition applies (i.e., scale in condition is valid) then decision maker generates scale in action, inhabits for inL duration, and then goes to step 2.
6. If neither scale out condition nor scale in condition is valid, then decision maker generates no scale action and goes to step 2.

**Figure 4-8. The decision maker algorithm**

The decision maker algorithm that is shown in Figure 4-8 uses the *ceiling capacity* and the *floor capacity* concepts. The ceiling capacity ($Cap_{ceiling}$) refers to the maximum number of the requests that can be handled by the cloud service, and is calculated as:

$$Cap_{ceiling} = n \times CapU_{vm} \tag{4-3}$$

Where, $n$ is the number of the provisioned VMs and $CapU_{vm}$ is the upper threshold of the number of the requests that can be handled by each of the VMs per second. According to the QNM theory, utilization of a VM is calculated as [81]:

$$U_{VM}(\lambda) = \lambda \times D_{VM} \tag{4-4}$$

Where, $\lambda$ is the incoming workload and $D_{VM}$ is the service demand of the requests on the VM. Given that $D_{VM}$ is static during the experiments (i.e., the service demand of the requests does not change), the maximum number of the requests that can be handled by a VM (i.e., $CapU_{vm}$) is:

$$\lambda_{max} = CapU_{vm} = \frac{U_{VM_{MAX}}}{D_{VM}} = \frac{thrU}{D_{VM}} \tag{4-5}$$

The floor capacity ($Cap_{floor}$) represents the minimum capacity of the cloud service:

$$Cap_{floor} = n \times CapL_{vm} \tag{4-6}$$

where, $CapL_{vm}$ is the lower threshold of the number of the requests that can be handled by a VM per second, and is calculated as:

$$\lambda_{min} = CapL_{vm} = \frac{U_{VM_{MIN}}}{D_{VM}} = \frac{thrL}{D_{VM}} \tag{4-7}$$

The workload should always be bounded by the ceiling and the floor capacities. If the workload exceeds the ceiling capacity (i.e., the under-provisioning condition), then a SLA violation occurs. In addition, if the workload is less than the floor capacity (i.e., the over-provisioning condition) then, an excessive amount of VMs is being used.

In addition, according to the fifth step of the decision maker algorithm in Figure 4-8, the decision maker scales in the cloud service only if the *smart kill* condition is valid. Some of the researchers (e.g., [82][83]) have proposed the smart kill idea, which suggests not to kill the VMs before they are used for a full hour. The reason is that the IaaS providers round up the partially used hours to the full hours and charge the cloud clients for a full hour VM usage, even if the VM is being used for less than an hour. According to the experimental results in Section 5.6 the smart kill reduces the total number of the SLA violations as well as the resource cost. Therefore, the smart kill is used in the proposed decision maker of this thesis. Readers are encouraged to see Section 5.6 for more details about the effect of the smart kill on the auto-scaling cost factors.

## 4.3 Evaluation

Section 4.1 and Section 4.2 present the components of the proposed predictive auto-scaling system. This section evaluates the proposed predictive auto-scaling system. Section 4.3.1 investigates the functionality of the proposed decision maker and verifies whether the proposed decision maker is able to adjust the configuration parameters properly according to the cloud $_{\text{clieVMnt's}}$ $_{\text{c}}$ost preference. Furthermore, Section 4.3.2 evaluates the predictive auto-scaling system and measures its accuracy against the Amazon auto-scaling system.

### 4.3.1 Evaluation of the Cost Driven Decision Maker

The proposed cost driven decision maker identifies the configuration parameters that minimize the total auto-scaling cost. This section experimentally evaluates whether the proposed algorithm is able to identify a local optimum configuration.

**Table 4-1. The genetic algorithm behavior in an environment with the periodic workload**

| Iteration no. | $C_R$ | $C_{SLA}$ | No. of rented VMs | No. of SLA breaches | Local Optimum Configuration <thrU, thrL, durU, durL, inU, inL> |
|---|---|---|---|---|---|
| 1 | 1 | 1 | 157 | 4 | <92, 68, 0, 3, 1, 0> |
| 2 | 1 | 5 | 157 | 4 | <92, 68, 0, 3, 1, 0> |
| 3 | 1 | 10 | 171 | 2 | <90, 65, 0, 4, 1, 0> |
| 4 | 1 | 15 | 189 | 1 | <82, 59, 0, 4, 0, 1> |
| 5 | 1 | 20 | 192 | 0 | <80, 55, 0, 4, 0, 1> |
| 6 | 1 | 25 | 192 | 0 | <80, 55, 0, 4, 0, 4> |
| 7 | 1 | 30 | 192 | 0 | <80, 55, 0, 4, 0, 4> |
| 8 | 5 | 1 | 131 | 59 | <96, 49, 4, 0, 3, 0> |
| 9 | 10 | 1 | 126 | 84 | <97, 56, 4, 0, 3, 0> |
| 10 | 15 | 1 | 124 | 129 | <97, 88, 4, 0, 3, 0> |
| 11 | 20 | 1 | 119 | 143 | <97, 88, 4, 0, 3, 0> |
| 12 | 25 | 1 | 119 | 156 | <97, 88, 4, 0, 5, 0> |
| 13 | 30 | 1 | 119 | 156 | <97, 88, 4, 0, 5, 0> |

To this end, an experiment is carried out to examine the proposed algorithm's behavior in the environments with the different workload patterns and the different client's cost preferences. Table 4-1, Table 4-2, and Table 4-3 show the cost driven algorithm's results in the environments with the periodic, growing, and unpredictable workload patterns, respectively. (in the following table $C_R$ and $C_{SLA}$ represent the resource and the SLA violation costs, respectively)

As shown in the results, the proposed algorithm finds local optimum solutions in accordance with the $\frac{C_{SLA}}{C_{VM}}$ fraction. When $\frac{C_{SLA}}{C_{VM}}$ is increased (i.e., the cloud client prefers to reduce the SLA violation cost – iterations 1 to 7), the algorithm decreases the upper and the lower thresholds which increases the spare capacity of the provisioned VMs. This helps the cloud service to tolerate the unforeseen workload surges. Reducing the upper and the lower thresholds causes less SLA violations but increases the resource cost. On the other hand, decreasing $\frac{C_{SLA}}{C_{VM}}$ fraction (i.e., the cloud client prefers to decrease the

resource cost – iterations 8 to 13) results in configurations with a higher thresholds, which expedites the scale in actions and decreases the resource cost. Increasing the upper threshold cause the cloud service to fully use the provisioned VMs' capacities which makes the cloud service to become prone to more SLA violations.

**Table 4-2. The genetic algorithm behavior in an environment with the growing workload**

| Iteration no. | $C_R$ | $C_{SLA}$ | No. of rented VMs | No. of SLA breaches | Local Optimum Configuration <thrU, thrL, durU, durL, inU, inL> |
|---|---|---|---|---|---|
| 1 | 1 | 1 | 127 | 1 | <95, 53, 0, 3, 0, 0> |
| 2 | 1 | 5 | 127 | 1 | <95, 53, 0, 3, 0, 0> |
| 3 | 1 | 10 | 153 | 0 | <80, 29, 0, 4, 0, 4> |
| 4 | 1 | 15 | 153 | 0 | <80, 29, 0, 4, 0, 4> |
| 5 | 1 | 20 | 153 | 0 | <80, 29, 0, 4, 0, 4> |
| 6 | 1 | 25 | 153 | 0 | <80, 29, 0, 4, 0, 4> |
| 7 | 1 | 30 | 153 | 0 | <80, 29, 0, 4, 0, 4> |
| 8 | 5 | 1 | 96 | 80 | <96, 70, 4, 0, 3, 2> |
| 9 | 10 | 1 | 96 | 80 | <96, 70, 4, 0, 3, 2> |
| 10 | 15 | 1 | 96 | 80 | <96, 70, 4, 0, 3, 2> |
| 11 | 20 | 1 | 90 | 91 | <97, 65, 4, 0, 4, 0> |
| 12 | 25 | 1 | 88 | 94 | <99, 69, 4, 0, 4, 0> |
| 13 | 30 | 1 | 88 | 94 | <99, 69, 4, 0, 4, 0> |

**Table 4-3. The genetic algorithm behavior in an environment with the unpredictable workload**

| Iteration no. | $C_R$ | $C_{SLA}$ | No. of rented VMs | No. of SLA breaches | Local Optimum Configuration <thrU, thrL, durU, durL, inU, inL> |
|---|---|---|---|---|---|
| 1 | 1 | 1 | 155 | 2 | <96, 66, 0, 2, 2, 4> |
| 2 | 1 | 5 | 155 | 2 | <96, 66, 0, 2, 2, 4> |
| 3 | 1 | 10 | 153 | 1 | <96, 57, 0, 2, 0, 4> |
| 4 | 1 | 15 | 158 | 0 | <92, 47, 0, 3, 0, 4> |
| 5 | 1 | 20 | 158 | 0 | <92, 47, 0, 3, 0, 4> |
| 6 | 1 | 25 | 158 | 0 | <92, 47, 0, 3, 0, 4> |
| 7 | 1 | 30 | 158 | 0 | <92, 47, 0, 4, 0, 4> |
| 8 | 5 | 1 | 121 | 106 | <97, 73, 4, 0, 4, 1> |
| 9 | 10 | 1 | 112 | 128 | <97, 79, 4, 0, 4, 1> |
| 10 | 15 | 1 | 105 | 188 | <97, 90, 4, 0, 3, 1> |
| 11 | 20 | 1 | 96 | 201 | <99, 90, 4, 0, 3, 1> |
| 12 | 25 | 1 | 96 | 201 | <99, 90, 4, 0, 3, 1> |
| 13 | 30 | 1 | 96 | 201 | <99, 90, 4, 0, 3, 1> |

The experimental results confirm that the proposed cost driven decision maker is able to find the configuration parameters that reduce the total cost based on the cloud clients' preferences.

### 4.3.2 Evaluation of the Proposed Predictive Auto-Scaling System

This thesis proposes a self-adaptive prediction suite in Section 4.1, as well as a cost driven decision maker in Section 4.2. This section puts these two components (i.e., the self-adaptive predictor and the cost driven decision maker) together and evaluates the resulted predictive auto-scaling system. The proposed system is evaluated against the Amazon auto-scaling system, which is a well-known and popular auto-scaling system in the industrial environments [4]. To conduct the evaluation, the Amazon auto-scaling system's behavior is simulated and its resulted auto-scaling cost is compared with the cost of the proposed predictive auto-scaling system.

#### 4.3.2.1 Amazon Auto-Scaling System Details

The Amazon auto-scaling system uses different approaches to generate the scaling actions [84]. In this thesis, the proposed predictive auto-scaling system is compared against the Amazon auto-scaling policy approach. The Amazon auto-scaling policy uses the "*scaling based on the metrics*" technique to generate the scaling actions [85]. In the "*scaling based on the metrics*" technique, the cloud client defines the policies that are similar to equation (4-5). The Amazon auto-scaling system uses the defined policies to generate the scaling actions. The Amazon policy scaling has similar parameters as the proposed decision maker (i.e., *thrU, thrL, durU, durL, inU,* and *inL*). In addition,

Amazon lets clients to decide about the amount of the resources that should be added/removed to/from the underlying IaaS. The Amazon auto-scaling system and the proposed decision maker of this thesis both work based on the rule-based mechanism, and have the same list of the configuration parameters. Therefore, the same parameter values were used to configure both of the auto-scaling systems. Readers are encouraged to see [85] from more details about the Amazon scaling policies.

#### 4.3.2.2 Evaluation Results

In the simulation, three different workload trace files were used to represent the three dominant workload patterns in the IaaS environment (i.e., the periodic, the growing, and the unpredictable). According to the experimental results in Section 5.2, 60% of the workload trace files were used to train the predictor component. The same training trace files were used to identify a local optimum set of the configuration parameters by the cost driven genetic algorithm (see Section5.10). Figure 4-9, Figure 4-10, and Figure 4-11 show the comparison results.

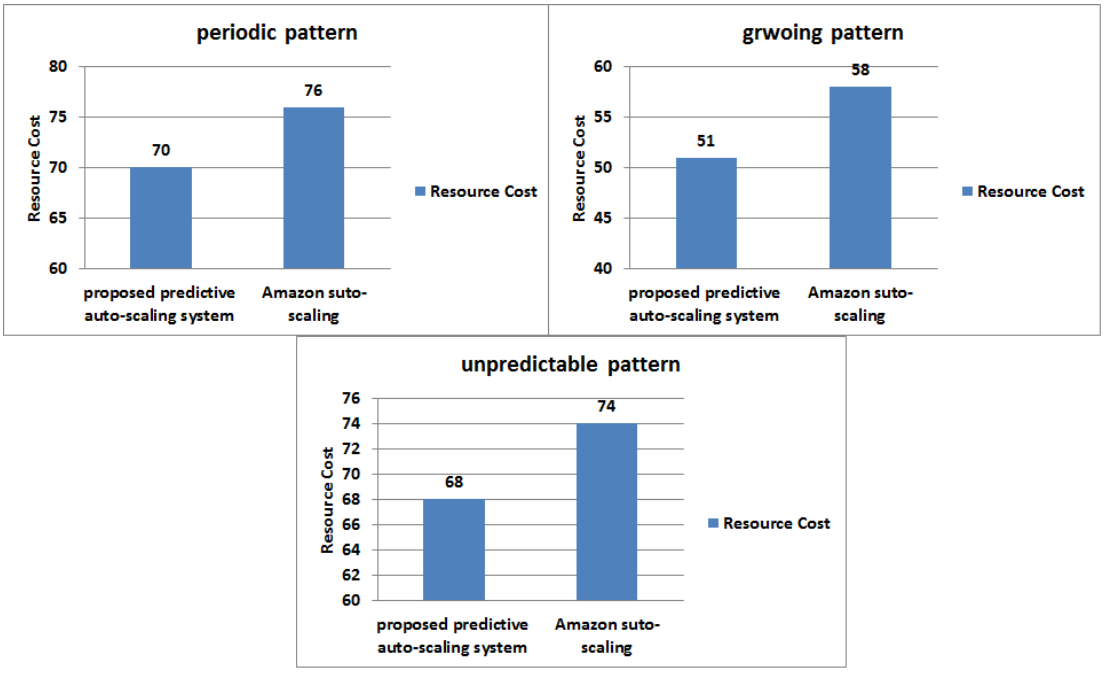


**Figure 4-9. Resource cost comparison**

According to Figure 4-9, the proposed auto-scaling system incurs less resource cost compared to the Amazon auto-scaling system. Although the identical values were used to configure both of the decision makers, since the proposed system uses the predictive approach, it can forecast the future workload and generate more accurate decisions to reduce the resource cost.

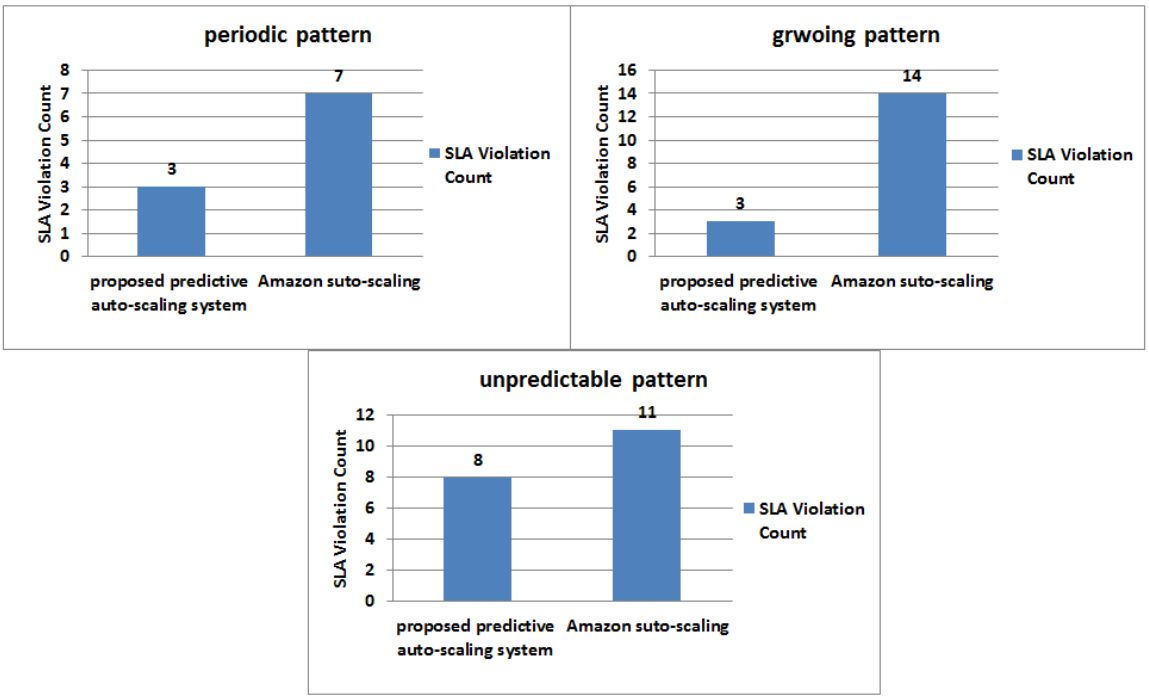


**Figure 4-10. SLA violation count comparison**

The numbers of the SLA violations are compared in Figure 4-10. Section 5.5 argues that using the predictive technique is similar to reducing the VM boot-up time, which reduces the SLA violations. Unlike the Amazon auto-scaling, the proposed system in this thesis uses the prediction methods to generate the scaling actions. This helps the proposed system to request the VMs ahead of time and prevents the under-provisioning condition. Therefore, the proposed system leads to less SLA violations compared with the Amazon auto-scaling system.

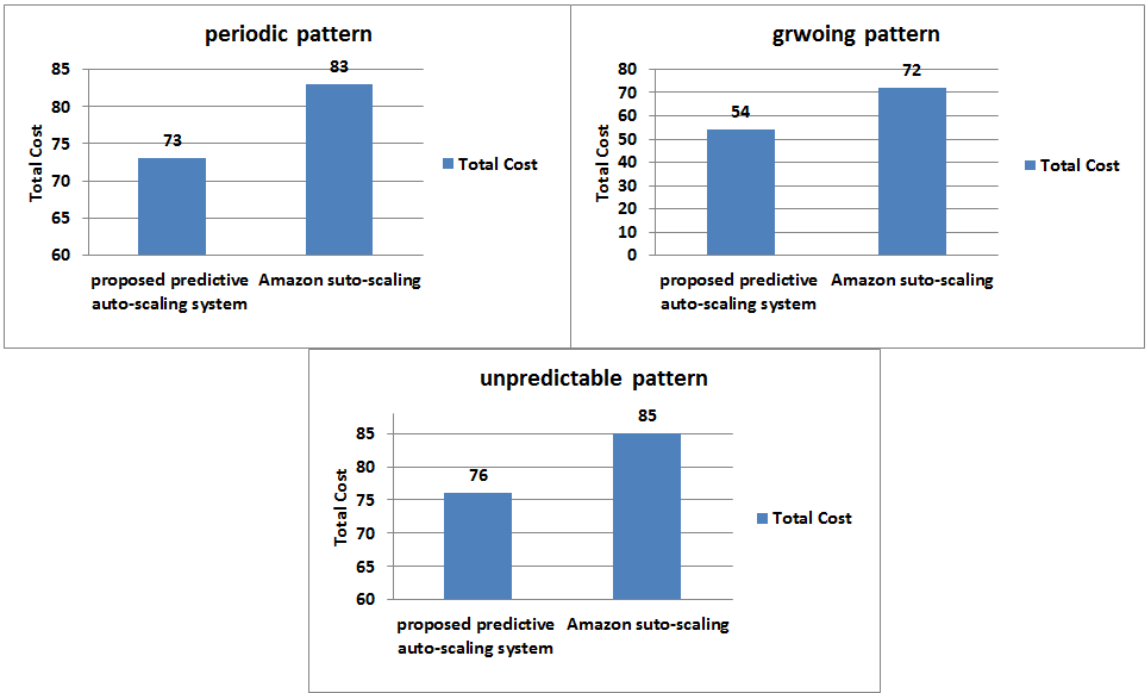


**Figure 4-11. Total cost comparison**

The total costs that are related to the proposed auto-scaling system and the Amazon auto-scaling system are shown in Figure 4-11. According to the results, the proposed auto-scaling system reduces the total cost by:

- 12% in the environments with the periodic workload pattern
- 25% in the environments with the growing workload pattern
- 10.6% in the environments with the unpredictable workload pattern

Since the total cost is the metric to evaluate the accuracy of the auto-scaling systems (see Section 1.4), it can be concluded that the proposed auto-scaling system is more accurate than the Amazon auto-scaling system.

# Chapter 5 - Experiments and Results

This chapter presents the experiments and the results that provide the experimental foundation for the proposed predictive auto-scaling system of this thesis. Each section of this chapter is dedicated to an experiment and presents its goals, setup, evaluation metrics, and results.

## 5.1 Experiment I: Cloud Resource Auto-Scaling Based on Hidden Markov Model (HMM)

This section proposes a predictive auto-scaling system based on Hidden Markov Model (HMM). ). HMM is a rich mathematical structure which can form a theoretical basis for a wide range of applications [86]. Since the difficulty of selecting the correct parameters is one of the shortcomings of the rule-based systems, this experiment's innovation is to replace the rule-based systems with HMM which doesn't require a manual configuration. The goal of this experiment is to measure the accuracy of HMM as a predictive decision maker.

### 5.1.1 Hidden Markov Models (HMM)

Markov models are used to capture and model the stochastic observations. Although Markov models are powerful abstractions of the time-series data, they fail to capture the model when the states are not visible. On the contrary, *Hidden Markov Models* are able to model the stochastic time-series when the actual sequence of the states is not observable. In addition, when partial information on the states is available, *Partially-Hidden Markov Model* is used to capture the time-series. It depends on the problem domain to select one

of the aforementioned models (i.e., Markov model, Hidden Markov Model, and Partially Hidden Markov Model).

The Hidden Markov Model (HMM) is a ubiquitous tool for the time-series data modeling. HMM is used in almost all of the current speech recognition systems, the applications in the computational molecular biology, the data compression field, the artificial intelligence field, and the pattern recognition field [87]. In other words, HMM is a tool for representing the probability of the distributions over a sequence of the observations. An HMM is characterized by five elements [86]:

1. *N:* the number of the states in the model
2. *M*: the number of the distinct observation symbols per state. The observation symbols represent the physical output of the system that is being modeled.
3. *A*: the probability distribution of the state transition.
4. *B*: the probability distribution of the observation symbol in state j
5. $\pi$: the initial state distribution.

A complete specification of a HMM requires specification of the two model parameters (i.e., N and M), specification of the observation symbols, and specification of the three probability measures A, B, and π. In the literature, the following notation is used to indicate ta complete parameter set of the model [87].

$$\lambda = (A, B, \pi) \tag{5-1}$$

### 5.1.2 Experimental Setup

In the auto-scaling problem domain, the states are associated with the cloud service's conditions. Therefore, in order to decide about the model's states, the cloud service's

conditions should be thoroughly identified. In this experiment a set of the performance indicators is used to describe the cloud service's conditions.

$$PM = \{pm_1, pm_2, \dots, pm_n\} \tag{5-2}$$

And each of the performance indicators $pm_i$ has different values:

$$pm_i = \{v_{i,1}, v_{i,2}, \dots, v_{i,m}\} \tag{5-3}$$

Assume there are $n$ performance indicators and each of the performance indicators has averagely $m$ values, then the cloud service has $m \times n$ different conditions. Because the condition space is huge, the Markov Model's performance dramatically decreases. Also, there is no partial information about the available states therefore the Partially Hidden Markov Model is not applicable in the cloud auto-scaling domain. As the result, HMM is used in this experiment to deal with the auto-scaling problem.

