TOWARDS A NOVEL BIOLOGICALLY-INSPIRED CLOUD ELASTICITY FRAMEWORK

AMJAD ULLAH

Doctor of Philosophy

Division of Computing Science and Mathematics
University of Stirling

August 2017

Dedicated to

My beloved Parents, Sisters and Brothers

For their love, encouragement, prays and patience

DECLARATION

I, Amjad Ullah, hereby declare that the work in this thesis is original and has been composed by myself, except where reference is made to other works, and has not been submitted for examination for any other degree at this university or any other learning institutions.

Stirling, August 2017

Amjad Ullah

ABSTRACT

With the wide spread use of the Internet, the popularity of web applications has significantly increased. Such applications are subject to unpredictable workload conditions that vary from time to time. For example, an e-commerce website may face higher workloads than normal during festivals or promotional schemes. Such applications are critical and performance related issues, or service disruption can result in financial losses. Cloud computing with its attractive feature of dynamic resource provisioning (elasticity) is a perfect match to host such applications. The rapid growth in the usage of cloud computing model, as well as the rise in complexity of the web applications poses new challenges regarding the effective monitoring and management of the underlying cloud computational resources. This thesis investigates the state-of-the-art elastic methods including the models and techniques for the dynamic management and provisioning of cloud resources from a service provider perspective.

An elastic controller is responsible to determine the optimal number of cloud resources, required at a particular time to achieve the desired performance demands. Researchers and practitioners have proposed many elastic controllers using versatile techniques ranging from simple if-then-else based rules to sophisticated optimisation, control theory and machine learning based methods. However, despite an extensive range of existing elasticity research, the aim of implementing an efficient scaling technique that satisfies the actual demands is still a challenge to achieve. There exist many issues that have not received much attention from a holistic point of view. Some of these issues include: 1) the lack of adaptability and static scaling behaviour whilst considering completely fixed approaches; 2) the burden of additional computational overhead, the inability to cope with the sudden changes in the workload behaviour and the preference of adaptability over reliability at runtime whilst considering the fully dynamic approaches; and 3) the lack of considering uncertainty aspects while designing auto-scaling solutions. This thesis seeks solutions to address these issues altogether using an integrated approach. Moreover, this thesis aims at the provision of qualitative elasticity rules.

This thesis proposes a novel biologically-inspired switched feedback control methodology to address the horizontal elasticity problem. The switched methodology utilises multiple controllers simultaneously, whereas the selection of a suitable controller is realised using an intelligent switching mechanism. Each controller itself depicts a different elasticity policy that can be designed using the principles of fixed gain feedback controller approach. The switching mechanism is implemented using a fuzzy system that determines a suitable controller/-policy at runtime based on the current behaviour of the system. Furthermore, to improve the possibility of bumpless transitions and to avoid the oscillatory behaviour, which is a problem commonly associated with switching based control methodologies, this thesis proposes an alternative soft switching approach. This soft switching approach incorporates a biologically-inspired Basal Ganglia based computational model of action selection.

In addition, this thesis formulates the problem of designing the membership functions of the switching mechanism as a multi-objective optimisation problem. The key purpose behind this formulation is to obtain the near optimal (or to fine tune) parameter settings for the membership functions of the fuzzy control system in the absence of domain experts' knowledge. This problem is addressed by using two different techniques including the commonly used Genetic Algorithm and an alternative less known economic approach called the Taguchi method. Lastly, we identify seven different kinds of real workload patterns, each of which reflects a different set of applications. Six real and one synthetic HTTP traces, one for each pattern, are further identified and utilised to evaluate the performance of the proposed methods against the state-of-the-art approaches.

ACKNOWLEDGMENTS

This work has not been accomplished, if I was not getting help and support of many people. Here, I would like to exploit the opportunity to express my sincere and deepest gratitude to all of them for their direct or indirect contribution to this achievement. My sincere gratitude and deepest appreciation goes to my principal supervisor Dr. Jingpeng Li, firstly, for his confidence and acceptance of me as a PhD student even after knowing that I lost two year of my PhD life already. Secondly, for his constant guidance and support throughout of my PhD study, without which I wouldn't be able to produce this thesis. Dr Li, insightful comments, suggestions and his frequent availability has been very helpful and I greatly appreciate that. I would also like to express my appreciation to Prof Amir Hussain for all his support, insights and also for all those non-technical discussions, which I must admit is equally important and was a great source of learning.

I am indebted to the Scottish Institute of Computer Alliance (SICSA) and Division of Computing Science and Mathematics of University of Stirling for granting me the PhD scholarship. I also appreciate the grant, I received in the form of six months stipend from DAASE project and therefore, I am very thankful to the DAASE research group members, who approved this and again to Dr Li for his initiation and chasing. I must not forget the role of Dr Andrea Bracciali and Dr Marwan Fayad for their constant interactions, discussions, valuable suggestions while applying for the PhD scholarship that enabled me to begin my PhD journey. My sincere gratitude and thanks goes to Dr Bracciali for all his efforts and guidance in my initial two years of PhD, which has significantly contributed in the completion of this PhD thesis.

I must also offer my sincere thanks to my department colleagues and staff members in the School of Computing Science and Mathematics at the University of Stirling. I would particularly like to thank Dr. Marwan Fayad for his support and encouragement; Dr. Kamran Farooq for his constant support; Dr. Paul Mc-Menemy for helping with review and discussions related to mathematics; Grace, Lynn, Gemma, and Linda for helping with a number of administrative stuff; Sam and Graham for their IT support. I also want to acknowledge the company and support of all my friends from Stirling, Glasgow and Pakistan.

Finally and more importantly, I would like to express my thanks and gratitude to my family whose love, continuous support and encouragement were the key motivating factors for the completion of this hard and challenging work. I am sure, I couldn't have done it, If it wasn't their support, prayers, motivation and patience. During all this period, they have freed me completely from all responsibilities and only let me concentrate on my PhD. Therefore, I must say that, you all are very important for me and I am eternally grateful to have you all in my life.

LIST OF PUBLICATIONS

During the period of this research, the following papers have been produced:

- Amjad Ullah, Jingpeng Li and Amir Hussain. "Towards workload-aware cloud resource provisioning using a multi-controller fuzzy switching approach", *International Journal of High Performance Computing and Networking*, Inderscience UK, (in-press), December 2016. Materials from this paper is included within this thesis in Chapter 4, and 5.
- Amjad Ullah, Jingpeng Li, Amir Hussain and Erfu Yang. "Towards a biologically-inspired soft switching approach for cloud resource provisioning", *Cognitive Computation*, Springer