According to [86], the followings are the three basic problems that must be solved for the HMM to be useful in the real-world applications:

1. **Problem I:** For a given observation sequence $O=O_1O_2O_3...O_T$ and a model $\lambda=(A, B, \pi)$, how to compute $P(O|\lambda)$. (i.e., how to compute the probability of observation O given model $\lambda$?)
2. **Problem II:** For a given observation sequence $O=O_1O_2O_3...O_T$ and a model $\lambda=(A, B, \pi)$, how to choose a corresponding state sequence $Q=q_1q_2q_3...q_T$ which best explains the observation.
3. **Problem III:** How to adjust the model parameters $\lambda=(A, B, \pi)$ to maximize $P(O|\lambda)$.

In the auto scaling domain, the observation $O$ refers to a sequence of the scaling actions that are generated by an auto-scaling system. In this experiment, the objective is

to design a N-state HMM. The first step to design the model is to adjust its parameters (i.e., *Problem III*). Then, the model should be optimized. To this end, a number of observations (i.e., the historical scaling actions) should be applied to the model to refine its parameters (i.e., *Problem II*). Finally, once the model has been thoroughly studied and optimized, the model can be used to generate the future scaling actions (i.e., *Problem I*).

Weka [88] is used to create and evaluate the HMM model. Since HMM algorithm is not incorporated in the default version of Weka, the HMMWeka extention [89] is used in this experiment. In addition, to generate the historical data, the TPC-W benchmark (see Section 2.2.2.1) is used as the load generator. The experimental setup is shown in Figure 5-1.

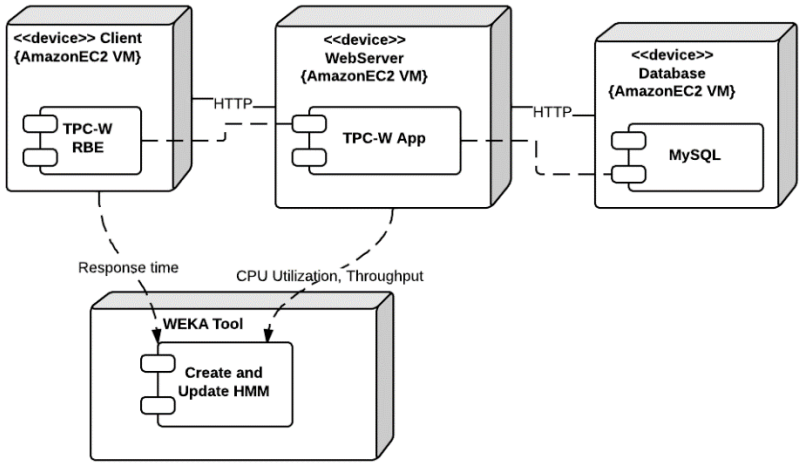


**Figure 5-1. Experimental setup (experiment I)**

The CPU utilization, the throughput, and the response time are monitored as the performance indicators. The response time values are collected from the end users' machines, while the CPU utilization and the throughput values are retreived from the webserver. This experiment assumes that the database tier is not a bottleneck, hence, a

relatively powerful virtual machine is dedicated to the database tier. Amazon EC2 [8] infrastructure is used to deply the TPC-W benchmark. Table 5-1 represents the details of the infrastructure.

**Table 5-1. The hardware specification (experiment I)**

| | **Memory** | **Compute Unit** | **Storage** |
|---|---|---|---|
| **Client** | 1 Megabyte | 4 Core | 8 Gigabyte |
| **Webserver** | 1 Megabyte | 4 Core | 8 Gigabyte |
| **Database** | 2 Megabyte | 8 Core | 20 Gigabyte |

To collect the historical data, the TPC-W benchmark is run for 5 hours (i.e., the experiment duration is 300 minutes). Once data is collected, the dataset is divided into the training and the testing datasets. According to the results of Section 5.2, the accuracy of the machine learning algorithm maximizes as 60% of the experiment duration is used for the training. Therefore, 60% of the experiment duration (i.e., 180 minutes) is used to train the HMM and the rest 40% (i.e., 120 minutes) is used to test the HMM model.

There are different parameters to configure the HMM algorithm in Weka. Table 5-2 describes these parameters and their associated values.

**Table 5-2. The HMM parameters and values**

| **Parameter name** | **Description** | **Value** |
|---|---|---|
| **States** | number of the HMM states to use | 2 |
| **Iteration cutoff** | the proportional minimum change of the likelihood at which to stop the EM iteractions | 0.01 |
| **Covariance type** | whether the covariances of the gaussian outputs should be full matrices or limited to the diagonal or the spherical matrices | Spherical |
| **Tied covariance** | whether the covariances of the gaussian outputs are tied to be the same across all of the outputs | False |
| **Left right** | whether the state transitions are constrained to go only to the next state in a numerical order | False |
| **Random initialization** | whether the state transition probabilities are intialized randomly | False |

Among the parameters in Table 5-2, the default values of the "*iteration cutoff*", "*tied covariance*", "*left right*", and "*random initialization*" are used. Moreover, the "*covariance type*" is set to "*spherical*". The reason is that there is a correlation between the attributes (i.e., the performance indicators) and "*spherical*" covariance type is more suitable when the attributes are related to each other. In addition, the model is configured to have two states, because any value larger than two leads to a large condition set which reduces the model's performance.

### 5.1.3 Evaluation Metrics

The accuracy of the experimental results can be evaluated based on the different metrics such as Mean Absolute Error (MAE), Root Mean Square Error (RMSE), PRED (25) and R2 Prediction Accuracy [90]. Among these metrics, PRED (25) only considers the percentage of the observations whose their prediction accuracy falls within 25% of the actual value. In addition, R2 Prediction Accuracy is a measure of the goodness-of-fit, which its value falls within the range [0, 1] and is commonly applied to the linear regression models [3]. Due to the limitations of PRED (25) and R2 Prediction Accuracy, the MAE and the RMSE metrics are used to evaluate the results. The formal definitions of these metrics are:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|YP_i - Y_i| \tag{5-4}$$

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(YP_i - Y_i)^2}{n}} \tag{5-5}$$

where $YP_i$ is the predicted output and $Y_i$ is the actual output for i-th observation, and $n$ is the number of the observations for which the prediction is made. The MAE metric is a popular metric in the statistics, especially in the prediction accuracy evaluation. The

RMSE metric represents the sample standard deviation of the differences between the predicted values and the actual values. The smaller MAE and RMSE values indicate a more accurate prediction model.

The MAE metric is a linear score which assumes all of the individual errors are weighted equally. Moreover, the RMSE is most useful when the large errors are particularly undesirable. In the auto-scaling domain, a regression model that generates a greater number of small errors is more desirable than a regression model that generates a fewer number of the large errors. The reason is because the rule-based decision makers issue the scale actions based on the prediction values and to generate a correct scale action, the prediction should be close enough to the actual value. In other words, the rule-based decision makers are not sensitive to the small errors in the prediction results. Therefore, the smaller errors in the prediction results are negligible. As a result, in the cloud auto-scaling domain, the RMSE factor is more important than the MAE factor. However, considering both metrics (i.e., MAE and RMSE) provides a comprehensive analysis of the accuracy of the prediction models. The greater is the difference between RMSE and MAE the greater is the variance in the individual errors in the sample.

### 5.1.4 Results

In this experiment, the HMM algorithm is trained to create an auto-scaling model. Then, the created model is evaluated based on the testing dataset. To validate the results, the HMM scaling decisions were compared with the decisions generated by the Amazon auto-scaling system. Figure 5-2 illustrates the HMM and the Amazon scaling decisions.

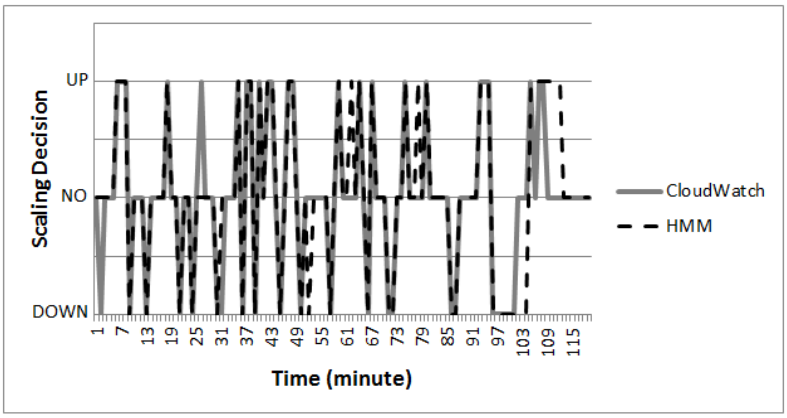


**Figure 5-2. The HMM and the Amazon scaling actions**

There are three possible scaling actions. NO, refers to the conditions in which the auto-scaling system does not take any scaling action. Similarly, DOWN and UP represent the conditions in which the auto-scaling system decides to scale in or out, respectively.

Table 5-3 and Table 5-4 represent the evaluation summaries of the training and the testing phases of the experiment, respectively.

**Table 5-3. The evaluation of HMM (training phase)**

| Evaluation metric | Result |
|---|---|
| Correctly classified instances | 74.32% |
| Incorrectly classified instances | 25.68% |
| MAE | 0.104 |
| RMSE | 0.2624 |

**Table 5-4. The evaluation of HMM (testing phase)**

| Evaluation metric | Result |
|---|---|
| Correctly classified instances | 97.06% |
| Incorrectly classified instances | 2.94% |
| MAE | 0.034 |
| RMSE | 0.1522 |

As shown in Table 5-3 and Table 5-4, the HMM performance increases in the testing phase compared with the training phase. According to the results, once HMM is configured properly it can perfectly handle the scaling conditions in more than 97% of time. In fact, the experimental results show that the HMM auto-scaling system has the similar accuracy as the Amazon auto-scaling system. However, the ultimate goal of this thesis is to improve the accuracy of the existing auto-scaling systems. Therefore, the HMM algorithm is not used in the proposed auto-scaling of this thesis.

## 5.2 Experiment II: The Influence of the Training Duration on the Accuracy of the Prediction Models

The goal of this experiment is to find an optimal training duration for the SVM and the MLP prediction models. It can be hypothesized that increasing the training duration improves the accuracy of the SVM and the MLP prediction models. However, training the SVM and the MLP models not only is an expensive and a time consuming task, but also increases the possibility of the overfitting problem (see Section 3.4). In this experiment the TPC-W benchmark is used to generate a random workload for 300 minutes (i.e., the experiment duration is 300 minutes) and the accuracy of the prediction models (i.e., SVM and MLP) is measured during the following iterations:

- **Iteration 1:** the training duration is 75 minutes, the testing duration is 225 minutes (25% training and 75% testing)
- **Iteration 2:** the training duration is 120 minutes, the testing duration is 180 minutes (40% training and 60% testing)

- **Iteration 3:** the training duration is 150 minutes, the testing duration is 150 minutes (50% training and 50% testing)
- **Iteration 4:** the training duration is 180 minutes, the testing duration is 120 minutes (60% training and 40% testing)
- **Iteration 5:** the training duration is 225 minutes, the testing duration is 75 minutes (75% training and 25% testing)

### 5.2.1 Experimental Setup

To setup the environment, Java implementation of the TPC-W benchmark is deployed in the Amazon EC2 infrastructure. Figure 5-3 illustrates the architectural overview of the experimental setup, which consists of three virtual machines for the client (RBE), web server and database, respectively. All of the virtual machines run on Ubuntu Linux operating system.

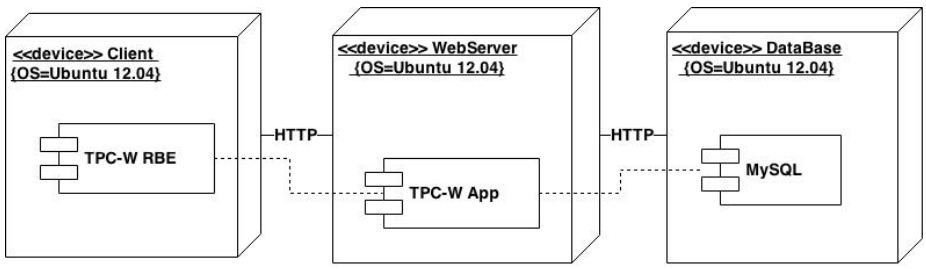


**Figure 5-3. The architectural view of the experiment setup (experiment II)**

Because this thesis investigates the business tier auto-scaling, it is assumed that the database tier is not a bottleneck. Therefore, a relatively powerful virtual machine is dedicated to the database tier. Table 5-5 presents the details of the three virtual machines that are used in this experiment.

**Table 5-5. The hardware specification of the virtual machines (experiment II)**

| | Memory | Processor | Storage |
|---|---|---|---|
| **Client** | 1 Megabyte | 4 Core | 8 Gigabyte |
| **Web Server** | 1 Megabyte | 4 Core | 8 Gigabyte |
| **Database** | 2 Megabyte | 8 Core | 20 Gigabyte |

### 5.2.2 Results

The accuracy of the results is measured by the MAE and the RMSE metrics (see Section 5.1.3). Table 5-6 shows the results.

**Table 5-6. The influence of the training duration on the prediction accuracy**

| | SVM | | ANN | |
|---|---|---|---|---|
| **Iteration #** | **MAE** | **RMSE** | **MAE** | **RMSE** |
| **1** | 2.88 | 4.88 | 3.71 | 5.06 |
| **2** | 2.89 | 4.89 | 3.84 | 5.95 |
| **3** | 2.87 | 4.57 | 3.59 | 5.72 |
| **4** | 2.77 | 4.79 | 3.18 | 4.98 |
| **5** | 2.66 | 4.91 | 3.84 | 5.95 |

According to Table 5-6, there is no significant statistical difference between the MAE and the RMSE values in the different iterations of the experiment. However, the optimal values of these metrics for the MLP algorithm occur in the fourth iteration (i.e., 60% training and 40% testing). Moreover, for the SVM algorithm, the MAE and the RMSE values do not follow a linear pattern and fluctuate by increasing the training duration. For the SVM algorithm, the MAE values decrease by increasing the training duration (except for the second iteration) which means increasing the training duration increases the accuracy of the SVM algorithm. Furthermore, the RMSE values of the SVM algorithm fluctuate until the third iteration (50% training) and then the RMSE values increase afterwards. It means that according to the RMSE metric, for the training

durations that are more than the 50% of the experiment duration, the SVM accuracy decreases. The optimal MAE value for the SVM algorithm is gained at the last iteration (i.e., 75% training), while the best RMSE value of the SVM algorithm occurs in the third iteration (i.e., 50% training). Based on the results, there is no training duration for the SVM algorithm which minimizes the RMSE and the MAE values at the same time. However, the fourth iteration (i.e., the training duration is 60% of the experiment duration) has the second accurate results for both of the metrics. It can be concluded that, considering the MAE and the RMSE results, the optimal training duration for the SVM and the MLP prediction models is 60% of the experiment duration.

## 5.3 Experiment III: The Influence of the Workload Patterns and the Sliding Window Size on the Accuracy of the MLP, MLPWD, and SVM Prediction Models

The main goal of this experiment is to verify the accuracy of the prediction algorithms in the environments with the periodic, the growing, and the unpredictable workload patterns. The SVM algorithm trains the prediction models based on the structural risk minimization, but the ANN algorithm generally uses the empirical risk minimization to create the prediction models. In this experiment, two implementations of the ANN algorithm (i.e., MLP and MLPWD) are investigated. The MLP algorithm uses the empirical risk minimization, and the MLPWD algorithm uses the structural risk minimization. Therefore, comparing MLP with MLPWD isolates the influence of the risk minimization approach on the prediction accuracy of the ANN algorithms. In addition, since the SVM and the MLPWD algorithms use the same risk minimization approach (i.e., structural risk minimization), comparing the SVM algorithm with the MLPWD

algorithm isolates the influence of the regression model on the prediction accuracy (i.e., the neural networks vs. the support vectors).

### 5.3.1 Experimental Setup

The same experimental setup similar to Section 5.2.1 is used in this experiment. On the end user side, a customized script is used along with the TPC-W workload generator to produce the growing, the periodic, and the unpredictable workload patterns. Each of the workload patterns is generated for 500 minutes (i.e., the experiment duration is 500 minutes). Then, on the web-server machine, the total number of the end users' requests is stored in a log file every minute. This results in a workload trace file with 500 data points. The workload trace files are generated for the three workload patterns. These workload trace files are referred to as the *actual workloads*.

The SVM and the ANN algorithms are the most effective prediction algorithms to predict the future cloud services performance [2]. Therefore, in this experiment the SVM and the ANN algorithms are used in the *prediction* component of an auto-scaling system. Moreover, the ANN and the SVM implementations in WEKA tool are used to carry out this experiment. WEKA (Waikato Environment for Knowledge Analysis) is an open-source data mining software in Java that has a collection of several machine learning algorithms for the data mining tasks [88].

According to the results of Section 5.2, the optimal training duration for the ANN algorithm (including the MLP and the MLPWD algorithms), and the SVM algorithm is 60% of the experiment duration. Therefore, the first 300 data points (i.e., 60%) of the

actual workload trace files are considered as the training datasets and the rest 200 data points are considered as the testing datasets.

Another important factor in training and testing of the prediction algorithms is the number of the data features. In this experiment, the actual data has only one feature, which is the number of the requests per minute. Therefore, in order to use the machine learning prediction algorithms, sliding window technique is used. The sliding window technique uses the last *k* samples of a given feature to predict the future value of that feature. For example, given that *b* is the performance indicator, to predict value of $b_{k+1}$, the sliding window technique uses the $[b_1, b_2, \ldots, b_k]$ values. Similarly, to predict value of $b_{k+2}$, the sliding window technique updates the historical window by adding the actual value of $b_{k+1}$ and removing the oldest value from the window (i.e., the sliding window becomes $[b_2, b_3, \ldots, b_{k+1}]$). Deciding the sliding window size is not a trivial task. Usually the smaller window sizes do not reflect the correlation between the historical data, while using the bigger window sizes increases the chance of the over-fitting problem. This experiment studies the effect of the sliding window size on the prediction accuracy of the MLP, MLPWD, and SVM algorithms, as well.

Table 5-7 presents the configuration of the MLP and the MLPWD algorithms in this experiment. Configuration of the SVM algorithm is shown in Table 5-8.

**Table 5-7. MLP and MLPWD configurations**

| Parameter name | MLP | MLPWD |
|---|---|---|
| Learning Rate (ρ) | 0.3 | 0.3 |
| Momentum | 0.2 | 0.2 |
| Validation Threshold | 20 | 20 |
| Hidden Layers | 1 | 1 |
| Hidden Neurons | (attributes + classes) / 2 | (attributes + classes) / 2 |
| Decay | False | True |

**Table 5-8. SVM configuration**

| Parameter name | Value |
|---|---|
| C (complexity parameter) | 1.0 |
| kernel | RBF Kernel |
| regOptimizer | RegSMOImproved |

### 5.3.2 Results

In this experiment, to evaluate the accuracy of the prediction algorithms, the MAE and the RMSE values of the prediction algorithms are measured (see Section 5.1.3). Table 5-9 and Table 5-10 present the accuracy of the prediction models in the training and the testing phases in an environment with the periodic workload pattern. The results are plotted in Figure 5-4.

**Table 5-9. The MAE and the RMSE values in the training phase (periodic pattern)**

| Window size | MAE | | | RMSE | | |
|---|---|---|---|---|---|---|
| | MLP | MLPWD | SVM | MLP | MLPWD | SVM |
| 2 | 6.12 | 4.12 | 4.85 | 8 | 6.13 | 6.65 |
| 3 | 6 | 4.11 | 4.8 | 8.12 | 6.04 | 6.71 |
| 4 | 5.92 | 4.21 | 4.71 | 7.85 | 6.12 | 6.7 |
| 5 | 6.01 | 4 | 4.79 | 7.77 | 6.31 | 6.65 |
| 6 | 5.72 | 3.98 | 4.8 | 7.95 | 6.25 | 6.45 |
| 7 | 5.5 | 4.07 | 4.66 | 7.61 | 6.19 | 6.34 |
| 8 | 5.21 | 4.11 | 4.67 | 7.31 | 6.21 | 5.98 |
| 9 | 5.44 | 4 | 4.71 | 6.55 | 6.11 | 6.04 |
| 10 | 4.55 | 3.99 | 4.78 | 6.34 | 6.01 | 6.11 |

**Table 5-10. MAE and RMSE values in the testing phase (periodic pattern)**

| Window size | MAE | | | RMSE | | |
|---|---|---|---|---|---|---|
| | MLP | MLPWD | SVM | MLP | MLPWD | SVM |
| 2 | 5.25 | 5.04 | 5.19 | 7.25 | 7.13 | 7.19 |
| 3 | 5.31 | 5.05 | 5.12 | 7.54 | 7.35 | 7.46 |
| 4 | 5.34 | 5.12 | 5.35 | 7.34 | 7.12 | 7.37 |
| 5 | 6.98 | 5.24 | 5.56 | 7.69 | 7.11 | 7.28 |
| 6 | 7.04 | 5.26 | 5.61 | 7.98 | 7.06 | 7.3 |
| 7 | 7.11 | 5.19 | 5.58 | 7.98 | 6.99 | 7.41 |
| 8 | 7.58 | 5 | 5.78 | 8.15 | 7.22 | 7.6 |
| 9 | 8.31 | 4.98 | 5.59 | 10.32 | 7.04 | 7.75 |
| 10 | 9.54 | 5.17 | 5.61 | 12.88 | 7.1 | 7.6 |

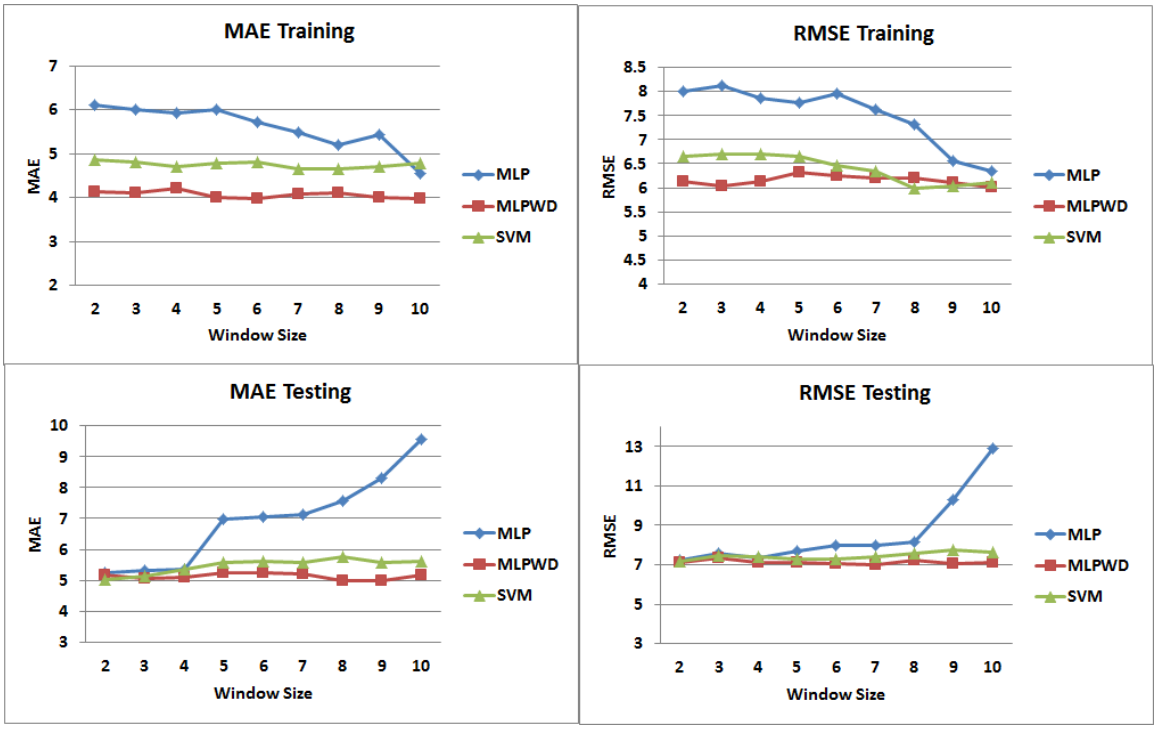


**Figure 5-4. MAE and RMSE values (periodic pattern)**

According to the results, the MLPWD and the SVM prediction accuracies do not change by increasing the window size in the testing and training phases. On the other hand, increasing the window size has a negative effect on the MLP prediction accuracy in the testing phase (cf. Table 5-10), while the MLP accuracy increases slowly by increasing the window size in the training phase (cf. Table 5-9). The negative effect of increasing the window size on the MLP is due to the over-fitting issue. By increasing the window size, the MLP prediction model gets over-fitted to the training dataset. That is the reason why increasing the window size increases the MLP accuracy in the training phase. But, since the prediction model is over-fitted to the training dataset, its prediction accuracy decreases over the testing dataset. Therefore, it can be concluded that for the periodic

workloads, increasing the window size has no substantial effect on the prediction accuracy of the MLPWD and the SVM algorithms. In addition, increasing the window size of the MLP algorithm leads to the over-fitting issue which decreases the MLP accuracy.

Moreover, based on Figure 5-4, for the smaller window sizes, the MLPWD has slightly better prediction accuracy compared to the MLP and the SVM algorithms (for window size = 2, the MLPWD's MAE value is 5.04, which is slightly less than the MLP and the SVM's MAE values that, respectively, are 5.25 and 5.19). By increasing the window size, the MLP accuracy decreases while the MLPWD and the SVM accuracies remain constant. The results show that for the environments with the periodic workload pattern the MLPWD outperforms the MLP and the SVM algorithms. Moreover, increasing the window size does not improve the prediction accuracy of the algorithms in the periodic environments.

The prediction results for the growing workload pattern are presented in Table 5-11, Table 5-12, and Figure 5-5. Similar to the results of the periodic pattern, in the environments with the growing workload pattern, the MLPWD and the SVM prediction accuracies do not change much by increasing the window size. On the other hand, the MLP prediction accuracy has a decreasing trend but does not change too much by increasing the window size. Although the prediction models have a similar reaction to increasing the window size, Figure 5-5 illustrates that the SVM has a better prediction accuracy compared with the MLP and the MLPWD algorithms in the environments with the growing workload pattern.

The last step in this experiment is to analyze the SVM, MLPWD, and MLP prediction accuracies for the unpredictable workload pattern. By definition, an unpredictable workload pattern represents the workloads with consecutive fluctuations.

**Table 5-11. The MAE and The RMSE values in the training phase (growing pattern)**

| Window size | MAE | | | RMSE | | |
|---|---|---|---|---|---|---|
| | MLP | MLPWD | SVM | MLP | MLPWD | SVM |
| 2 | 2.44 | 2.1 | 1.77 | 3.69 | 3.58 | 3.6 |
| 3 | 2.61 | 2.34 | 1.75 | 3.77 | 3.69 | 3.51 |
| 4 | 2.7 | 2.36 | 1.79 | 3.79 | 3.77 | 3.54 |
| 5 | 2.68 | 2.45 | 1.95 | 3.84 | 3.89 | 3.55 |
| 6 | 2.73 | 2.42 | 1.8 | 3.86 | 3.96 | 3.55 |
| 7 | 2.66 | 2.39 | 1.81 | 3.88 | 3.81 | 3.6 |
| 8 | 2.63 | 2.12 | 1.76 | 3.98 | 3.75 | 3.49 |
| 9 | 2.7 | 2.2 | 1.77 | 4.02 | 3.77 | 3.48 |
| 10 | 2.71 | 2.21 | 1.78 | 4.11 | 3.77 | 3.49 |

**Table 5-12. The MAE and The RMSE values in the testing phase (growing pattern)**

| Window size | MAE | | | RMSE | | |
|---|---|---|---|---|---|---|
| | MLP | MLPWD | SVM | MLP | MLPWD | SVM |
| 2 | 3.65 | 2.98 | 2.54 | 4.4 | 4 | 3.78 |
| 3 | 3.77 | 2.87 | 2.5 | 4.87 | 4.75 | 3.68 |
| 4 | 3.54 | 3.11 | 2.48 | 4.92 | 4.5 | 3.65 |
| 5 | 3.64 | 3.29 | 2 | 4.71 | 4.2 | 3.54 |
| 6 | 3.77 | 3.2 | 2.05 | 4.86 | 4.16 | 3.74 |
| 7 | 3.83 | 3.15 | 2.11 | 5.02 | 4 | 3.6 |
| 8 | 4.12 | 3.04 | 2.18 | 5.7 | 3.98 | 3.51 |
| 9 | 4.31 | 3 | 2.21 | 5.24 | 3.9 | 3.5 |
| 10 | 4.33 | 2.95 | 2.16 | 5.45 | 3.87 | 3.7 |

**Table 5-13,**

Table 5-14, and Figure 5-6 illustrate the experimental results for the unpredictable workload pattern. Unlike the growing and the periodic patterns, increasing the window size has a positive effect on the prediction accuracies of the prediction algorithms in the environment with the unpredictable workload pattern. The reason is that in the unpredictable environments there are many fluctuations in the data; therefore, the

prediction models cannot extract the relationships between the features thoroughly. Thus, increasing the window size increases the input size of the algorithms, which improves their prediction accuracies.

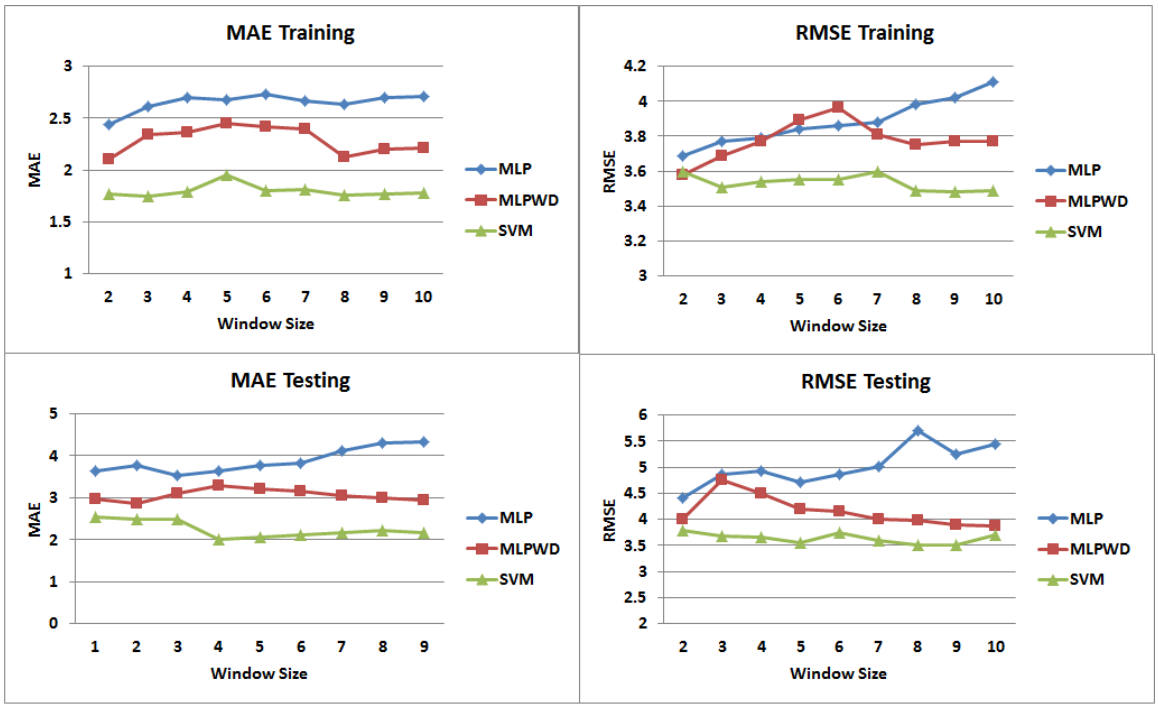


**Figure 5-5. MAE and RMSE values (growing pattern)**

**Table 5-13. The MAE and the RMSE values in the training phase (unpredictable pattern)**

| Window size | MAE | | | RMSE | | |
|---|---|---|---|---|---|---|
| | MLP | MLPWD | SVM | MLP | MLPWD | SVM |
| 2 | 1.58 | 1.64 | 1.72 | 2.55 | 2.59 | 2.98 |
| 3 | 1.56 | 1.63 | 1.7 | 2.64 | 2.51 | 3.11 |
| 4 | 1.56 | 1.62 | 1.68 | 2.54 | 2.64 | 3.18 |
| 5 | 1.54 | 1.63 | 1.65 | 2.34 | 2.66 | 3.05 |
| 6 | 1.56 | 1.65 | 1.69 | 2.41 | 2.6 | 3.19 |
| 7 | 1.57 | 1.66 | 1.72 | 2.38 | 2.59 | 3.3 |
| 8 | 1.55 | 1.65 | 1.74 | 2.2 | 2.59 | 3.2 |
| 9 | 1.65 | 1.66 | 1.64 | 2.1 | 2.61 | 3.09 |
| 10 | 1.54 | 1.64 | 1.79 | 2.15 | 2.54 | 3.06 |

**Table 5-14. The MAE and the RMSE values in the testing phase (unpredictable pattern)**

| Windows size | MAE | | | RMSE | | |
|---|---|---|---|---|---|---|
| | MLP | MLPWD | SVM | MLP | MLPWD | SVM |
| 2 | 2.51 | 2.7 | 2.68 | 3.01 | 3.24 | 3.2 |
| 3 | 2.1 | 2.68 | 2.65 | 2.98 | 3.26 | 3.28 |
| 4 | 1.85 | 2.42 | 2.64 | 3.01 | 3.25 | 3.26 |
| 5 | 1.85 | 2.41 | 2.68 | 3.02 | 3.41 | 3.67 |
| 6 | 1.87 | 2.39 | 2.6 | 2.9 | 3.56 | 3.8 |
| 7 | 1.89 | 2.4 | 2.59 | 2.8 | 3.56 | 3.89 |
| 8 | 1.65 | 2.1 | 2.62 | 2.7 | 3.4 | 3.9 |
| 9 | 0.98 | 2 | 2.64 | 2.11 | 2.9 | 4.12 |
| 10 | 0.98 | 2.05 | 2.69 | 2.1 | 2.8 | 4.21 |

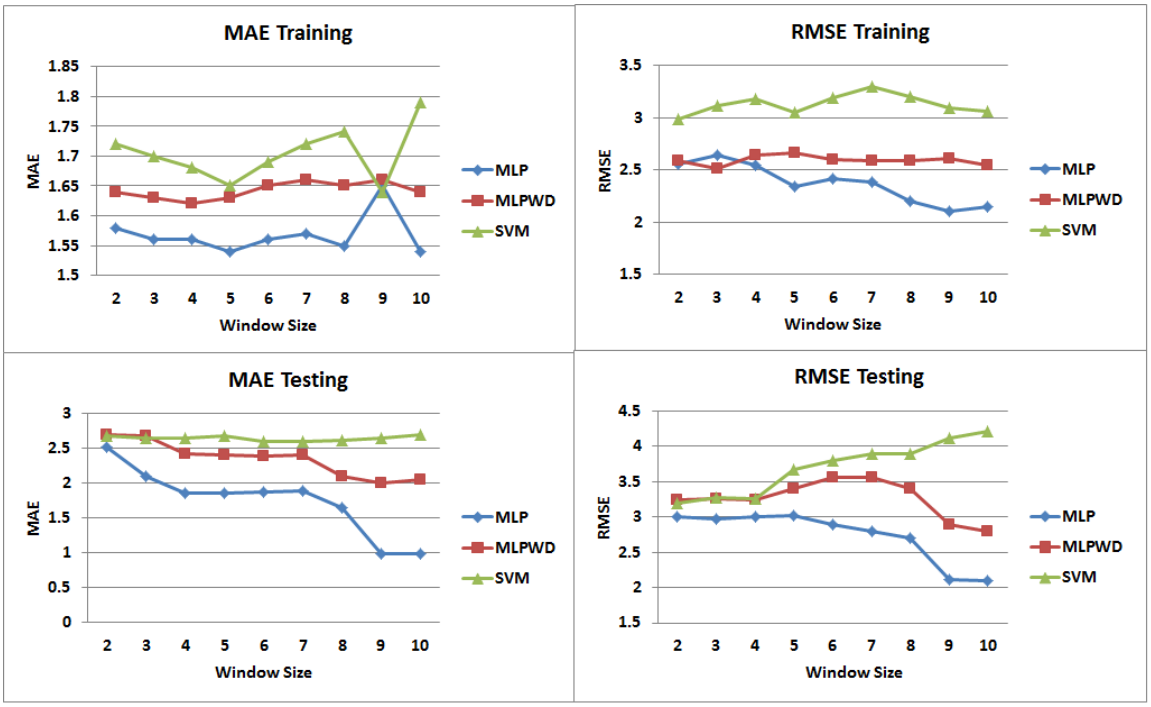


**Figure 5-6. MAE and RMSE values (unpredictable pattern)**

According to Figure 5-6, the MLP algorithm has a better prediction accuracy compared with the SVM and the MLPWD algorithms in the environments with the unpredictable workload pattern. The MLP algorithm tries to cover all of the noise in the

data while the MLPWD and the SVM try to remove the noise and find a smooth curve which covers the valuable data. In the environments with many fluctuations, the MLPWD and the SVM algorithms assume that the spikes are noise in the data. Therefore, the MLPWD and the SVM algorithms and do not capture the spikes in the dataset. The result is that in the environments with the unpredictable workload pattern the MLP algorithm outperforms the MLPWD and the SVM algorithms.

The results of the experiments support the mathematical conclusion that is presented in Section 3.4, which suggests the use of the SRM principle in the environments with the growing and the periodic workload patterns. In addition, the experimental results show that increasing the window size does not improve the SRM accuracy. On the other hand, for the environments with the unpredictable workload pattern, it is better to use the ERM principle with the bigger window sizes. According to the experimental results, Section 4.1 proposes an autonomic prediction suite which chooses the most accurate prediction algorithm based on the incoming workload pattern.

## 5.4 Experiment IV: The Sensitivity of the Auto-Scaling Decisions to the Prediction Results

In the previous section, the impact of the different workload patterns on the prediction accuracy of the SVM, MLP, and MLPWD algorithms was studied. According to the results, the prediction models' accuracies change based on the incoming workload pattern. However, Section 5.3 does not investigate the impact of the different workload patterns on the final scaling decisions. This section investigates the effect of the different workload patterns on the workload prediction results and consequently on the final

scaling decisions. This is an important point to consider, as the prediction results play a crucial role on the final scaling decisions.

Sensitivity analysis is to quantify and analyze the effects of the uncertainty in the inputs of a mathematical model to the uncertainty in its outputs [91]. In this section, the sensitivity of an auto-scaling system is referred to as the effect of the different performance predictions (inputs) to the scaling actions (output). To this end, an experiment is conducted with the workload as the performance indicator, the SVM, the MLP, and the MLPWD algorithms as the prediction algorithms, and the rule-based technique as the decision maker approach to investigate the influence of the prediction results on the scaling decisions in the environments with the different workload patters.

### 5.4.1 Experimental Setup

To conduct the experiment, the similar experimental setup similar as Section 5.3.1 is used. Moreover the three trace files for the periodic, growing, and unpredictable workload patterns that are generated in Section 5.2 are used to represent the different workload patterns. Each of the trace files has 300 data points (i.e., the experiment duration is 300 minutes), and the first 180 data points (i.e., 60%) of the actual workload trace files are considered as the training datasets and the rest 120 data points are used as the testing datasets. Figure 5-7 illustrates the testing dataset of the actual workload trace files for the periodic, growing, and unpredictable patterns.

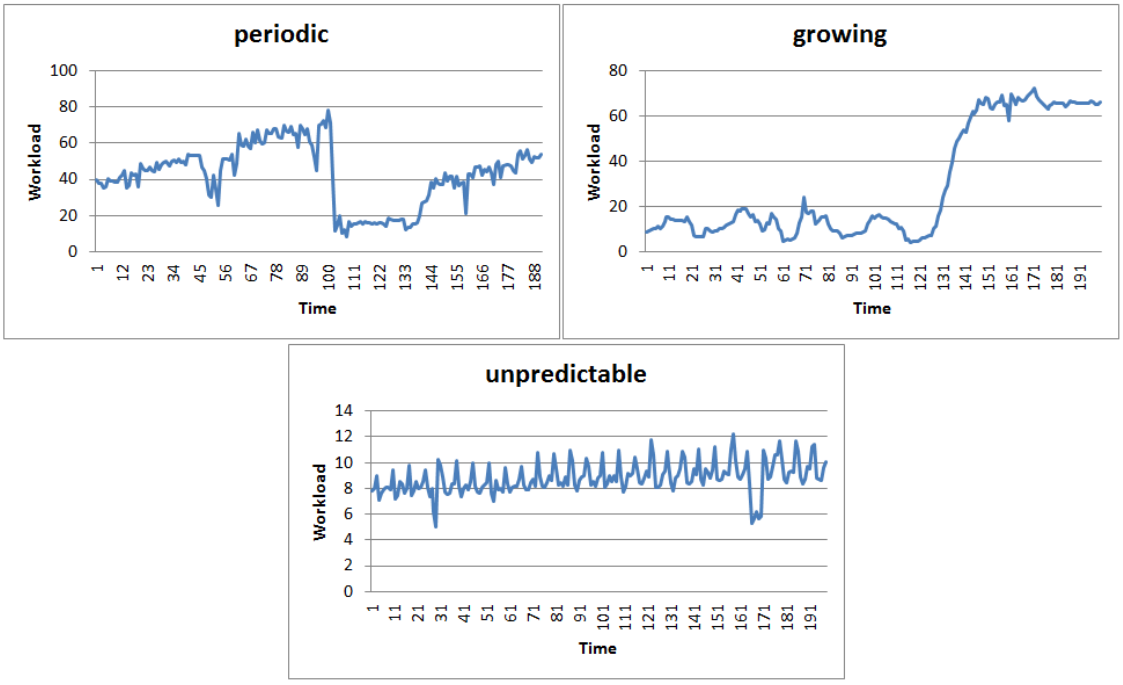


**Figure 5-7. The workload patterns**

In addition, the sliding window technique is used to create the prediction models. The results of Section 5.2 show that increasing the sliding window size only has a positive impact on the accuracy of the prediction models in the environments with the unpredictable workload pattern. However, in the other environments (i.e., the growing or the periodic workload patterns), increasing the window size does not improve the prediction accuracy. Therefore, in this experiment a small window size (i.e., window size = 2 minutes) is used for the growing and the periodic workload patterns and a large window size (i.e., window size = 10 minutes) is used for the unpredictable workload pattern. Furthermore, to prevent the over-fitting problem, the cross-validation technique is used in the training phase.

There are three types of the auto-scaling decisions: scale out, scale in, and no scale. In this experiment, a rule-based technique is used as the decision-maker. One of the

issues that can have a negative effect on the performance of the rule-based systems is the VM thrashing [6], which is measured by the VM thrashing factor. This factor shows the number of the successive opposite scaling actions. In other words, if *Actions* = $\{a_1, a_2, \ldots, a_n\}$ is a set of the scaling actions that are generated during the experiment for a given threshold *t*, and each $a_i$ is a scaling action, then, equation (4-3) calculates the VM thrashing factor for the threshold *t*:

$$VM\ Thrashing_t = Count(a_i) \quad if\ a_i \neq a_{i-1} \tag{5-6}$$

The lower thrashing factor represents a better auto-scaling performance. To decrease the VM thrashing, the time parameters are added to the rule-based technique. These parameters (that are called *durU* and *durL*) show how long a scaling condition must persist to trigger a scaling action [4]. Technically, the job of the time parameters (i.e., *durU* and *durL*) is to postpone the scale out/in actions to decrease the VM thrashing, therefore, the impacts of the time parameters are equal on the performance of the rule-based technique. In this experiment the two time parameters (i.e., *durU* and *durL*) are combined and are referred to as *durUL*.

To investigate the sensitivity of the auto-scaling decisions to the SVM, the MLP, and the MLPWD prediction results, the total number of the *contradictory decisions* is counted. For a given threshold value *t*, each time the SVM, the MLP, and the MLPWD predictions result in an unequal scaling action a *contradictory decision (CD)* occurs. Equation (5-7) measures the percentage of the *identical scaling decisions (ISD)* made by the SVM, the MLP, and the MLPWD predictions for each of the threshold values *t*.

$$ISD_t = \left(1 - \frac{count(CD)}{experiment\ duration}\right) \times\ 100 \tag{5-7}$$

where *count(CD)* represents the total number of the contradictory decision for the threshold *t* during the experiment, and the experiment duration is 300 minutes.

### 5.4.2 Results

According to Figure 5-7, the highest and the lowest workload values for the periodic, growing, and unpredictable patterns, respectively, are [8 to 87], [4 to 72], and [5 to 12]. The threshold values cannot be higher than the maximum and lower than the minimum of the performance indicator's value. For the sake of completeness, in the first iteration of the experiment, the total number of the contradictory decisions for all of the possible threshold values is measured. Table 5-15 represents the percentage of the identical decisions made based on the SVM, MLP, and MLPWD prediction results. Figure 5-8 shows the VM thrashing factor of the threshold values for the three workload patterns.

**Table 5-15. The identical scaling decisions**

| Workload Pattern | Identical Scaling Decisions |
|---|---|
| **Periodic** | 78.2% |
| **Growing** | 78.0% |
| **Unpredictable** | 85.9% |

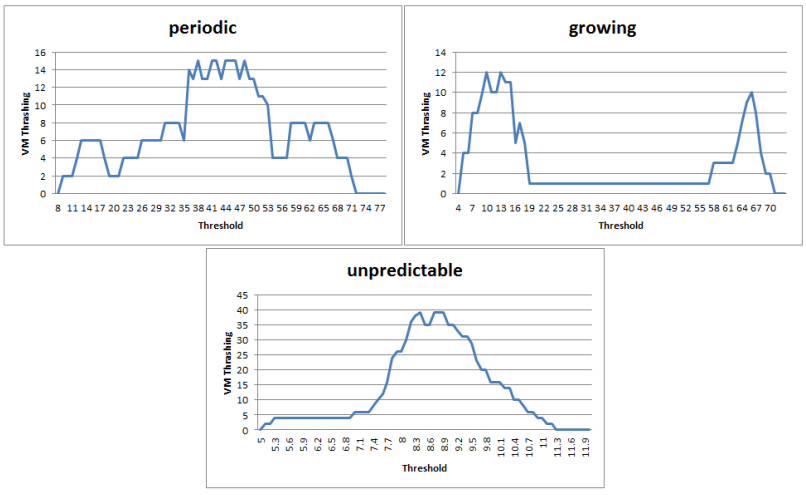


**Figure 5-8. VM thrashing factor**

According to the results, the VM thrashing factor is unacceptably high for some of the threshold values. Therefore, setting the threshold to any of those values is not efficient. According to Figure 5-8, the reasonable threshold values (a reasonable threshold value refers to a threshold value that does not result in a high VM thrashing factor) for each of the workload patterns are:

- Periodic pattern: threshold values in [8 to 26] or [68 to 87]
- Growing pattern: threshold values in [19 to 56]
- Unpredicted pattern: threshold values in [5 to 7.3] or [10.8 to 12]

Table 5-16 represents the percentage of the identical scaling decisions for the aforementioned reasonable threshold values.

**Table 5-16. The identical scaling decisions for the reasonable threshold values**

| Workload Pattern | Identical Scaling Decisions |
|---|---|
| **Periodic** | 99.3% |
| **Growing** | 100% |
| **Unpredictable** | 97.9% |

For the growing workload pattern, the SVM, MLP, and MLPWD predictions lead to the identical scaling decisions literally in all time. This number decrease to 99.3% in the environments with the periodic workload pattern. The least identical scaling decisions percentage occurs for the unpredictable workload pattern with 97.9% of time. It can be concluded that the SVM, MLP, and MLPWD predictions result in the identical scaling decisions in at least 97.9% of time. It should be noted that in this experiment the percentage of the identical decisions are measured. The percentage of the *correct* (or *optimal*) scaling decisions falls out of the scope of this experiment.

## 5.5 Experiment V: The Virtual Machine Boot-Up Time Effect on the Auto-Scaling Cost factors

The VM boot-up time is one of the reasons that causes the under-provisioning condition and is defined as the lag time between requesting a new VM until the VM is fully functional. This experiment analyses the effect of the VM boot-up time on the cost factors of a rule-based decision maker.

### 5.5.1 Experimental Setup

An in-house simulation package is used to carry out this experiment (see Appendix A). The simulation package is accessible at https://github.com/alinikravesh/Carleton-Auto-Scaling-Simulator.git. In the simulation, three instances of the cloud workload patterns are sent to a multi-layer cloud service which is deployed to an IaaS infrastructure. In each of the monitoring intervals, the decision maker compares the incoming workload with the capacity of the cloud service. The capacity of the cloud service refers to the number of the requests that cloud service can accommodate per second. If the incoming workload exceeds the cloud service's capacity (i.e., the upper threshold) then the auto-scaling system scales up the infrastructure. If the cloud service's upper threshold is not reached, the auto-scaling system verifies whether the cloud service is still able to accommodate the incoming workload after releasing one of the provisioned VMs (i.e., the lower threshold). If so, then the auto-scaling system scales down the cloud service. Table 5-17 shows the configuration parameters that are used in the simulation.

According to Table 5-17 the cloud service has two tiers. Most of the cloud services at least include a business tier and a database tier. However, cloud services can have

more tiers. Since the number of the software tiers does not affect the results, in this experiment a 2-tier cloud service is simulated. Moreover, the VM boot-up time for both of the tiers is the same (i.e., 10 minutes).

**Table 5-17. Simulation parameters and values**

| Parameter Name | Value |
|---|---|
| Number of the cloud service tiers | 2 |
| Business tier's service demand | 0.076 seconds |
| Business tier's access rate | 1 |
| Business tier's VM boot-up time | 10 minutes |
| Database tier's service demand | 0.076 seconds |
| Database tier's access rate | 0.7 |
| Database tier's VM boot-up time | 10 minutes |
| The monitoring interval | 5 minutes |

The VM boot-up time is reported to be between 5 and 15 minutes [1][2]. Therefore, in this experiment the initial VM boot-up time is considered to be 10 minutes. Moreover the service demand of the business tier and the database tier are identical. Since the goal of this experiment is to measure the impact of the VM boot-up time on the cost factors, the service demand and the database access rate do not affect the result. The service demand and the database access rate values are decided based on the values that are used in the similar experiments in [81].

Figure 5-9 depicts the three workloads as well as the *ceiling capacity* (see Section 4.2.3) and the *floor capacity* (see Section 4.2.3) of the IaaS infrastructure.

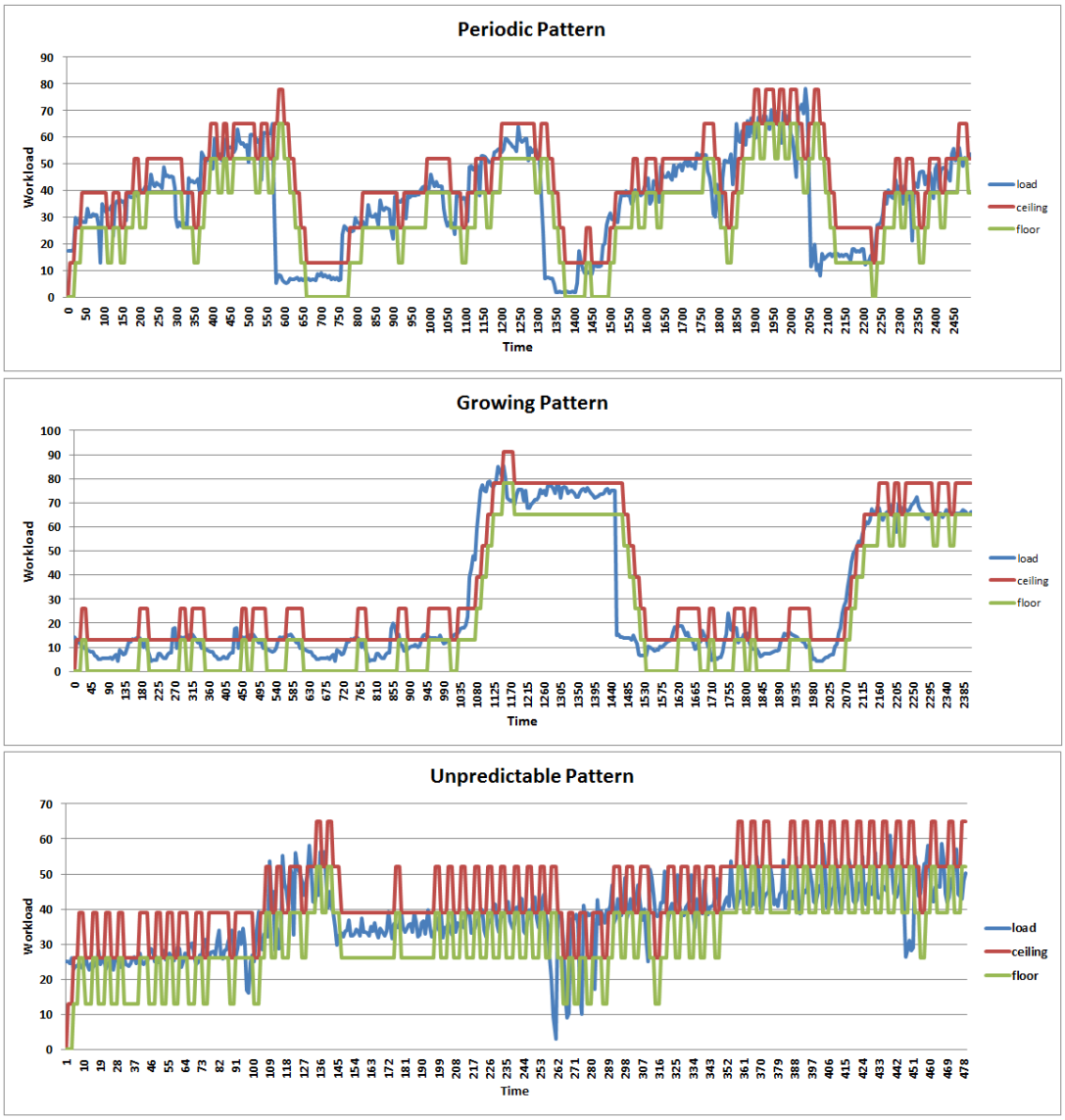


**Figure 5-9. The workload patterns, the ceiling capacity, and the floor capacity**

### 5.5.2 Results

This experiment compares the number of the SLA violations and the resource cost of three IaaS environment: "*IaaS A*" with a VM boot-up time = 10 minutes, "*IaaS B*" with a VM boot-up time = 5 minutes, and "*IaaS C*" with a VM boot-up time = 0 minutes. The experiment has two iterations. In the first iteration the cloud service is simplified and it is assumed that the cloud service consists of just a business tier and there is no database

tier. In the second iteration, a more common scenario is investigated in which, the cloud service consists of a business tier as well as a database tier.

Table 5-18 and Table 5-19 present a comparison of the SLA violation count and the resource cost of the *IaaS A, IaaS B,* and the *IaaS C*, in the first and the second iterations, respectively. In this experiment it is assumed that $c_{vm} = 1\ \$/hr$ (i.e., the leasing rate of a VM is \$1 per hour).

**Table 5-18. The cost factor values for the different VM boot-up times (iteration 1 – no database tier)**

| Workload type | Resource cost | | | SLA violation count | | |
|---|---|---|---|---|---|---|
| | Boot-up time = 10 | Boot-up time = 5 | Boot-up time = 0 | Boot-up time = 10 | Boot-up time = 5 | Boot-up time = 0 |
| Periodic | \$192 | \$172 | \$155 | 83 | 14 | 6 |
| Growing | \$147 | \$138 | \$119 | 47 | 8 | 1 |
| Unpredictable | \$182 | \$173 | \$151 | 90 | 23 | 11 |

**Table 5-19. The cost factor values for the different VM boot-up times (iteration 2 – with database tier)**

| Workload type | Resource cost | | | SLA violation count | | |
|---|---|---|---|---|---|---|
| | Boot-up time = 10 | Boot-up time = 5 | Boot-up time = 0 | Boot-up time = 10 | Boot-up time = 5 | Boot-up time = 0 |
| Periodic | \$333 | \$301 | \$271 | 106 | 28 | 14 |
| Growing | \$243 | \$219 | \$205 | 72 | 18 | 9 |
| Unpredictable | \$318 | \$288 | \$263 | 107 | 39 | 14 |

According to the results, by reducing the VM boot-up the cost factors (i.e., the resource cost and the SLA violation count) decrease. Comparing the cost factors of the *IaaS A* with the *IaaS C* shows that decreasing the VM boot-up time from 10 minutes to 0 minutes causes:

- A significant decrease in the SLA violations

    - Iteration 1: the SLA violations for the periodic, the growing, and the unpredictable workloads are reduced by 92.7%, 97.9%, and 87.8%, respectively.
    - Iteration 2: the SLA violations for the periodic, the growing, and the unpredictable workloads are reduced by 86.8%, 87.5%, and 86.9%, respectively.
- A decrease in the resource cost
    - Iteration 1: the resource cost of the periodic, the growing, and the unpredictable workloads are reduced by 19.3%, 19%, and 17%, respectively.
    - Iteration 2: the resource cost of the periodic, the growing, and the unpredictable workloads are reduced by 18.6%, 15.6%, and 17.3%, respectively.

The VM boot-up time is a characteristic of the IaaS environment. In the real world, the VM boot-up time cannot be reduced to zero. Therefore, researchers have used the predictive auto-scaling approach to imitate reducing the VM boot-up time. For instance, assume the VM boot-up time in a given IaaS environment is 10 minutes. In that environment, a prediction algorithm can be used to forecast the workload of the cloud service 10 minutes ahead of time. This allows the decision maker to generate the scaling decisions based on the prediction results. Therefore, to scale up the infrastructure, a scaling action gets generated 10 minutes ahead of time, and as the result, the new VM will be ready in time. This is practically equivalent to reducing the VM boot-up time to

zero minutes. It should be noted that the prediction cost in the auto-scaling systems is negligible.

## 5.6 Experiment VI: The Smart Kill Effect on the Auto-Scaling Cost factors

This experiment analyses the effect of the smart kill on the cost factors of a rule-based decision maker. The smart kill technique suggests not killing the VMs before they are used for a full hour. The reason is that the IaaS providers round up the partially used hours to the full hours and charge the cloud clients for a full hour VM usage, even if the VM is being used for less than an hour. This experiment compares the number of the SLA violations and the resource cost of two rule-based systems: *System A* which uses the smart kill, and *System B* which doesn't use the smart kill. The configuration parameters of the rule-based decision maker are presented in Table 5-20. Furthermore, in this simulation the target cloud service has two tiers and the VM boot-up time is assumed to be zero minutes. The experimental setup is identical to the setup presented in Section 5.5.1. Table 5-21 shows the results.

**Table 5-20. The parameters of the rule-based decision maker**

| Parameter Name | Value |
|---|---|
| thrU | ceiling capacity |
| thrL | floor capacity |
| durU | 0 minutes |
| durL | 0 minutes |
| inU | 0 minutes |
| inL | 0 minutes |

**Table 5-21. The impact of the smart kill on the cost factors**

| Workload type | Operational cost | | SLA violation count | |
|---|---|---|---|---|
| | Smart kill | No smart kill | Smart kill | No smart kill |
| Periodic | $265 | $271 | 5 | 14 |
| Growing | $204 | $205 | 2 | 9 |
| Unpredictable | $259 | $263 | 5 | 14 |

According to the results, using the smart kill:

- Slightly decrease the resource cost: periodic workload 2.2%, growing workload 0.5%, and unpredictable workload 1.5%
- Reduces the SLA violations: periodic workload 64.3%, growing workload 77.8%, and unpredictable workload 64.3%

The reason why the smart kill decreases the SLA violations is that the smart kill postpones the scale in actions to the end of the VM's billing hour. This increases the capacity of the cloud service and prevents some of the SLA violations that occur due to the unexpected workload surge. In addition, the smart kill reduces the resource cost, because according to the Amazon AWS pricing model [92], if the cloud clients stop and start the exact same instance 5 times in an hour, they get billed for 5 hours. Since the approaches that do not use the smart kill stop and start the VMs more frequently, they use averagely more VMs during each hour period, which increases their resource cost.

## 5.7 Experiment VII: The Impact of the Rule-Based Configuration Parameters on the Auto-Scaling Cost Factors

The previous experiment suggests using the smart kill to reduce the auto-scaling cost factors. However, using the smart kill neither obliterates the number of the SLA violations nor eliminates the excessive resource cost. This section investigates the impact of the configuration parameters of a rule-based system on the cost factors. The rule-based decision makers have six configuration parameters: *thrU, thrL, durU, durL, inU*, and *inL*. Therefore, this simulation has six iterations and each of the iterations is dedicated to

investigate only one of the six configuration parameters. The setup of this experiment is similar to Section 5.5.1.

### 5.7.1 The Upper Threshold (thrU)

The first iteration analyzes the influence of the *thrU* on the SLA violations and the resource cost. The values of the other configuration parameters – except the *thrU* – are held constant during the simulation to isolate the relationship of the *thrU* and the cost factors. In the previous experiments (i.e., Sections 5.5 and 5.6), the *thrU* is equal to the maximum capacity of the VMs. According to Table 5-17, the business and the database tiers have the same service demand which is 0.076 seconds. This value is decided according to the service demand that is used in the similar experiments in [81]. Based on the QNM theory, each of these tiers can handle up to $\frac{1}{0.076} \approx 13$ requests per second. Therefore, to fully utilize each of the VMs capacities, the *thrU* should be set to 13 requests per second. This way, the cloud service can accommodate up to $13 \times n$ requests per second, where *n* is the number of the leased VMs. To test the relationship between the *thrU* and the cost factors, the *thrU* value is deceased and the number of the SLA violations and the resource cost are measured. Table 5-22 shows the results.

According to the results, decreasing the upper threshold increases the resource cost, while it reduces the number of the SLA violations. Since the upper threshold indicates the capacity of the leased VMs, decreasing the upper threshold prevents the VMs from becoming fully utilized by increasing the spare capacity of each of the VMs. Increasing the spare capacity of the VMs necessitates renting more resources and increases the resource cost. However, because the leased VMs are not fully utilized, they are able to

accommodate the unforeseen workload surges; therefore, the number of the SLA violations reduces.

**Table 5-22. The impact of the *thrU* on the cost factors**

| thrU | Periodic | | Growing | | Unpredictable | |
|---|---|---|---|---|---|---|
| | Resource cost | SLA violations | Resource cost | SLA violations | Resource cost | SLA violations |
| 13 | $265 | 5 | $204 | 2 | $259 | 5 |
| 12.5 | $270 | 4 | $210 | 2 | $263 | 5 |
| 12 | $284 | 4 | $214 | 1 | $271 | 3 |
| 11.5 | $287 | 5 | $221 | 1 | $277 | 2 |
| 11 | $295 | 1 | $229 | 0 | $282 | 2 |
| 10.5 | $304 | 0 | $234 | 0 | $294 | 1 |
| 10 | $311 | 0 | $242 | 0 | $310 | 0 |

### 5.7.2 The Lower Threshold (thrL)

The second iteration of the experiment investigates the impact of the *thrL* on the SLA violations and the resource cost. Equation (4-6) calculates the floor capacity of the rule-based decision makers. The floor capacity represents the lower threshold. The decision maker scales down the cloud service whenever the incoming workload becomes less than the floor capacity. The floor capacity calculates the maximum capacity of the cloud service after releasing one of the leased VMs, and indicates the earliest time the decision maker can release one of the VMs without increasing the risk of the under-provisioning. To study the impact of the floor capacity on the cost factors, the floor capacity is gradually increased and the SLA violations and the resource cost are measure. Theoretically, increasing the floor capacity expedites the scale in action and reduces the resource cost. On the other hand, increasing the lower threshold (i.e., the floor capacity) can increase the risk of the SLA violations, because it makes the decision maker to release one VM before the remaining VMs' capacities are enough to handle the incoming workload. Table 5-23 shows the results.

**Table 5-23. Impact of increasing *thrL* on the cost factors**

| thrL | Periodic | | Growing | | Unpredictable | |
|---|---|---|---|---|---|---|
| | Resource cost | SLA violations | Resource cost | SLA violations | Resource cost | SLA violations |
| 13 | $265 | 5 | $204 | 2 | $259 | 5 |
| 13.5 | $264 | 24 | $202 | 28 | $257 | 17 |
| 14 | $260 | 50 | $201 | 31 | $253 | 35 |
| 14.5 | $260 | 61 | $201 | 35 | $252 | 49 |
| 15 | $259 | 71 | $199 | 41 | $250 | 63 |
| 15.5 | $255 | 89 | $199 | 57 | $248 | 70 |
| 16 | $253 | 95 | $198 | 61 | $248 | 77 |

Table 5-23 indicates that, although increasing the lower threshold increases the SLA violations, it doesn't significantly reduce the resource cost. Table 5-24 represents the result of decreasing the *thrL*. Based on equation (4-6), decreasing the lower threshold postpones the scale in action, thus increases the resource cost. On the other hand, decreasing the lower threshold reduces the SLA violations (except for the unpredictable workload) because the decision maker releases a VM only if the remaining VMs have more than enough capacity to handle the incoming workload. However, in the environments with the unpredictable workload the number of the incoming requests may change dramatically between the monitoring intervals.

**Table 5-24. Impact of decreasing *thrL* on the cost factors**

| thrL | Periodic | | Growing | | Unpredictable | |
|---|---|---|---|---|---|---|
| | Resource cost | SLA violations | Resource cost | SLA violations | Resource cost | SLA violations |
| 13 | $265 | 5 | $204 | 2 | $259 | 5 |
| 12.5 | $266 | 4 | $207 | 1 | $262 | 4 |
| 12 | $266 | 4 | $207 | 1 | $261 | 3 |
| 11.5 | $267 | 4 | $209 | 1 | $266 | 4 |
| 11 | $269 | 3 | $213 | 1 | $271 | 3 |
| 10.5 | $270 | 3 | $215 | 1 | $276 | 2 |
| 10 | $270 | 3 | $216 | 1 | $280 | 3 |

Therefore, in some of the timeslots, right after scaling in the IaaS layer the workload increases which causes a SLA violation. However, this does not refute the fact that reducing the *thrL* increasing the marginal capacity of the VMs and reduces the SLA violations (the trend of the SLA violations for the unpredictable workload in Table 5-24 is decreasing).

### 5.7.3 The Duration Parameters (durU and durL)

In the third iteration of the experiment, the influence of the *durL* and the *durU* parameters on the resource cost and the SLA violations are measured. The results for *durL* and *durU* are shown in Table 5-25 and Table 5-26, respectively.

**Table 5-25. Impact of increasing *durU* on the cost factors**

| durU | Periodic | | Growing | | Unpredictable | |
|---|---|---|---|---|---|---|
| | Resource cost | SLA violations | Resource cost | SLA violations | Resource cost | SLA violations |
| 0 | $265 | 5 | $204 | 2 | $259 | 5 |
| 1 | $255 | 62 | $199 | 48 | $241 | 71 |
| 2 | $248 | 106 | $196 | 86 | $233 | 112 |
| 3 | $247 | 134 | $187 | 123 | $229 | 137 |
| 4 | $236 | 177 | $183 | 148 | $220 | 173 |
| 5 | $234 | 197 | $180 | 161 | $220 | 174 |

Similar to reducing the *thrL*, increasing the *durU* postpones the scale out action which reduces the resource cost and has a significant negative influence on the SLA violations. Furthermore, increasing the *durL* postpones the scale in action, thus increases the resource cost and reduces the SLA violations. However, the number of the SLA violations does not significantly decrease by increasing the *durL*. Therefore, one can suggest the use of a smaller *durL* value to reduce the overall auto-scaling cost.

**Table 5-26. The impact of increasing the *durL* on the cost factors**

| durL | Periodic | | Growing | | Unpredictable | |
|---|---|---|---|---|---|---|
| | Resource cost | SLA violations | Resource cost | SLA violations | Resource cost | SLA violations |
| 0 | $265 | 5 | $204 | 2 | $259 | 5 |
| 1 | $271 | 5 | $208 | 2 | $263 | 5 |
| 2 | $280 | 3 | $211 | 2 | $268 | 3 |
| 3 | $284 | 3 | $214 | 1 | $271 | 2 |
| 4 | $291 | 3 | $217 | 1 | $276 | 2 |
| 5 | $294 | 3 | $219 | 1 | $280 | 0 |

### 5.7.4 The Freezing Periods Durations

The last iteration of the experiment studies the impacts of the *inU* and the *inL* parameters on the cost factors (cf. Table 5-27 and Table 5-28). In theory, increasing the *inU* postpones the scale out actions which should increase the SLA violations and reduce the resource cost.

**Table 5-27. Impact of increasing *inU* on the cost factors**

| inU | Periodic | | Growing | | Unpredictable | |
|---|---|---|---|---|---|---|
| | Resource cost | SLA violations | Resource cost | SLA violations | Resource cost | SLA violations |
| 0 | $265 | 5 | $204 | 2 | $259 | 5 |
| 1 | $259 | 5 | $204 | 4 | $259 | 5 |
| 2 | $259 | 5 | $203 | 11 | $259 | 5 |
| 3 | $259 | 7 | $201 | 22 | $259 | 7 |
| 4 | $259 | 7 | $197 | 45 | $259 | 7 |
| 5 | $257 | 15 | $197 | 47 | $257 | 19 |

Similarly, increasing the *inL* postpones the scale in actions which should reduce the SLA violations and increase the resource cost. However, according to the experimental results, the parameters *inU* and *inL* do not significantly impact the cost factors. This is because the auto-scaling model uses the smart kill technique which postpones the scale in action. Therefore, since the scale-in actions are already frozen by the smart kill technique, increasing the *durL* doesn't change the cost factors much.

**Table 5-28. Impact of increasing *inL* on the cost factors**

| inL | Periodic | | Growing | | Unpredictable | |
|---|---|---|---|---|---|---|
| | Resource cost | SLA violations | Resource cost | SLA violations | Resource cost | SLA violations |
| 0 | $265 | 5 | $204 | 2 | $259 | 5 |
| 1 | $265 | 5 | $206 | 2 | $259 | 5 |
| 2 | $265 | 4 | $208 | 2 | $259 | 4 |
| 3 | $265 | 4 | $210 | 2 | $259 | 4 |
| 4 | $265 | 4 | $213 | 2 | $259 | 4 |
| 5 | $270 | 4 | $213 | 2 | $260 | 4 |

### 5.7.5 Section Summary

The results of this section suggest that all of the configuration parameters can influence the SLA violations and the resource cost of the rule-based decision makers. However, some of the parameters, such as the *thrU* and the *durU*, have more influence on the cost factors. In addition, the experimental results show that to decrease the resource cost, cloud clients should:

1. Increase the upper threshold (*thrU*)
2. Increase the lower threshold (*thrL*)
3. Increase the upper scaling duration (*durU*)
4. Decrease the lower scaling duration (*durL*)
5. Increase the upper freezing duration (*inU*)
6. Decrease the lower freezing duration (*inL*)

On the other hand, to reduce the SLA violations, cloud clients should:

1. Decrease the upper threshold (*thrU*)
2. Decrease the lower threshold (*thrL*)
3. Decrease the upper scaling duration (*durU*)
4. Increase the lower scaling duration (*durL*)
5. Decrease the upper freezing duration (*inU*)
6. Increase the lower freezing duration (*inL*)

## 5.8 Experiment VIII: Finding the Optimal Values of the Genetic Algorithm for the Auto-Scaling Problem

This experiment studies the effect of the genetic algorithm parameters on the accuracy of the algorithm that is presented in Figure 4-7 (i.e., the genetic algorithm to configure the rule-based decision makers). The accuracy of the rule-based algorithm is defined by the auto-scaling cost which is related to the set of the configuration parameters that is found by the genetic algorithm.

In this experiment, to calculate the fitness value an individual (each individual represents a valid combination of the rule-based configuration parameters), the genetic algorithm uses the individual's genes values to configure a rule-based decision maker and measures the total auto-scaling cost that rule-based decision maker incurs to the cloud clients. The genetic algorithm has four configuration parameters: *population size, stop condition, crossover rate,* and *mutation rate*. This experiment has four iterations and each iteration is dedicated to the investigate one of the four configuration parameters.

### 5.8.1 Population Size

The population size indicates the number of the solutions that are examined in each of the iterations of the genetic algorithm. The population should be big enough to cover the diversity of the search space. On the other hand, the larger population sizes increase the duration of the experiment. In the first iteration of the experiment, the relationship between the population size and the accuracy of the genetic algorithm is investigated. The fitness value and the duration of the experiment for the different population sizes are shown in Table 5-29. Since the initial population in the first iteration of the genetic

algorithm is randomly generated (see the step 3 of the algorithm in Figure 4-7), the experiment is repeated five times and the results are averaged to increase the precision of the experiment. Figure 5-10 shows the fitness value to the population size, and the duration to the population size relationships.

**Table 5-29. Impact of population size on the genetic algorithm accuracy**

| Configuration | | | | Result | |
|---|---|---|---|---|---|
| **Population size** | **No. of generations** | **Crossover rate** | **Mutation rate** | **Fitness** | **Duration** |
| 10 | 6 | 0.95 | 0.015 | 176 | 657037 |
| 15 | 6 | 0.95 | 0.015 | 170 | 858748 |
| 20 | 6 | 0.95 | 0.015 | 167 | 1182530 |
| 25 | 6 | 0.95 | 0.015 | 166 | 1488723 |
| 30 | 6 | 0.95 | 0.015 | 164 | 1881464 |
| 35 | 6 | 0.95 | 0.015 | 158 | 2103345 |
| 40 | 6 | 0.95 | 0.015 | 158 | 2324168 |
| 45 | 6 | 0.95 | 0.015 | 157 | 2787369 |
| 50 | 6 | 0.95 | 0.015 | 158 | 3022506 |
| 55 | 6 | 0.95 | 0.015 | 156 | 3297244 |
| 60 | 6 | 0.95 | 0.015 | 158 | 3678164 |
| 65 | 6 | 0.95 | 0.015 | 158 | 4159851 |
| 70 | 6 | 0.95 | 0.015 | 157 | 4378326 |
| 75 | 6 | 0.95 | 0.015 | 157 | 4724120 |
| 80 | 6 | 0.95 | 0.015 | 157 | 5057848 |

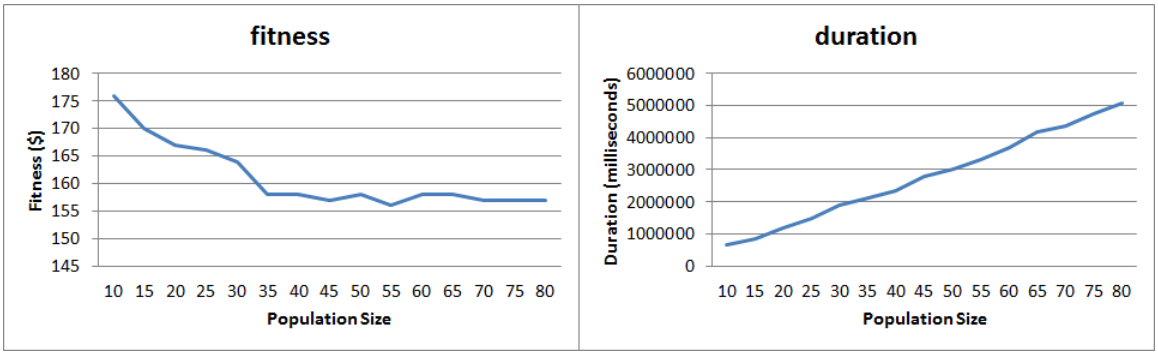


**Figure 5-10. The population size effect on the experiment duration and the fitness value**

According to the results, increasing the population size increases the experiment duration and improves the fitness value. However, increasing the population size to more

than 35 individuals does not have a significant impact on the fitness value. Therefore, the optimal population size for the genetic algorithm in the cloud auto-scaling domain is 35 individuals.

### 5.8.2 Stop Condition

There are three termination conditions that are used in the genetic algorithms: 1) when an upper limit on the number of the generations is reached, 2) when an upper limit on the number of the evaluations of the fitness function is reached, or 3) when the chance of achieving to a significant change in the next generations is excessively low [93]. In this thesis the upper limit on the number of the generations is used as the stop condition. The reason is that this approach is easy to use and is popular in the similar domains [93]. To find the optimal number of the generations, an experiment is carried out to investigate the relationship between the number of the generations and the fitness value of the genetic algorithm. Table 5-30 and Figure 5-11 show the results.

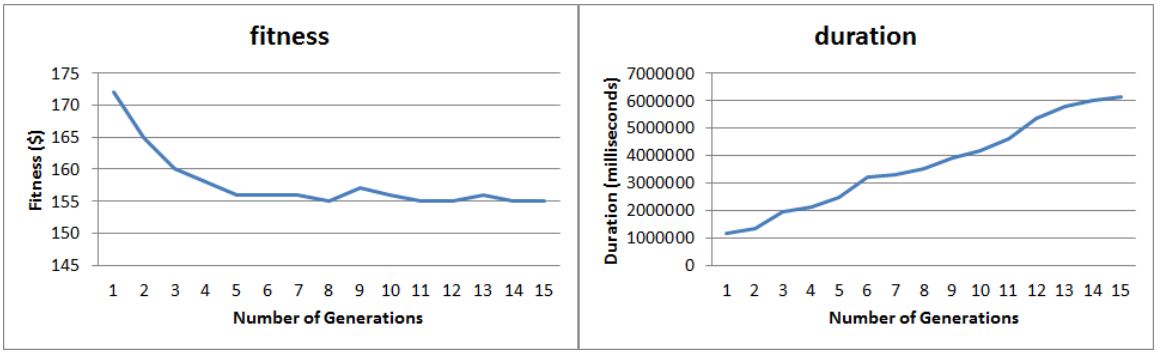


**Figure 5-11. The number of generation's effect on the fitness value and the experiment duration**

**Table 5-30. The impact of the number of the generations on the genetic algorithm accuracy**

| Configuration | | | | Result | |
|---|---|---|---|---|---|
| **Population size** | **No. of generations** | **Crossover rate** | **Mutation rate** | **Fitness** | **Duration** |
| 35 | 3 | 0.95 | 0.015 | 172 | 1161147 |
| 35 | 4 | 0.95 | 0.015 | 165 | 1315245 |
| 35 | 5 | 0.95 | 0.015 | 160 | 1934609 |
| 35 | 6 | 0.95 | 0.015 | 158 | 2109607 |
| 35 | 7 | 0.95 | 0.015 | 156 | 2461951 |
| 35 | 8 | 0.95 | 0.015 | 156 | 3197381 |
| 35 | 9 | 0.95 | 0.015 | 156 | 3305716 |
| 35 | 10 | 0.95 | 0.015 | 155 | 3501466 |
| 35 | 11 | 0.95 | 0.015 | 157 | 3889925 |
| 35 | 12 | 0.95 | 0.015 | 156 | 4180724 |
| 35 | 13 | 0.95 | 0.015 | 155 | 4606840 |
| 35 | 14 | 0.95 | 0.015 | 155 | 5354189 |
| 35 | 15 | 0.95 | 0.015 | 156 | 5780868 |
| 35 | 16 | 0.95 | 0.015 | 155 | 5996488 |
| 35 | 17 | 0.95 | 0.015 | 155 | 6143828 |

Increasing the number of the generations increases the experiment duration and improves the fitness value (i.e., decreases the total cost). However, the fitness value does not significantly improve after 7 generations (cf. Figure 5-11). Therefore, the optimal number of the generations for the genetic algorithm in the cloud auto-scaling domain is 7 generations.

### 5.8.3 Crossover Rate

The crossover rate or the crossover probability indicates a ratio of how many couples will be selected to generate a new offspring. An ideal genetic algorithm accomplishes a proper balance between the exploration and the exploitation of the search space [80]. The exploration means searching the search space as much as possible, while the exploitation means concentrating on the global optimum point. In the genetic algorithm, the crossover

operator is used to lead the population to converge to the good solutions (i.e., the exploitation). Furthermore, the mutation operators are mostly used to provide the exploration.

The higher crossover rates help to concentrate more on the good solutions in the population. However, if the good solutions are not close to the global optimum solution (i.e., the good solutions are close to a local optimum), then the genetic algorithm cannot find the global optimum solution. This experiment explores the relationship between the crossover rate and the fitness value of the genetic algorithm.

**Table 5-31. The impact of the crossover rate on the genetic algorithm accuracy**

| Configuration | | | | Result | |
|---|---|---|---|---|---|
| **Population size** | **No. of generations** | **Crossover rate** | **Mutation rate** | **Fitness** | **Duration** |
| 35 | 7 | 0.50 | 0.015 | 176 | 1763287 |
| 35 | 7 | 0.55 | 0.015 | 173 | 1816652 |
| 35 | 7 | 0.60 | 0.015 | 170 | 2021755 |
| 35 | 7 | 0.65 | 0.015 | 165 | 2164869 |
| 35 | 7 | 0.70 | 0.015 | 161 | 2267017 |
| 35 | 7 | 0.75 | 0.015 | 160 | 2398733 |
| 35 | 7 | 0.80 | 0.015 | 158 | 2567368 |
| 35 | 7 | 0.85 | 0.015 | 157 | 2628260 |
| 35 | 7 | 0.90 | 0.015 | 155 | 2840778 |
| 35 | 7 | 0.95 | 0.015 | 155 | 3070889 |

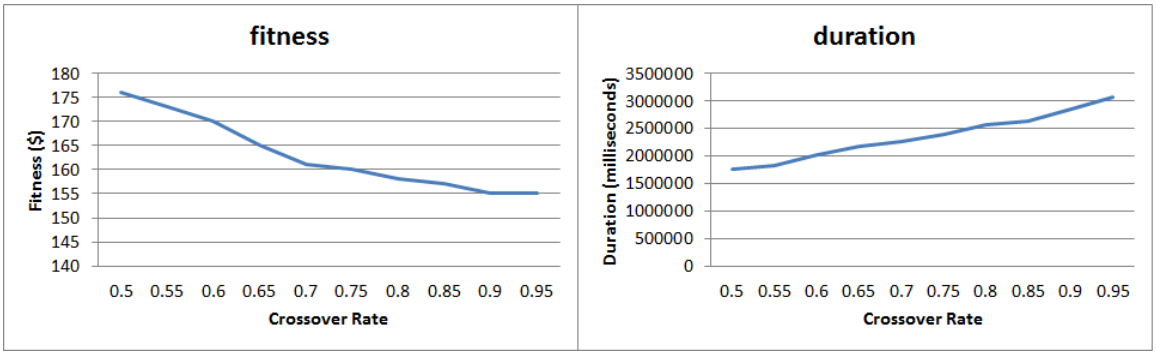


**Figure 5-12. The crossover rate's effect on the fitness value and the experiment duration**

According to the results increasing the crossover rate increases the experiment duration (cf. Figure 5-12). This is because increasing the crossover rate increases the number of the new offspring in each generation, and it takes more time to calculate the fitness of the new offspring, which increases the experiment duration. On the other hand, increasing the crossover rate improves the fitness value (i.e., reduces the auto-scaling cost), because the higher crossover rates increase the exploitation which causes the genetic algorithm to converge to the optimum solution. According to the results, the crossover rate = 0.95 is used in the genetic algorithm to create the cost driven decision maker.

### 5.8.4 Mutation Rate

As mentioned in Section 5.8.3, the mutation operator relates to the exploration function of the genetic algorithms. While the crossover operator tries to converge to a specific point in the search space, the mutation operator avoids the convergence and explores more areas. This helps to prevent from converging around a local optimum point and helps to discover the global optimum solution. However, a high mutation rate increases the probability of searching more areas in the search space and prevents the population to converge to any optimum solution. In other words, a high mutation rate reduces the search ability of the genetic algorithm to a simple random walk while a small mutation rate fails to a local optimum [80]. This experiment investigates the impact of the mutation rate on the accuracy of the genetic algorithm. Table 5-32 shows the impact of the mutation rate on the evolution of the results in the different generations. In the experiment, the population size is 35 individuals, and the crossover rate is 0.95.

**Table 5-32. The impact of the mutation rate on the genetic algorithm accuracy**

| Mutation rate / Generation | 0.02 | 0.03 | 0.04 | 0.05 | 0.06 | 0.07 | 0.08 | 0.09 | 0.1 | 0.11 |
|---|---|---|---|---|---|---|---|---|---|---|
| **1** | 175 | 172 | 173 | 169 | 171 | 173 | 171 | 169 | 158 | 158 |
| **2** | 169 | 165 | 173 | 162 | 171 | 173 | 171 | 169 | 157 | 158 |
| **3** | 169 | 159 | 156 | 162 | 171 | 171 | 171 | 169 | 157 | 158 |
| **4** | 158 | 159 | 156 | 162 | 170 | 171 | 171 | 168 | 157 | 158 |
| **5** | 158 | 159 | 156 | 161 | 168 | 168 | 169 | 168 | 157 | 158 |
| **6** | 158 | 159 | 154 | 161 | 167 | 168 | 169 | 168 | 157 | 158 |
| **7** | 158 | 159 | 154 | 161 | 167 | 168 | 168 | 166 | 157 | 158 |

According to the results, increasing the mutation rate increases the exploration power of the genetic algorithm. However, as shown in Table 5-32, by increasing the mutation rate, the fitness value does not evolve much in the different generations. This is because the higher mutation rates prevent the algorithm from converging to a optimum point. Therefore, mutation rate = 0.02 is used in the genetic algorithm to create the cost driven decision maker.

## 5.9 Experiment IX: Applying the Empirical Risk Minimization (ERM) to Train the Rule-Based Decision Makers

Similar to the functional risk problem of the learning machines in Section 3.2, suppose there is a set of the candidate decision-maker functions $f(x, w)$ and the goal is to find the most accurate function among them. The parameter $x$ is the incoming workload and the parameter $w$ is the set of the configuration parameters. Section 4.2 uses the ERM principle to find the optimal configuration parameters of the rule-based decision maker.

The ERM principle suggests splitting the experimental dataset to the training and the testing datasets, and finding the decision maker with the least auto-scaling cost over the training dataset. According to the ERM principle, the decision maker with the least auto-scaling cost over the training dataset minimizes the auto-scaling cost over the testing dataset, as well. This section provides the experimental evidence that shows the ERM principle is a suitable approach to find the local optimum configuration parameters of the rule-based decision maker. The ERM principle is used in this thesis to train the prediction models (see Section 3.2). This experiment aims to verify whether the ERM principle applies to training the rule-based decision makers.

**Table 5-33. The fitness values for the training and the testing datasets (periodic pattern)**

| Configuration <thrU, thrL, durU, durL, inU, inL> | Training fitness | Testing fitness |
|---|---|---|
| <92, 68, 0, 3, 1, 0> | 161 | 165 |
| <90, 65, 0, 4, 1, 0> | 173 | 181 |
| <82, 59, 0, 4, 0, 1> | 190 | 183 |
| <80, 55, 0, 4, 0, 1> | 192 | 195 |
| <96, 49, 4, 0, 3, 0> | 190 | 189 |
| <97, 56, 4, 0, 3, 0> | 210 | 280 |
| <97, 88, 4, 0, 3, 0> | 275 | 288 |

To carry out this experiment, three workload trace files are generated. Each of the workload trace files represents one of the cloud workload patterns (i.e., periodic, growing, and unpredictable). The workload trace files are similar to Figure 5-9. The generated trace files are split to the training and the testing datasets. In this experiment the training and the testing datasets have equal size (i.e., 50% of the trace file is dedicated to the training and 50% is dedicated to the testing). Then, the different rule-based decision makers are created with the different configuration parameters and their accuracy (i.e., the auto-scaling cost) over the training and the testing dataset is measured.

Table 5-33, Table 5-34, and Table 5-35 show the results for the periodic, the growing, and the unpredictable environments, respectively. To measure the fitness value, it is assumed that $c_{vm} = c_{SLA}$ (i.e., the SLA violations and the resource cost have equal weight).

**Table 5-34. The fitness values for the training and the testing datasets (growing pattern)**

| Configuration <thrU, thrL, durU, durL, inU, inL> | Training fitness | Testing fitness |
|---|---|---|
| <95, 53, 0, 3, 0, 0> | 128 | 122 |
| <80, 29, 0, 0, 0, 4> | 153 | 167 |
| <96, 70, 4, 0, 3, 2> | 176 | 175 |
| <97, 65, 4, 0, 4, 0> | 181 | 179 |
| <99, 69, 4, 0, 4, 0> | 182 | 184 |
| <88, 69, 1, 1, 1, 4> | 167 | 170 |
| <55, 20, 0, 3, 0, 4> | 358 | 551 |

**Table 5-35. The fitness values for the training and the testing datasets (unpredictable pattern)**

| Configuration <thrU, thrL, durU, durL, inU, inL> | Training fitness | Testing fitness |
|---|---|---|
| <96, 66, 0, 2, 2, 4> | 157 | 153 |
| <96, 57, 0, 2, 0, 4> | 154 | 148 |
| <92, 47, 0, 0, 0, 4> | 158 | 166 |
| <97, 73, 4, 0, 4, 1> | 227 | 211 |
| <97, 79, 4, 0, 4, 1> | 240 | 251 |
| <97, 90, 3, 0, 3, 1> | 293 | 311 |
| <99, 90, 3, 0, 3, 1> | 297 | 321 |

As shown in the results, in the three environments, the decision maker which minimizes the fitness value (i.e., the auto-scaling cost) over the training dataset minimizes the fitness value over the testing dataset, as well. In addition, there is a linear relationship between the training fitness and the testing fitness values. Therefore it can be concluded that the ERM principle can be used to train the rule-based decision makers.

## 5.10 Experiment X: The Influence of the Training Duration on the Accuracy of the Decision Maker

Section 5.2 presents an experiment which investigates the impact of the training duration on the accuracy of the prediction models. In addition, Section 5.9 shows that the ERM principle can be used to train the rule-based decision makers. This section investigates the impact of the training duration on the accuracy of the rule-based systems. The accuracy of the rule-based decision makers is defined by their corresponding auto-scaling cost (see Section 1.4). Therefore, in this experiment the auto-scaling cost of a given decision maker is measured in the following iterations. The workload trace files of Section 5.5 are used in this experiment (i.e., the experiment duration is 300 minutes).

- **Iteration 1:** the training duration is 75 minutes, the testing duration is 225 minutes (25% training and 75% testing)
- **Iteration 2:** the training duration is 120 minutes, the testing duration is 180 minutes (40% training and 60% testing)
- **Iteration 3:** the training duration is 150 minutes, the testing duration is 150 minutes (50% training and 50% testing)
- **Iteration 4:** the training duration is 180 minutes, the testing duration is 120 minutes (60% training and 40% testing)
- **Iteration 5:** the training duration is 225 minutes, the testing duration is 75 minutes (75% training and 25% testing)

The parameter sets <92, 68, 0, 3, 1, 0>, <95, 53, 0, 3, 0, 0>, and <96, 57, 0, 2, 0, 4> are used to configure the rule-based decision maker in the periodic, the growing, and the

unpredictable environments, respectively. Table 5-36, Table 5-37, and Table 5-38 show the results.

**Table 5-36. The influence of the training duration on the auto-scaling accuracy (periodic pattern)**

| Iteration # | Training fitness | Testing fitness |
|---|---|---|
| **1** | 158 | 165 |
| **2** | 161 | 167 |
| **3** | 161 | 165 |
| **4** | 167 | 160 |
| **5** | 167 | 162 |

**Table 5-37. The influence of the training duration on the auto-scaling accuracy (growing pattern)**

| Iteration # | Training fitness | Testing fitness |
|---|---|---|
| **1** | 128 | 124 |
| **2** | 130 | 124 |
| **3** | 128 | 122 |
| **4** | 129 | 122 |
| **5** | 127 | 126 |

**Table 5-38. The influence of the training duration on the auto-scaling accuracy (unpredictable pattern)**

| Iteration # | Training fitness | Testing fitness |
|---|---|---|
| **1** | 155 | 157 |
| **2** | 155 | 154 |
| **3** | 154 | 148 |
| **4** | 157 | 145 |
| **5** | 158 | 145 |

According to the results, there is no significant statistical difference between the fitness values in the different iterations. However, the local optimum fitness values in the testing phase occur in the fourth iteration (i.e., 60% training and 40% test). In addition, since the same proportion of the training and the testing datasets is suitable to train the prediction models, to simplify the training of the decision maker, 60% of the experiment duration is used to train and 40% of the experiment duration is used to test the decision

maker. This way, the same training and testing datasets can be used to train the predictor and the decision maker components.

## 5.11 Summary

This chapter presents the experiments and the results that lay out the experimental groundwork for the proposed predictive auto-scaling system of this thesis. Section 5.1 investigates the accuracy of the HMM as the cloud auto-scaling systems. The results show that using the HMM algorithm solves the configuration challenge of the rule-based systems. However, the HMM auto-scaling system has the similar accuracy as the Amazon auto-scaling system, therefore using the HMM algorithm cannot improve the accuracy of the existing auto-scaling systems. For this reason, the HMM algorithm is not further used in the proposed auto-scaling systems of this thesis.

Section 5.2 studies the impact of the training duration on the prediction accuracy of the ANN and the SVM algorithms. The results show that to maximize the prediction accuracy of the ANN and the SVM algorithms, the training and the testing durations should be 60% and 40% of the experiment duration, respectively. The results of this experiment are further used in Sections 5.3 and 5.4.

Section 5.3 investigates the impact of the workload patterns and the sliding window sizes on the prediction accuracy of the MLP, the MLPWD, and the SVM algorithms. Section 5.3 concludes that the MLP outperforms the MLPWD and the SVM for predicting the unpredictable workloads, while the MLPWD is more suitable for the periodic workload patterns and the SVM is more accurate in the environments with a growing workload pattern. In addition, the experimental results suggest using a smaller

window size in the growing and the periodic workload environments and a larger window size for forecasting the unpredictable workloads. The experimental results are used in Section 4.1 to design a self-adaptive prediction suite.

Although Section 5.3 shows the different prediction models are suitable for the different workload patterns, the experimental results of Section 5.4 show that the simple rule-based systems (i.e., the rule-based systems with two configuration parameters) are not sensitive to the different prediction results. Therefore, section 5.4 suggests using a more accurate decision maker in the auto-scaling systems. According to the results, a rule-based decision maker with six configuration parameters is used to generate the scaling decisions.

Sections 5.5, 5.6, and 5.7 investigate the impact of the VM boot-up time, the smart kill technique, and the configuration parameters on the auto-scaling cost factors, respectively. The results of these sections provide the basis for a cost driven decision maker which is presented in Section 4.2. The cost driven decision maker uses the genetic algorithm to find the local optimum values of the configuration parameters. Section 5.8 experimentally finds the local optimum values of the genetic algorithm in the cloud auto-scaling environment.

Section 5.9 investigates the performance of the ERM principle to train a rule-based decision maker. According to the results the ERM principle can be applied to find the local optimum rule-based decision maker. Section 5.10 investigates the optimal duration to train a rule-based decision maker by the ERM principle and suggests using 60% of the experiment duration to train and 40% of the experiment duration to test the rule-based decision maker.

# Chapter 6 - The Impact of the Database Capacity on the Business tier Auto-Scaling

In the previous chapters it is assumed that the database tier is not a bottleneck. Therefore, in all of the previous experiments a powerful server is dedicated to the database tier. However, in the real world, the database servers are the same as the business tier servers and the database tier capacity needs to be managed by an auto-scaling system. The goal of this chapter is to evaluate the impact of the database tier on the business tier auto-scaling. It should be noted that, this chapter does not provide a solution to the database auto-scaling problem, but investigates how the database tier capacity affects the auto-scaling decisions of the business tier. The research question of this chapter is:

*"What is the impact of the database tier capacity on the business tier auto-scaling?"*

To answer this question, the multi-tiered cloud services are analyzed by using the Queuing Network Model (QNM) [81] and the Layered Queuing Network Model (LQNM) [94] theories (see Section 6.1). The analytical investigation provides a theoretical basis for the following hypothesis:

*"The database capacity has no effect on the business tier auto-scaling decisions"*

The QNM and the LQNM theories model a computer system as a network of queues and mathematically evaluate the created model. However, the simulation packages that work based on the QNM and the LQNM theories assume that the capacity of the underlying hardware is static. However, due to the elastic nature of the cloud computing environments, the capacity of the IaaS layer is not static. Therefore, the existing QNM and LQNM simulators cannot be used to evaluate the hypothesis of this chapter. For this

reason, an in-house simulation package (see Appendix A) is developed and used to investigate the hypothesis. Section 6.2 presents an experiment which is carried out by using the simulation package.

## 6.1 Analytical Investigation

This chapter considers a typical cloud service with a 3-tier architecture deployed in an IaaS infrastructure (cf. Figure 6-1).

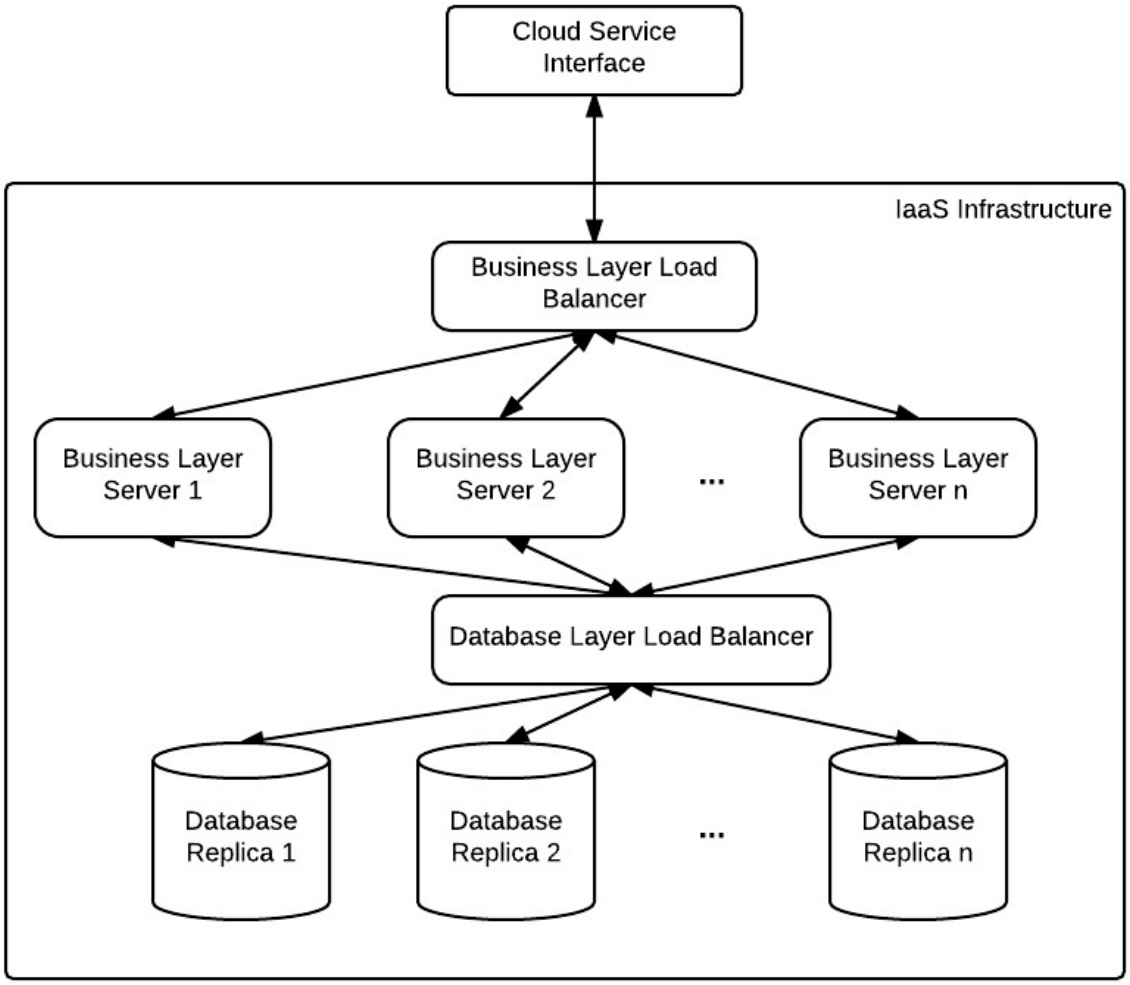


**Figure 6-1. Typical cloud service with 3-tier architecture**

The cloud service owner (i.e., the cloud client) aims to maintain a certain level of the Quality-of-Service (QoS) with the end users. In addition, the cloud service owner aims to lease as minimum amount of resources from the IaaS provider. In this chapter the response time is considered to be the only factor that represents the cloud service's QoS. The response time (i.e., the time lagged between submitting a request until the response of that request is generated [81]) is a variable of the workload and the computational power of the underlying resources [81]. Since the workload of the cloud service changes with time, the cloud client has to constantly acquire and release the IaaS resources to provision a certain response time to the end users.

The main objective of this chapter is to investigate how the database tier's capacity impacts the cloud client's decisions to acquire and release the resources for the business tier of a cloud service. To examine this objective a cloud service is modeled by using the QNM and the LQNM theories.

There are two types of the queuing network models: product form and non-product form. If the following characteristics are hold, then the queueing network has a product-form solution [81]:

1. The routing of customers from one service center to the next must be history independent.
2. The queuing disciplines may be first come first served (FCFS), processor sharing (FS), infinite server (IS), or last come first serve with preemptive-resume (LCFSPR).

3. For an FCFS center, the service time distribution must be exponential; for other servers, the service time distribution does not have to be exponential but must be differentiable
4. A product-form network may have multiple chains (multiple classes) of jobs and may be open with respect to some chains of jobs and closed with respect to others. External arrivals for all open chains must be Poisson.

Since the aforementioned characteristics are valid for the cloud service (c.f. Figure 6-1), the queuing network model of the cloud service has a product-form solution. Therefore, the product-form equations are used in this chapter to model the cloud service.

The first step of modeling a cloud service by using the QNM theory is to find the intensity type of the workload of the cloud service. The workload intensity represents the arrival pattern of the requests to the cloud service. According to the QNM theory there are three workload intensity types [81]:

1. **Transaction workload** has its intensity specified by a parameter $\lambda$ that indicates the rate at which the requests arrive. The transaction workload population varies over time and the requests that have completed service instantaneously leave the cloud service.
2. **Batch workload** intensity is specified by a parameter $N$ that indicates the average number of the active jobs currently in the cloud service. A batch workload has a fixed population and the requests that have completed service leave the cloud service and are replaced instantaneously by the new requests from a backlog of the waiting jobs.

3. **Terminal workload** intensity is defined by two parameters $N$ and $Z$. The parameter $N$ indicates a number of the active jobs currently in the cloud service and the parameter $Z$ indicates the average length of time that jobs use the terminals (i.e., I/O devices) between the interactions (i.e., think time). Similar to the batch workload, the terminal workload population is fixed.

The workload of the cloud service of Figure 6-1 varies with time. Therefore, the cloud service has the *transaction workload* intensity. The QNM theory proposes equation (6-1) [81] to calculate the response time of a cloud service with the transaction workload intensity.

$$R(\lambda) = \sum_{k=1}^{K} \frac{D_k}{1-U_k(\lambda)} \tag{6-1}$$

where $K$ is the number of the service centres, $D_k$ is the service demand of centre $k$, $\lambda$ is the incoming workload intensity (i.e., the number of the requests that arrive at each time unit) and $U_k$ is the utilization of centre $k$.

To model the cloud service of Figure 6-1, it is assumed that each of the software tiers is a service center. Thus, the model of the cloud service has two service centers: a business tier service centre and a database tier service center. The utilization of each of the service centres is calculated by:

$$U_k(\lambda) = X(\lambda) \times D_k \tag{6-2}$$

where $X(\lambda)$ is the throughput of the cloud service. According to the "flow balance assumption" in the QNM, the throughput equals the incoming workload (i.e., $X(\lambda) = \lambda$ ) [81]. Therefore:

$$U_k(\lambda) = \lambda \times D_k \tag{6-3}$$

It should be noted that the maximum value for the service centre utilization is 1. Therefore, the denominator of the fraction in equation (6-1) always has a positive value. Also according to the fraction in equation (6-1), the higher is the service centre utilization, the longer is the response time.

The service demand refers to the time that a request spends in a given service center. This chapter assumes that the service demands (i.e., $D_k$) of the service centers are known. The service demand can be measured by using the instrumentation technique [95]. Substituting equation (6-3) in equation (6-1) results in:

$$R(\lambda) = \sum_{k=1}^{K} \frac{D_k}{1-(\lambda \times D_k)} \tag{6-4}$$

This chapter assumes that the load balancer, as shown in Figure 6-1, equally distributes the incoming workload over the VMs of the cloud service tiers. Therefore, from the VMs' perspective, adding a new VM to a tier is equal to reducing the incoming workload intensity (i.e., $\lambda$) for that tier. On the other hand, removing a VM is similar to increasing the incoming workload intensity. In equation (6-4), the number of the service centres (i.e., $k$) and the service demand of each centre (i.e., $D_k$) are static. Therefore, according to equation (6-4) the only variable that affects the response time is the incoming workload intensity. Since the incoming workloads of the tiers of the cloud service are known, the workload intensity of each VM in a given tier $t$ is calculated as:

$$\lambda_{VM_t} = \frac{\lambda_{St}}{N} \tag{6-5}$$

where $\lambda_{St}$ is the total incoming workload of tier *t*, and $N$ is the number of the VMs in tier *t*.

In the auto-scaling problem domain, when the response time reaches a specified upper threshold (i.e., the scale up condition), the auto-scaling system adds a new VM to

the bottleneck tier to reduce the $\lambda$ and also reduce the response time. Similarly, when the response time reaches a specified lower threshold (i.e., the scale down condition), the auto-scaling system removes a VM to save the resource cost. According to the denominator of the fraction in equation (6-1), when the service center utilization equals 1 (i.e., the service center is fully utilized), the response time value is undefined. In this case, the service center is saturated and is referred to as the bottleneck. The QNM theory defines the maximum amount of the workload that is sustainable by the cloud service, as:

$$\lambda_{max} = \frac{1}{D_{max}} \tag{6-6}$$

where $D_{max}$ is the maximum service demand between the tiers of the cloud service.

According to equation (6-6), the bottleneck of the cloud service is determined by $D_{max}$. Therefore, an auto-scaling system needs to constantly monitor the workload intensity of the business and the database tiers to identify the bottleneck. It should be noted that, because not all of the requests that arrive at the cloud service need to access the database, the workload intensity of the database tier and the business tier are not equal. In our model, the workload intensity of the business tier equals the workload intensity of the cloud service, but the workload intensity of the database tier is calculated as:

$$\lambda_{database} = a \times \lambda \ , \quad 0 \leq a \leq 1 \tag{6-7}$$

where $a$ is the database access rate.

Equation (6-7) shows that the database tier workload is always equal to or less than the business tier workload. In other words, the business tier works like a filter which allows only a percentage of the requests to reach the database tier. In the LQNM theory, the bottleneck of a multi-tier system is defined as a tier that is fully utilized but the

resources it uses (by the nested calls or its processors) are not fully utilized [96]. Based on this definition, when the business tier is fully utilized, but the database tier is not, then the business tier is the bottleneck. Otherwise, when the business and the database tiers are both fully utilized, then the database tier is the bottleneck. The authors of [37] show that in a multi-tier cloud service, adding servers to the bottleneck tier only shifts the bottleneck to a downstream tier. The reason is that when a higher tier (e.g., the business tier) is situated (i.e., $\lambda_{BT} = \frac{1}{D_{BT}}$), but the lower tier (e.g., the database tier) is not ( $\lambda_{DT} < \frac{1}{D_{DT}}$), then the business tier cannot process the incoming requests (i.e., the business tier is fully utilized) and prevents the new requests from accessing the database tier. When the business tier is scaled out, it can handle more workload and transmits more workload to the database tier (i.e., increases $\lambda_{DT}$), which may saturate the database tier.

The research question of this chapter is to investigate how the database tier capacity affects the business tier auto-scaling decisions. According to equation (6-4), the response time of the business tier depends only on its service demand and its workload intensity. In addition, according to equation (6-6), the business tier can sustain at most $\frac{1}{D_{BT}}$ requests at each time unit (where $D_{BT}$ is the service demand of the business tier). The business tier is scaled out only if:

1. The business tier is saturated (i.e., the incoming workload is greater than $\frac{1}{D_{BT}}$)
2. The business tier's response time exceeds a certain threshold.

In both of the cases (i.e., saturation or lengthy response time) the business tier capacity is defined by its demand and its workload intensity, which are independent of the database tier's capacity. Therefore, the business tier auto-scaling decisions depend only on the

incoming workload and the service demand of the business tier. In other words, changing the capacity of the database tier does not change the business tier's scaling decisions. However, according to equation (6-1), adding a database tier to the environment increases the number of the service centres (i.e., value of $K$) which increases the cloud service's total response time and can cause more SLA violations. According to the QNM and the LQNM theories, it can be hypothesized that:

1. The database tier capacity has no effect on the business tier auto-scaling decisions.
2. Adding a database tier to the cloud service increases the number of the SLA violations.

Section 6.2 experimentally investigates these hypotheses.

## 6.2 Experiments and Results

To investigate the hypotheses of this chapter, an experiment is conducted with three cloud services: *Cloud service A* has a business tier with a limited capacity but does not have a database tier, *Cloud service B* has a business tier with a limited capacity and a database tier with an unlimited capacity, and *Cloud service C* has a business tier and a database tier both with the limited capacities.

The first hypothesis of this chapter, suggests that the business tier scaling for all of the three cloud services should include the same set of scaling actions. In addition, the second hypothesis proposes that the total number of the SLA violations in *Cloud service C* is greater than the total number of the SLA violations in *Cloud services A* and *B*. To have a comprehensive comparison between the cloud services, the experiment is

conducted for the periodic, the growing, and the unpredictable workload patterns. Table 6-1 shows the parameters that are used in this experiment. All of the parameter values in this experiment are chosen according to the values that are used in the similar QNM experiments in [81].

**Table 6-1. The experiment parameters and their values**

| | Parameter Name | Value |
|---|---|---|
| **Cloud service A** | Number of the tiers | 1 |
| | Business tier service demand | 0.076 seconds |
| | Business tier access rate | 1 |
| | Business tier VM boot-up time | 10 minutes |
| | Monitoring interval | 5 minutes |
| **Cloud service B** | Number of tiers | 2 |
| | Business tier service demand | 0.076 |
| | Business tier access rate | 1 |
| | Business tier VM boot-up time | 10 minutes |
| | Database tier service demand | 0.0001 seconds |
| | Database tier access rate | 0.7 |
| | Database tier VM boot-up time | 15 minutes |
| | Monitoring interval | 5 minutes |
| **Cloud service C** | Number of tiers | 2 |
| | Business tier service demand | 0.076 |
| | Business tier access rate | 1 |
| | Business tier VM boot-up time | 10 minutes |
| | Database tier service demand | 0.076 seconds |
| | Database tier access rate | 0.7 |
| | Database tier VM boot-up time | 15 minutes |
| | Monitoring interval | 5 minutes |

According to Table 6-1, *Cloud service A* has only one tier which can handle up to $\frac{1}{0.076} \approx 13$ requests per second. *Cloud services B and C* – in addition to the same business tier as *Cloud service A* – have a database tier, as well. The database tier capacity of *Cloud service B* is $\frac{1}{0.0001} \approx 10000$ requests per second, which practically is unlimited. On the other hand, the database tier capacity of *Cloud service C* is the same as its business tier capacity (i.e., $\frac{1}{0.076} \approx 13$ requests per second). The business tier access rate

for all of the three cloud services is 1, which indicates that all of the incoming requests visit the business tier. In *Cloud services B and C*, the database access rate is 0.7 which implies 70% of the incoming requests need the database access.

In this experiment, a reactive rule-based decision maker is used to generate the scaling decisions. The decision maker algorithm calculates the upper and the lower capacity thresholds of a cloud service tier and generates the scale out/in actions for that cloud service tier if the incoming workload of the tier crosses the tier's upper/lower capacity. The upper capacity of a cloud service tier is calculated based on the number of the running VMs for that tier, the incoming workload, the access rate of the tier, and the service demand of the tier. To calculate the lower threshold of a cloud service tier, the decision maker assumes one of the running VMs is released and then calculates the capacity of the infrastructure without that VM. Figure 6-2, Figure 6-3, and Figure 6-4 illustrate the scaling actions for the business tire of the three cloud services in the environments with the periodic, growing, and unpredictable workloads.

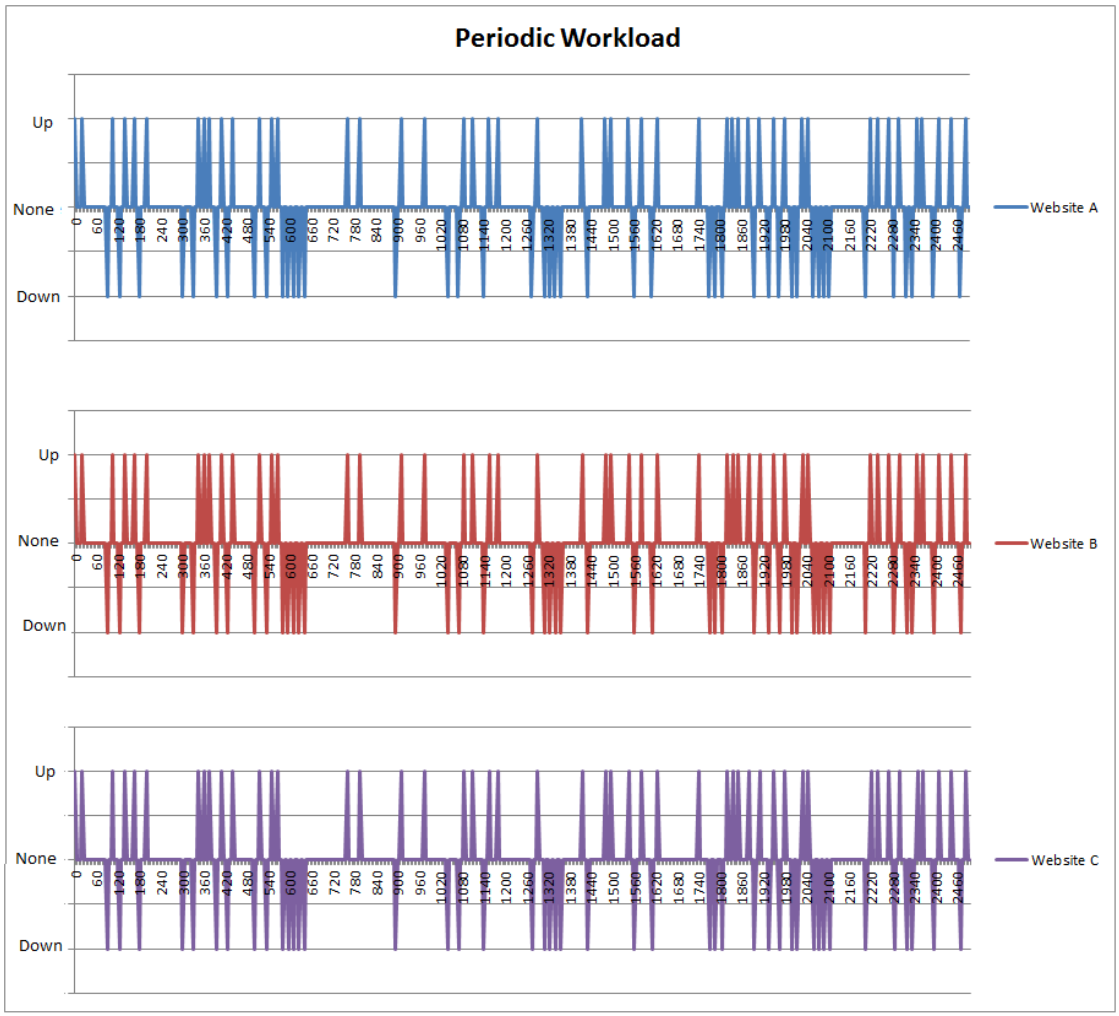


**Figure 6-2. Business tier scaling actions (periodic pattern)**

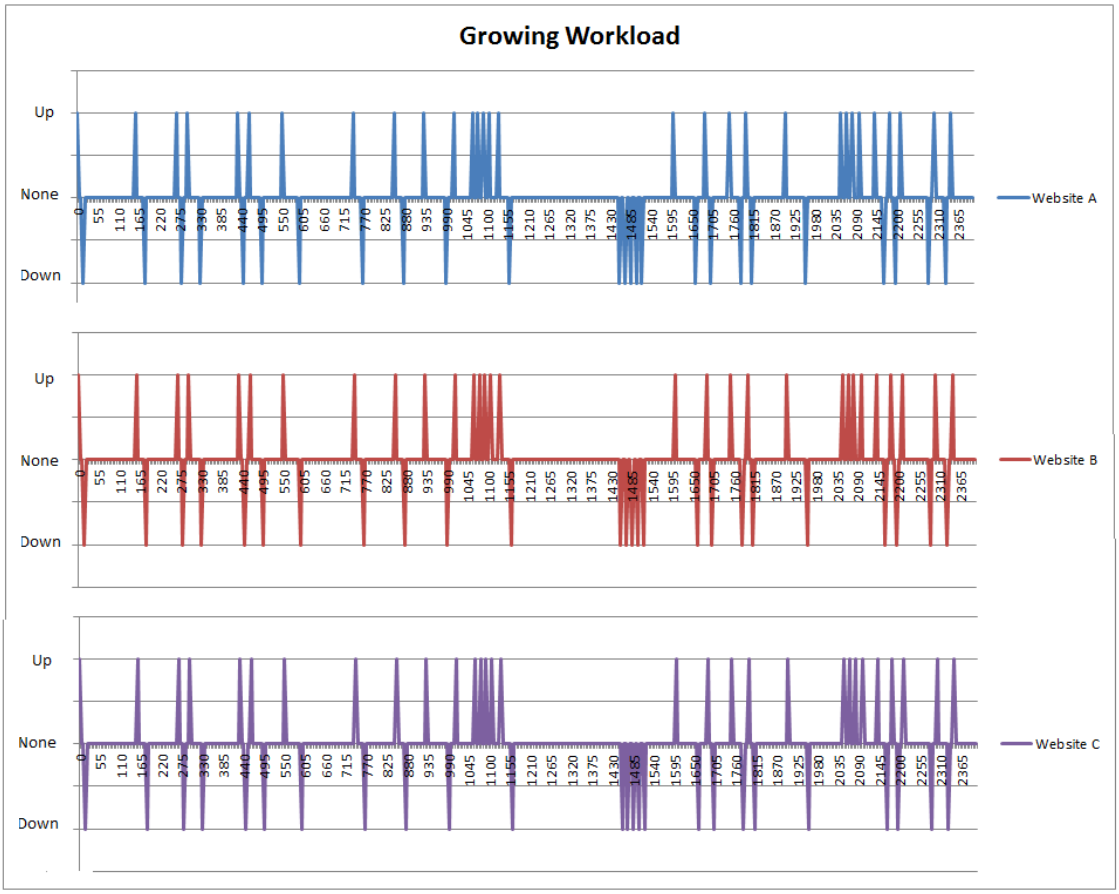


**Figure 6-3. Business tier scaling actions (growing pattern)**

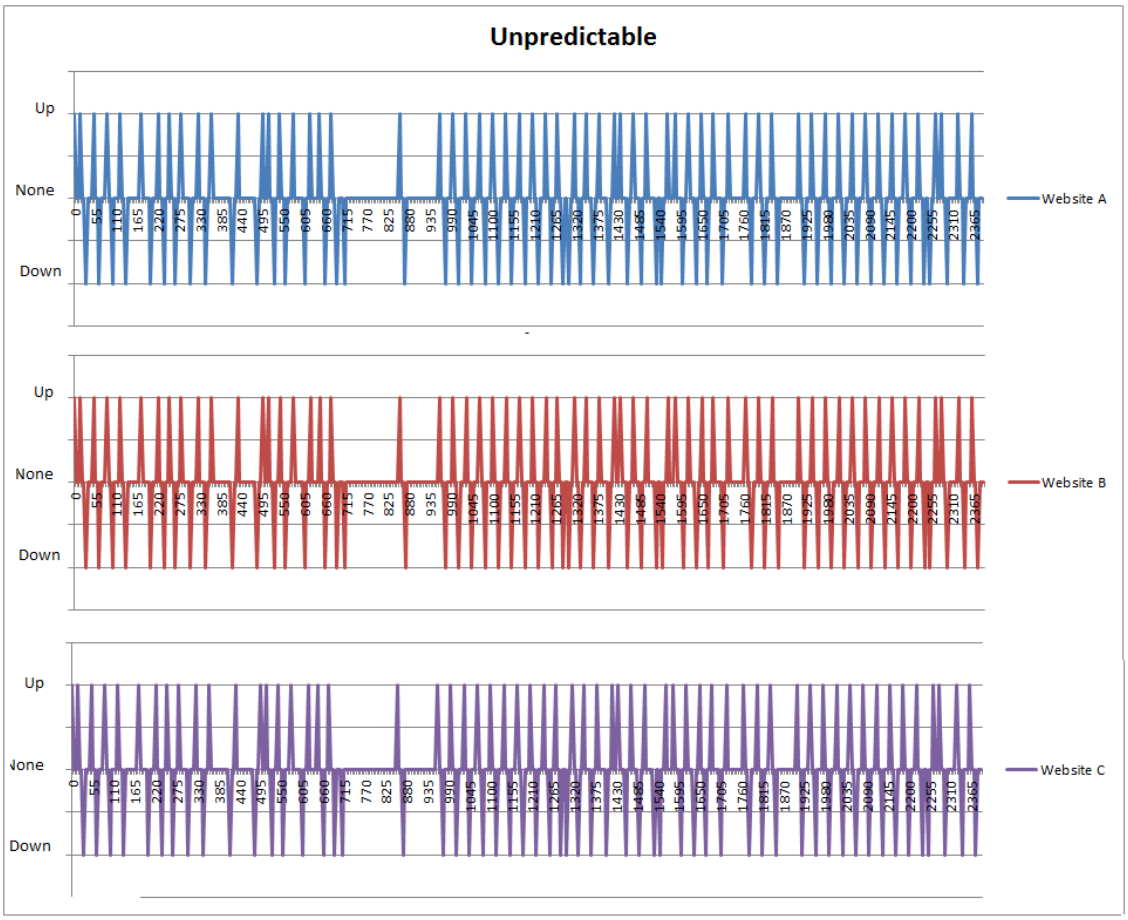


**Figure 6-4. Business tier scaling actions (unpredictable pattern)**

The results show that the scaling decisions for the business tier of *Cloud service A*, *Cloud service B*, and *Cloud service C* are identical. This proves the first hypothesis of this chapter, which suggests that the capacity of the database tier has no effect on the business tier auto-scaling.

Moreover, Table 6-2, Table 6-3, and Table 6-4 illustrate the resource cost and the number of the SLA violations of the cloud services in the environments with the periodic, the growing, and the unpredictable workloads. To measure the resource cost, it is assumed that there is only one VM type with a fixed hourly rate, and the resource cost is

calculated by having the total number of the leased VMs in each hour and the VM's hourly rate. The hourly rate that is used in this experiment is 1 $/hr. Therefore, if the resource cost of a given cloud service is $10 it means that the cloud service has leased 10 hour of VM during the experiment.

**Table 6-2. The resource cost and the SLA violations (periodic pattern)**

| | Resource cost | SLA violation count |
|---|---|---|
| Cloud service A | 154 | 131 |
| Cloud service B | 196 | 131 |
| Cloud service C | 267 | 181 |

**Table 6-3. The resource cost and the SLA violations (growing pattern)**

| | Resource cost | SLA violation count |
|---|---|---|
| Cloud service A | 119 | 74 |
| Cloud service B | 160 | 74 |
| Cloud service C | 206 | 106 |

**Table 6-4. The resource cost and the SLA violations (unpredictable pattern)**

| | Resource cost | SLA violation count |
|---|---|---|
| Cloud service A | 148 | 129 |
| Cloud service B | 188 | 129 |
| Cloud service C | 259 | 165 |

As shown in Table 6-2, Table 6-3, and Table 6-4, *Cloud service A* has less resource cost compared to *Cloud service B* and *Cloud service C*. The reason is that *Cloud service A* does not have a database tier. Therefore, it has less number of the VMs which reduces its resource cost. In addition, the resource cost of *Cloud service B* is less than that of *Cloud service C*. This is because the database tier of *Cloud service B* has an unlimited capacity, which implies that *Cloud service B* only needs one VM for its database during the experiment. On the other hand, since the database tier capacity of *Cloud service C* is

limited, it needs more VMs to handle the database workload which increases its resource cost.

Moreover, since the database tier capacity of *Cloud service B* is unlimited, its database response time is instantaneous which does not affect the total response time of the cloud service. Therefore, *Cloud service A* and *Cloud service B* have equal SLA violation count. However, the total number of the SLA violations in *Cloud service C* is greater than that of *Cloud service A* and *Cloud service B*. The reason is that the database tier of *Cloud service C* has a limited capacity and its database tier becomes the bottleneck during the experiment. Therefore, the auto-scaling system needs to scale out the database tier, which due to the VM boot-up time (see Table 6-1), takes 15 minutes. During this lag time, although the business tier is operating, the cloud service is not operational which increases the SLA violation count. This proves the second hypothesis of this chapter which suggests adding a database tier to the cloud service increases the SLA violations.

# Chapter 7 - Conclusions and Future Work

The elasticity characteristic and the pay-as-you-go pricing model of the cloud computing allow the cloud clients to reduce their cost by only paying for the resources they actually use. The cloud clients provide a service through the cloud environment to the end users and are obligated to maintain a certain level of the QoS. Reducing the cost, on the one hand, and maintaining the QoS, on the other hand, necessitates the cloud clients to deal with a cost/performance trade-off. The auto-scaling systems alleviate the cost/performance trade-off by automatically adjusting the amount of the provisioned infrastructural resources based on the cloud service's incoming workload.

The rule-based systems, due to their simple and easy to understand nature, are currently the most popular auto-scaling systems in the industrial environments [4]. However, the rule-based systems suffer from two main shortcomings: a) the reactive nature, and b) the difficulty of selecting the correct set of the configuration parameters. This thesis proposes a predictive auto-scaling system which overcomes the shortcomings of the rule-based systems. The proposed auto-scaling system consists of a prediction suite and a cost driven decision maker. The prediction suite solves the first shortcoming of the rule-based systems (i.e., the reactive nature) by forecasting the future workload of the cloud service. In addition, the proposed cost driven decision maker alleviates the second shortcoming of the rule-based systems (i.e., the difficulty of the configuration) by using a genetic algorithm to automatically identify the local optimum configuration. The proposed predictive auto-scaling system reduces the total auto-scaling cost that is paid by the cloud clients by 25% compared with the Amazon auto-scaling system.

## 7.1 Contributions

The contributions of this thesis are summarized as:

- **Proposing a self-adaptive prediction suite**

The proposed prediction suite chooses the most accurate prediction method based on the pattern of the incoming workload. To this end, the existing workload patterns in the IaaS layer of the cloud computing environment were investigated. In addition, the mathematical foundations of the time-series prediction models were analyzed, and the suitable prediction model for each of the workload patterns was identified. Moreover, the impact of the sliding window size on the accuracy of the prediction models was mathematically and experimentally investigated.

- **Evaluating the sensitivity of the decision makers with respect to the prediction results**

The sensitivity evaluation of the decision makers by using different prediction results helps to identify the fundamental cause of the poor accuracy of the current auto-scaling systems. To measure the sensitivity of a given decision maker, the scaling decisions generated by that decision makers were compared for different prediction results. In this thesis we measured the sensitivity of a rule-based decision maker by using the MLP, the MLPWD, and the SVM predictions. The experimental results show that the SVM, the MLP, and the MLPWD predictions result in the identical scaling decisions in at least 97.9% of time.

- **Investigating the impact of the database tier capacity on the business tier auto-scaling decisions.**

Many research efforts in the cloud auto-scaling domain (such as [2][3][12]) assume the database tier does not become a bottleneck during the auto-scaling of the business tier. Therefore, in all of the experiments in this thesis, a powerful server is dedicated to the database tier. However, it is important to investigate the influence of the database tier capacity on the business tier auto-scaling decisions. Therefore, this thesis has used the QNM and the LQNM theories to mathematically analyze the impact of the database tier capacity on the business tier scaling decisions. In addition, the mathematical analysis results were experimentally confirmed by using an in-house cloud auto-scaling simulation package.

- **Proposing a cost driven decision maker**

The cost driven decision maker uses a genetic algorithm to find the local optimum configuration of the rule-based decision makers. The local optimum configuration minimizes the total auto-scaling cost based on the cloud clients' preferences. To find the local optimum rule-based decision maker configuration, the local optimum operators and configuration of the genetic algorithm were experimentally identified. Moreover, the impacts of the VM boot-up time and the smart kill strategy on the accuracy of the rule-based decision makers were investigated. The accuracy of the proposed genetic algorithm was evaluated in the environments with different cloud clients' cost preferences.

- **Evaluating the proposed predictive auto-scaling system**

The proposed predictive auto-scaling system was evaluated against the Amazon auto-scaling system. The evaluation results show that the proposed predictive auto-scaling system reduces the total auto-scaling cost that is paid by the cloud clients by:

  - 12% in the environments with the periodic workload pattern

- 25% in the environments with the growing workload pattern,
- 10% in the environments with the unpredictable workload pattern.

## 7.2 Threats to Validity

This thesis is scoped to propose a predictive auto-scaling system to accommodate the compute resources for the business tier of the cloud services in the IaaS layer of the public clouds. This thesis focuses on the horizontal scaling and assumes there is only one type of the VMs in the IaaS layer. All of the experiments and the results that are presented in this thesis should be investigated in the aforementioned scope, and changing the scope in any form (such as considering different types of the VMs, considering other cloud service tiers, considering the network utilization, considering other cloud layers, etc.) may change the results.

In addition, the prediction suite that is proposed in this thesis finds the most accurate prediction model among the MLP, the MLPWD, and the SVM models based on the incoming workload pattern. The accuracy of the prediction models is measured by the MAE and the RMSE metrics. Changing the evaluation metrics may change the relative experimental results.

Furthermore, the proposed cost driven decision maker uses a genetic algorithm to find the optimal configuration of the rule-based decision makers. Changing the decision maker mechanism such as changing its parameter set or its scaling logic may change the genetic algorithm results.

## 7.3 Future Work

The proposed auto-scaling system uses the predictive approach to generate the scaling actions ahead of time to prevent the under-provisioning and the over-provisioning conditions. Another approach to prevent the under-provisioning condition is reducing the VM boot-up time by replacing the VMs with the container-based virtualization (such as Docker [97]) in the IaaS environment. Although reducing the VM boot-up time cannot completely prevent the under-provisioning condition (see Section 2.1), using the containers instead of the VMs can reduce the probability of the SLA violations. Therefore, investigating the impact of the container-based virtualization on the accuracy of the auto-scaling systems can be considered as a future work.

The proposed system measures the SLA violation cost by counting the number of the SLA violations and receiving the cost of each SLA violation from the cloud clients. In addition, the proposed system assumes the entire SLA violations have an equal cost. In future studies the proposed system can be improved by calculating the SLA violation cost more accurately. However, measuring the SLA violation cost depends on different factors, such as the downtime duration of the cloud service, the number of affected end users, and even the sociological aspects of the end users' behaviors. Therefore, to measure the SLA violation cost different classes of the end users as well as the characteristics of each class should be studied.

In addition, the proposed system is limited to the business tier auto-scaling. Extending the proposed auto-scaling system to manage the database tier can be investigated in the future studies. The logic of the database tier and the business tier auto-

scaling are similar. However, unlike the business tier, the database tier includes a set of the datafiles which can greatly affect the auto-scaling strategy. The datafiles should be consistent across the entire database replicas. Therefore, the database auto-scaling, in addition to the mechanisms to generate the scaling actions, includes the strategies to keep the consistency among the datafiles.

Moreover, the proposed system assumes there is only one VM type in the IaaS environment. In the future studies, the proposed genetic algorithm can be improved to configure the decision makers with several VM types. Considering several VM types converts the auto-scaling problem to become a scheduling problem, which can be solved by the genetic algorithms.

# Appendix

### Auto-Scaling Simulation Package

Existing QNM and LQNM simulation packages (such as LQNS [94]) cannot be used to carry out the cloud computing auto-scaling experiments, because all of these simulation packages assume processor capacity (i.e., underlying hardware capacity) is static and cannot change throughout the experiment [98]. But in IaaS environments underlying hardware is elastic and its capacity is adaptable with time. Therefore, we have implemented simulation package in Java which emulates auto-scaling systems and IaaS environment elements. This package has been put in the open-source domain and is publicly accessible [99]. This appendix describes the design of the auto-scaling simulation package and explains how it emulates auto-scaling systems elements.

## A.1 Simulation Package Design

The ultimate goal to implement the auto-scaling simulation package is to provide researchers with a tool that emulates elements of IaaS auto-scaling environment and help researchers to investigate performance of their auto-scaling approaches without deploying their auto-scaling systems in a real life IaaS environment.

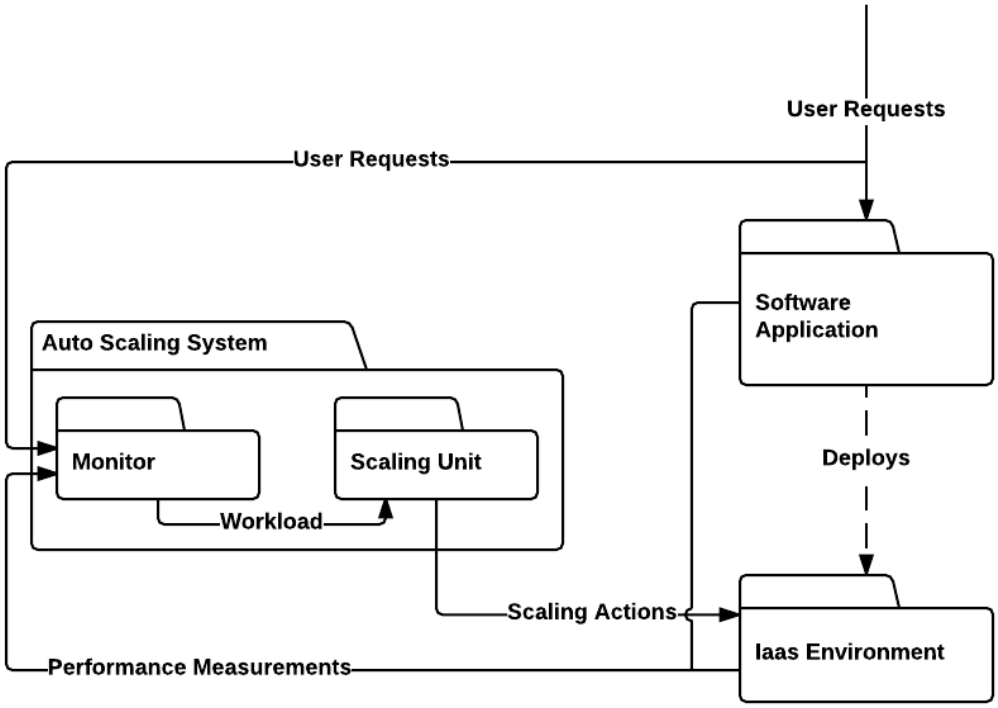


**Figure A-1. Class diagram of the simulation package**

As shown in Figure A-1, the main components of an IaaS auto-scaling domain are:

1. Software application: is the application which is deployed on an IaaS environment and its workload and performance are being monitored for auto-scaling purposes.
2. IaaS environment: represents the underlying infrastructure that its capacity is being adjusted by the auto-scaling system.
3. Auto-scaling system: is the mechanism that manages the IaaS capacity. Auto-scaling system sends the scaling actions to the IaaS environment based on the application workload and the performance measurements.

Our simulation package implements those three components of IaaS auto-scaling domain. Since the main goal of our simulation package is to facilitate auto-scaling

experiments, researchers can replace the auto-scaling system that already exists in the simulation package with their own auto-scaling approach and measure their own approach's performance. Figure A-2 shows the class diagram of the simulation package.

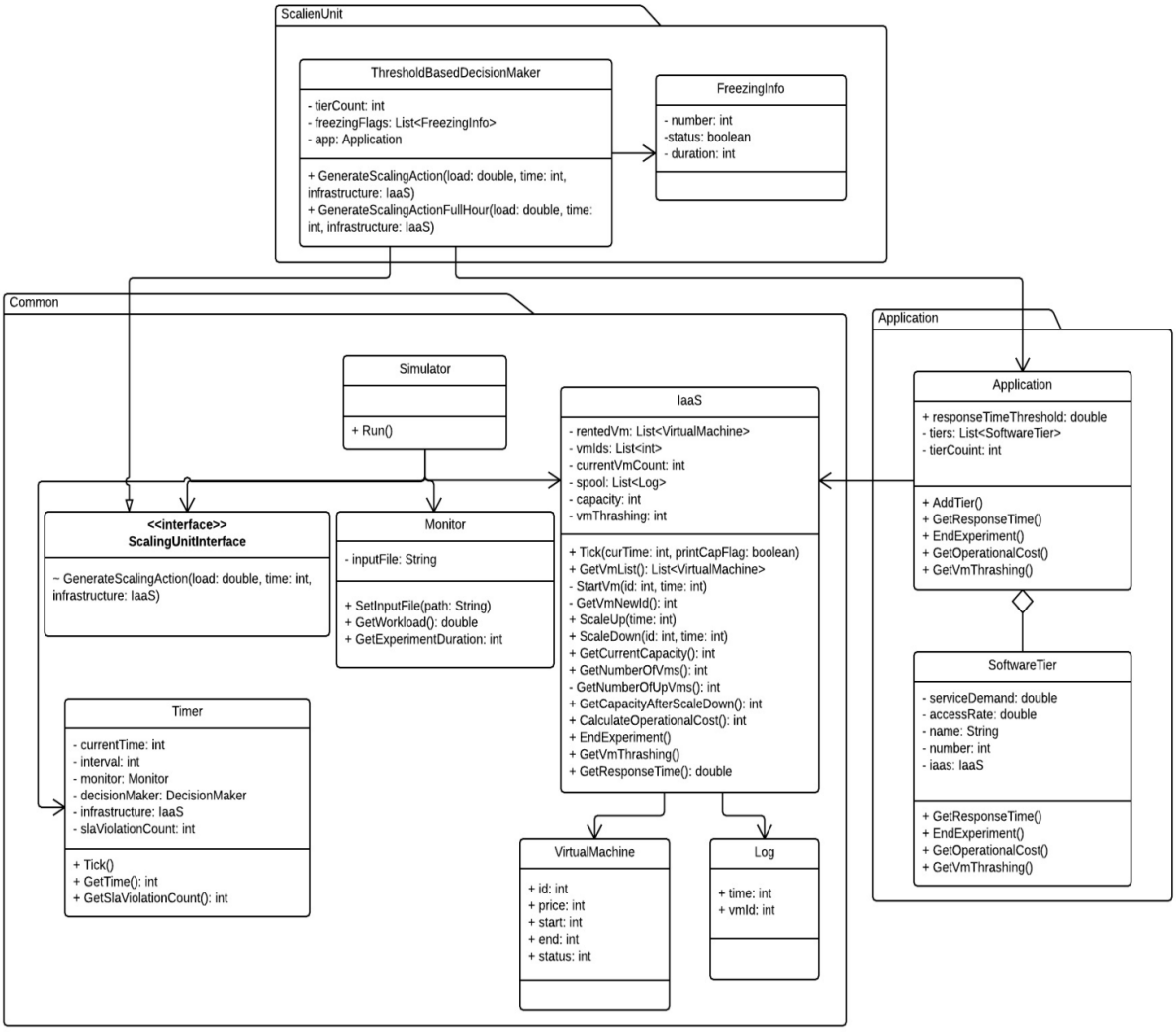


**Figure A-2. Class diagram of the simulation package**

As shown in Figure A-2, the simulation package includes three components: Application, Scaling Unit, and Common. The Application component includes *Application* and *SoftwareTier* classes which imitate software application and its tiers,

respectively. Separating application from its tiers allows researchers to model application with any number of tiers.

The Common component includes classes that are related to the IaaS environment, which are:

1. *Simulator* class: emulates the simulation task. This class only has a method which runs the simulation
2. *Monitor* class: emulates the Monitor component of the auto-scaling systems, as shown in Figure A-2. The *Monitor* class receives a MS Excel file which includes the incoming workload trace and in each time unit returns workload value of the application for that time unit.
3. *VirtualMachine* class: models a virtual machine which has start time, end time, price, and status. Each virtual machine can have three statuses during its lifespan: *Started, Running*, and *Killed*. *Started* VM is a VM which is started but is not fully operational. After spending VM boot-up time for initialization, the VM statues changes from *Started* to *Running*. After a scaling down action, VM status changes from *Running* to *Killed*.
4. *Log* class: represents logs of the generated scaling actions. For each scaling action a new log object gets created. The final list of the log objects represents all of the scaling actions of the system.
5. *IaaS* class: emulates the IaaS infrastructure and implements APIs to scale out or scale in the infrastructure. Since the simulation package assumes that the entire VMs in the IaaS layer have equal boot-up time, the IaaS class has a VM boot-up

time attribute. The IaaS class has mechanisms to enforce VM boot-up time and manage VM state changes.

6. *Timer* Class: is the universal timer of the simulation package and imitates the clock of the system. This class has two main operations: GetTime() and Tick(). GetTime() operation returns the current time of the system. Tick() operation emulates clock ticks, in which the new workload value is retrieved from the Monitor and the Scaling Unit is called to generate scaling action for the new workload value. Moreover, the *Timer* class keeps the total number of SLA violations. In each monitoring interval, the *Timer* class receives the response time from the *Application* class and if the response time exceeds the upper threshold (i.e., SLA breach condition) the Timer class increases the total number of SLA violations. The total number of SLA violations can be used to compare the accuracy of different auto-scaling systems.
7. *ScalingUnitInterface* interface: is an interface which enforces contract of the Scaling Unit package. Since the simulation package allows researchers to implement and use their own scaling units, this interface enforces scaling units to have a unique contract for generating scaling actions.

The Scaling Unit package includes classes that implement scaling unit element of the auto-scaling domain. The simulation package measures the accuracy of the scaling unit by calculating its related cost. The simulation package considers operational cost and SLA violation cost to measure the accuracy of the Scaling Unit. Interested researchers can implement their own scaling unit and measure its accuracy by using the simulation package. The simulation package includes a default scaling unit which implements a

simple reactive threshold based decision maker. The *ThresholdBasedDecisioMaker* class implements the decision maker which scales up/down the infrastructure if the workload exceeds application tier capacity. After each scaling up/down action the *ThresholdBasedDecisioMaker* freezes until the new VM is up and running. The *FreezingInfo* class implements a freezing mechanism for the *ThresholdBasedDecisioMaker* class. Figure A-3 represents the sequence diagram of the simulation package.

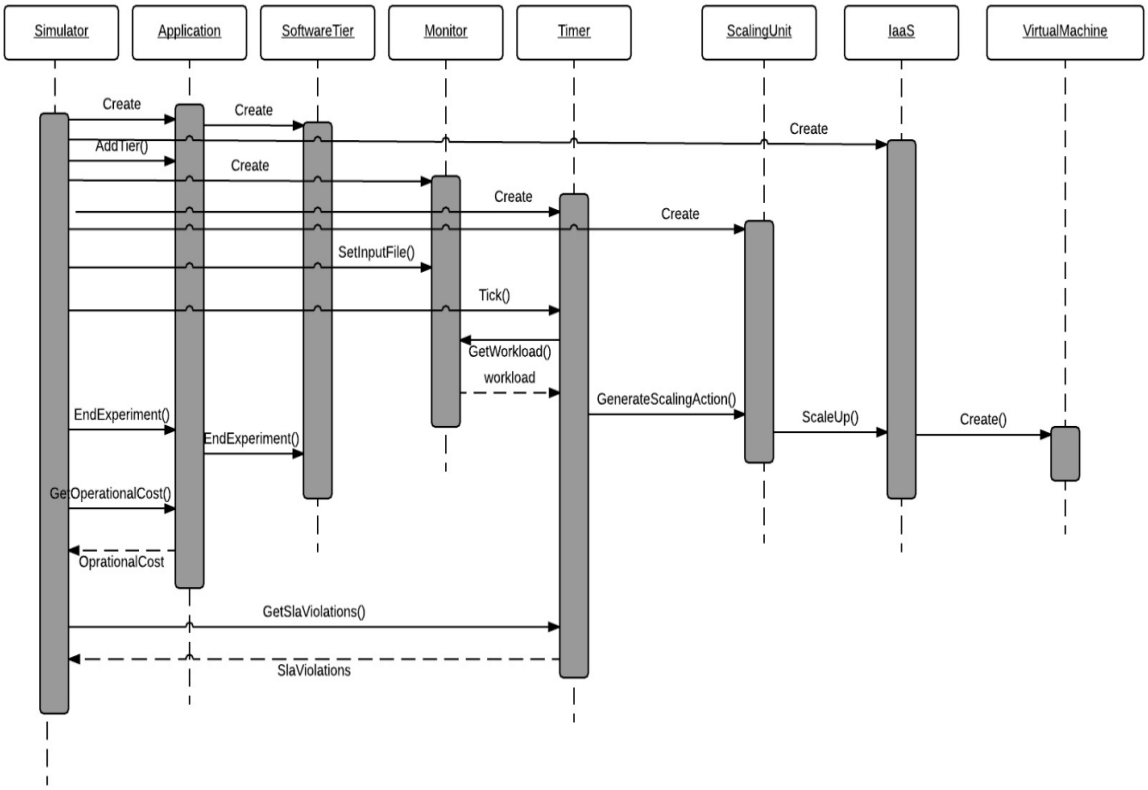


**Figure A-3. Simulation package sequence diagram**

## References


[1] A. Beloglazov and R. Buyya, “Adaptive threshold-based approach for energy-efficient consolidation of virtual machines in cloud data centers,” *Proceedings of. 8th International Workshop on Middleware Grids, Clouds e-Science*, , pp. 1-6, 2011.

[2] A. A. Bankole and S. A. Ajila, “Cloud client prediction models for cloud resource provisioning in a multitier web application environment,” *Proceedings of 7th IEEE International Symposium Service System Engineering*, pp. 156–161, 2013.

[3] S. Islam, J. Keung, K. Lee, and A. Liu, “Empirical prediction models for adaptive resource provisioning in the cloud,” *Journal of Future Generation Computer Systems*, vol. 28, no. 1, pp. 155–162, 2012.

[4] T. Lorido-Botran, J. Miguel-Alonso, and J. A. Lozano, “A review of auto-scaling techniques for elastic applications in cloud environments,” *Journal of Grid Computing*, vol. 12, no. 4, pp. 559–592, 2014.

[5] P. Mell and T. Grance, "The NIST definition of cloud computing recommendations of the National Institute of Standards and Technology," *U.S. Department of Commerce*, 2011.

[6] A. Y. Nikravesh, S. A. Ajila, and C.-H. Lung, “Measuring prediction sensitivity of a cloud auto-scaling system,” *Proceedings of 38th IEEE Annual International Computers, Software and Applications Conference Workshop*, pp. 690–695, 2014.

[7] G. D. C. Rodrigues, G. L. Dos Santos, V. T. Guimares, L. Z. Granville, and L. M. R. Tarouco, “An architecture to evaluate scalability, adaptability and accuracy in cloud monitoring systems,” *Proceedings of International Conference on*

*Information Networks*, pp. 46–51, 2014.

[8] S. Grove and N. Burns, *The Practice of Nursing Research: Conduct, Critique, and Utilization*, Edition 5th, Suanders publisher, 2004.

[9] M. Mao and M. Humphrey, "A performance study on the VM startup time in the cloud," *Proceedings of 5th IEEE International Conference on Cloud Computing*, pp. 423–430, 2012.

[10] S. Hu and J. E. Smith, "Reducing startup time in co-designed virtual machines," *Proceedings of International Symposium on Computer Architure*, pp. 277–288, 2006.

[11] *Xen hypervisor*. Retrieved April 2016 from: https://wiki.xen.org/wiki/Xen_Project_Software_Overview

[12] *VMware vSphere ESX and ESXi information center*. Retrieved August 2016 from: http://www.vmware.com/products/esxi-and-esx.html

[13] *Kernel based virtual machine.*Retrieved August 2016 from: http://www.linux-kvm.org/

[14] *Eucalyptus cloud.* Retrieved August 2016 from: http://www.eucalyptus.com/

[15] *OpenStack cloud software: Open-source software for building private and public clouds.* Retreived May 2015 from: http://www.openstack.org/

[16] (2012). *VMware vCloud director: Deliver complete virtual data centers for consumption in minutes.* Retreived from: http://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/products/vCloud/vmware-vcloud-director-datasheet.pdf

[17] *Amazon elastic compute cloud (Amazon EC2).* Retrieved August 2016 from:

http://aws.amazon.com/ec2

[18] *The httperf HTTP load generator.* Retrieved February 2014 from: http://code.google.com/p/httperf/

[19] D. Krishnamurthy and J. A. Rolia, "A synthetic workload generation technique for stress testing session-based systems," *Journal of IEEE Transactions on Software Engineering*, vol. 32, no. 11, pp. 868–882, 2006.

[20] (2013) *Greencloud: The green cloud simulator.* Retrieved from: http://greencloud.gforge.uni.lu/

[21] *CloudStone project by Rad Lab group.* Retrieved August 2016 from: http://radlab.cs.berkeley.edu/wiki/Projects/Cloudstone/

[22] *Rain workload toolkit.* Retrieved June 2013 from: https://github.com/yungsters/rain-workload-toolkit/wiki

[23] P. Barford and M. Crovella, "Generating representative Web workloads for network and server performance evaluation," *Journal of ACM SIGMETRICS Performance Evaluation Review*, vol. 26, no. 1, pp. 151–160, 1998.

[24] *Apache JMeter.* Retrieved August 2016 from: http://jmeter.apache.org/

[25] *ClarkNet HTTP trace (from the Internet traffic archive).* Retrieved August 2016 from: http://ita.ee.lbl.gov/html/contrib/ClarkNet-HTTP.html

[26] *World Cup 98 trace (From the Internet Traffic Archive).* Retrieved August 2016 from: http://ita.ee.lbl.gov/html/contrib/WorldCup.html

[27] *TPC-W benchmark.* Retrieved May 2016 from: http://www.tpc.org/tpcw/default.asp.

[28] *RUBiS: Rice University Bidding System.* Retrieved December 2015 from:

http://rubis.ow2.org/

[29] *WikiBench: A web hosting benchmark.* Retrieved June 2013 from: http://www.wikibench.eu

[30] A. Keller and H. Ludwig, "The WSLA framework: Specifying and monitoring service level agreements for web services," *Journal of Network System Management*, vol. 11, no. 1, pp. 57–81, 2003.

[31] S. Ran, "A model for web services discovery with QoS," *Journal of ACM Sigecom Exchange*, vol. 4, no. 1, pp. 1–10, 2003.

[32] Veenhof. B. (2015). *Workload patterns for cloud computing.* Retrieved August 2016 from: http://watdenkt.veenhof.nu/2010/07/13/workload-patterns-for-cloudcomputing/

[33] C. Fehling, F. Leymann, R. Retter, W. Schupeck, and P. Arbitter, "Cloud computing patterns: Fundamentals to design, build, and manage cloud applications", Edition 1st, Springer-Verlag Wien publisher. 2014

[34] E. Yaqub, R. Yahyapour, P. Wieder, A. I. Jehangiri, K. Lu, and C. Kotsokalis, "Metaheuristics-based planning and optimization for SLA-aware resource management in PaaS clouds," *Proceedings of 7th IEEE/ACM International Conference on Utility Cloud Computing,* pp. 288–297, 2015

[35] A. Biswas, S. Majumdar, B. Nandy, and A. El-Haraki, "Predictive auto-scaling techniques for clouds subjected to requests with service level agreements," *Proceedings of IEEE World Congress Services*, pp. 311–318, 2015

[36] J. Bi, H. Yuan, Y. Fan, W. Tan, and J. Zhang, "Dynamic fine-grained resource provisioning for heterogeneous applications in virtualized cloud data center,"

*Proceedings of 8th IEEE International Conference on Cloud Computing,* pp. 1–13, 2015

[37] B. Urgaonkar, P. Shenoy, A. Chandra, and P. Goyal, "Agile dynamic provisioning of multi-tier Internet applications," *Proceesings of 2nd International Conference on Autonomic Computing,* pp. 217–228, 2005

[38] H. Ghanbari, B. Simmons, M. Litoiu, and G. Iszlai, "Exploring alternative approaches to implement an elasticity policy," *Proceesings of 4th IEEE International Conference on Cloud Computing*, pp. 716–723, 2011

[39] Y. W. Ahn, A. M. K. Cheng, J. Baek, M. Jo, and H. H. Chen, "An auto-scaling mechanism for virtual resources to support mobile, pervasive, real-time healthcare applications in cloud computing," *Journal of IEEE Network*, vol. 27, no. 5, pp. 62–68, 2013.

[40] K. Kanagala and K. C. Sekaran, "An approach for dynamic scaling of resources in enterprise cloud," *Proceedings of 5th IEEE International Conference on Cloud Computing Technology Science*, pp. 345–348, 2013.

[41] P. Di Sanzo, D. Rughetti, B. Ciciani, and F. Quaglia, "Auto-tuning of cloud-based in-memory transactional data grids via machine learning," *Proceedings of 2nd Symposium Network Cloud Computing Applications*, pp. 9–16, 2012.

[42] F. J. A. Morais, F. V. Brasileiro, R. V. Lopes, R. A. Santos, W. Satterfield, and L. Rosa, "Autoflex: service agnostic auto-scaling framework for IaaS deployment models," *Proceedings of 13th IEEE/ACM International Symposium Cluster, Cloud, Grid Computing*, pp. 42–49, 2013

[43] M. Mao and M. Humphrey, "Auto-scaling to minimize cost and meet application

deadlines in cloud workflows," *Proceedings of International Conference on High Performance Computing Networking, Storage Analysis*, pp. 1-12, 2011.

[44] C.-C. Lin, J.-J. Wu, J.-A. Lin, L.-C. Song, and P. Liu, "Automatic resource Scaling based on application service requirements," *Proceedings of 5th IEEE International Conference on Cloud Computing*, pp. 941–942, 2012

[45] P. Wu and Y. Kao, "Computing resource minimization with content-aware workload estimation in cloud-based surveillance systems," *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing*, pp. 5017–5020, 2014

[46] A. Gandhi, M. Harchol-Balter, R. Raghunathan, and M. a. Kozuch, "Distributed, Robust auto-scaling policies for power management in compute intensive server farms," *Proceedings of 6th Open Cirrus Summit*, pp. 1–5, 2011

[47] N. Roy, A. Dubey, and A. Gokhale, "Efficient autoscaling in the cloud using predictive models for workload forecasting," *Proceesings of 4th IEEE International Conference on Cloud Computing*, pp. 500–507, 2011

[48] M. Sladescu, A. Fekete, K. Lee, and A. Liu, "Event aware workload prediction: A study using auction events," *Proceesings of Web Information Systems Engineering Conference,* pp. 368–381, 2012

[49] E. Caron, F. Desprez, and A. Muresan, "Forecasting for grid and cloud computing on-demand resources based on pattern matching," *Proceesings of 2nd IEEE International Conference on Cloud Computing Technology Science*, pp. 456–463, 2010

[50] F. Al-Haidari, M. Sqalli, and K. Salah, "Impact of CPU utilization thresholds and

scaling size on autoscaling cloud resources," *Proceesings of 5th IEEE International Conference on Cloud Computing Technology and Science*, pp. 256–261, 2013

[51] M. Z. Hasan, E. Magana, a. Clemm, L. Tucker, and S. L. D. Gudreddi, "Integrated and autonomic cloud resource scaling," *Proceesings of IEEE Network Operation Management Symposium*, pp. 1327–1334, 2012

[52] M. Tighe and M. Bauer, "Integrating cloud application autoscaling with dynamic VM allocation," *Proceesings of IEEE Network Operation Management Symposium*, pp. 1–9, 2014

[53] R. Hu, J. Jiang, G. Liu, and L. Wang, "KSwSVR: A new load forecasting method for efficient resources provisioning in cloud," *Proceesings of IEEE International Conference Service Computing*, pp. 120–127, 2013

[54] J. Jiang, J. Lu, G. Zhang, and G. Long, "Optimal cloud resource auto-scaling for web applications," *Proceesings of 13th IEEE/ACM International Symposium Cluster, Cloud, Grid Computing*, pp. 58–65, 2013

[55] Y. Wadia, R. Gaonkar, and J. Namjoshi, "Portable autoscaler for managing multi-cloud elasticity," *Proceesings of International Conference on Cloud Ubiquitous Computing Emerging Technology*, pp. 48–51, 2013

[56] F. Kabir and D. Chiu, "Reconciling cost and performance objectives for elastic web caches," *Proceesings of International Conference on Cloud Service Computing*, pp. 88–95, 2012

[57] W. Fang, Z. Lu, J. Wu, and Z. Cao, "RPPS: A novel resource prediction and provisioning scheme in cloud data center," *Proceesings of 9th IEEE International Conference on Service Computing*, pp. 609–616, 2012

[58] B. Bouterse and H. Perros, “Scheduling cloud capacity for time-varying customer demand,” *Proceesings of of International Conference on Cloud Networks,* pp. 35-43, 2012

[59] C. Huang, W. Hu, and C. Shih, “The improvement of auto-scaling mechanism for distributed database-A case study for MongoDB,” *Proceesings of 15th Asia-Pacific Network Operations and Management Symposium,* pp. 1–3, 2013

[60] L. R. Sampaio and R. V. Lopes, “Towards practical auto scaling of user facing applications,” *Proceesings of IEEE Latin American Conference on Cloud Computing Community*, pp. 60–65, 2012

[61] S. Wang and R. M. Summers, “Machine learning and radiology,” *Journal of Medical Image Analalysis*, vol. 16, no. 5, pp. 933–951, 2012.

[62] A. A. Bankole, “Cloud client prediction models for cloud resrouce provisioning in a multitier web application environment,” *Masters Thesis, Department of Systems and Computer Engineering, Carleton University*, 2013.

[63] V. Vapnik, “Principles of risk minimization for learning theory,” *Proceedings of Advanced Neural Information Processing Systems Conference*, pp. 831–838, 1992.

[64] M. Kantardzic, *Data Mining: Concepts, Models, and Algorithms*, Edition 2nd, Wiley-IEEE Press publisher, 2011.

[65] F. Aidu and A. Vapnik, “Estimating the probability density by the stochastic regularization method,” *Journal of Automatic Telemechanics*, vol. 4, 1989.

[66] V. N. Vapnik and A. Y. Chervonenkis, “Necessary and sufficient conditions for the uniform convergence of means to their expectations,” *Journal of Theory Probability,* vol. 3, no. 26, pp. 7–13, 2013.

[67] M. Sewell, "VC-Dimension," *Technical Report, Department of Comuter Science University of Collage London*, 2008.

[68] I. Yeh, P.-Y. Tseng, K.-C. Huang, and Y.-H. Kuo, "Minimum risk neural networks and weight decay technique," *Proceedings of 8th International Conference on Emerging Intelligent Computing Technology and Applications*, pp. 10–16, 2012.

[69] N. Sapankevych and R. Sankar, "Time series prediction using support vector machines: A survey," *IEEE Computing Intelligence Magazine*, vol. 4, no. 2, pp. 24–38, 2009.

[70] H. Wang and D. Hu, "Comparison of SVM and LS-SVM for regression," *Proceedings of International Conference on Neural Networks and Brain*, pp. 279–283, 2005.

[71] D. Garlan and B. Schmerl, "Model-based adaptation for self-healing systems," *Proceedings of the 1st Workshop on Selfhealing systems*, p. 27-32, 2002.

[72] R. Sterritt, B. Smyth, and M. Bradley, "PACT: Personal autonomic computing tools," *Proceedings 12th IEEE International Conference and Workshops on Engineering of Computer-Based Systems*, pp. 519–527, 2005.

[73] J. P. Bigus, D. a. Schlosnagle, J. R. Pilgrim, W. N. Mills III, and Y. Diao, "ABLE: A toolkit for building multiagent autonomic systems," *IBM Systems Journal*, vol. 41, no. 3, pp. 350–371, 2002.

[74] M. L. Littman, N. Ravi, E. Fenson, and R. Howard, "Reinforcement learning for autonomic network repair," *Proceedings of International Conference on Autonomic Computing* pp. 284–285, 2004.

[75] J. Dowling, R. Cunningham, E. Curran, and V. Cahill, "Building autonomic

systems using collaborative reinforcement learning," *Journal of Knowledge Engineering Review*, vol. 21, no. 03, pp. 231–238, 2006.

[76] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, Edition 3rd, Prentice Hall publisher, 2009.

[77] E. Asrts and P. J. Van Laarhoven, "Simulated annealing: Theory and applications", Kluwer Academic publishers, 1987.

[78] E. Talbi and T. Muntean, "Hill-climbing, simulated annealing and genetic algorithms: A comparative study and application to the mapping problem," *Proceeding of IEEE 26th Hawaii International Conference on System Science*, pp. 565-573, 1993.

[79] J. Yu and R. Buyya, "Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms," *Journal of Scientific Programing-Scientific Workflows*, vol. 14, no. 3, pp. 217–230, 2006.

[80] Obitko. M. (2012). *Introduction to genetic algorithms.* Retrieved from: http://www.obitko.com/tutorials/genetic-algorithms/about.php

[81] E. Lazowska, J. Zahorjan, S. Graham, and K. Sevcik, *Quantitative System Performance, Computer System Analysis Using Queueing Network Models*, Prentice-Hall publisher, 1984.

[82] E. Casalicchio and L. Silvestri, "Autonomic management of cloud-based systems: The service provider perspective," *IEEE Symposium on Computers and Communications*, pp. 161–166, 2013.

[83] D. Menasce, L. Dowdy, and V. Almeida, *Performance by Design: Computer Capacity Planning by Example*, Edition 1st, Prentice Hall publisher, 2004.

[84] Z. Xiao, Q. Chen, and H. Luo, "Automatic scaling of Internet applications for cloud computing services," *Journal of IEEE Transactions on Computing*, vol. 63, no. 5, pp. 1111–1123, 2014.

[85] *Amazon scaling based on metric*. Retrieved August 2016 from: http://docs.aws.amazon.com/autoscaling/latest/userguide/policy_creating.html#policy-updating-console

[86] L. R. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," *IEEE Journal of Readings in Speech Recognition*, vol. 77, no. 2, pp. 257–286, 1989.

[87] Z. Ghahramani, "An introduction to hidden Markov models and Bayesian networks," *International Journal of Pattern Recognittion Artificial Intelligence*, vol. 15, no. 01, pp. 9–42, 2001.

[88] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software," *Newsletter of ACM SIGKDD Explorations*, vol. 11, no. 1, pp. 10-18, 2009.

[89] *HMM Weka extension*. Retrieved June 2016 from: http://www.doc.gold.ac.uk/~mas02mg/software/hmmweka/index.html

[90] I. H. Witten, E. Frank, and M. a. Hall, *Data Mining: Practical Machine Learning Tools and Techniques,* Edition 3rd, Morgan Kaufmann publisher. 2011.

[91] J. Williams and F. Burton, "Sensitivity analysis in model-driven engineering," *Proceedings of 15th International Conference on Model Driven Engineering Languages and Systems*, pp. 743–758, 2012.

[92] *Amazon EC2 pricing.* Retrieved June 2016 from:

https://aws.amazon.com/ec2/pricing/

[93] M. Safe, J. Carballido, I. Ponzoni, and N. Brignole, "On stopping criteria for genetic algorithms," *Proceedings of 17th Brazilian Symposium on Advanced Artificial Intelligence,* pp. 405–413, 2004.

[94] M. Schmid, M. Thoss, T. Termin, and R. Kroeger, "A generic application-oriented performance instrumentation for multi-tier environments," *Proceedings of 10th IFIP/IEEE International Symposium on Integrated Network Management*, pp. 304–313, 2007.

[95] G. Franks, D. Petriu, M. Woodside, J. Xu, and P. Tregunno, "Layered bottlenecks and their mitigation," *Proceedings of 3rd International Conference on the Quantitative Evaluation of Systems,* pp. 103–112, 2006.

[96] *Docker*. Retrived July 2016 from: https://www.docker.com/

[97] Woodside. M. (2015). *Layered queueing network solver software package.* Retrived from: http://www.sce.carleton.ca/rads/lqns/.

[98] G. Franks, P. Maly, M. Woodside, D. Petriu, A. Hubbard, and M. Mroz, "Layered queueing network solver and simulator user manual," *Department of Systems and Computer Engineering, Carleton University,* 2013, Available: http://www.sce.carleton.ca/rads/lqns/LQNSUserMan-jan13.pdf. [Accessed: 27-Apr-2016].

[99] Nikravesh. A. (2016). *Carleton auto-scaling simulator*. Retrived from: https://github.com/alinikravesh/Carleton-Auto-Scaling-Simulator/.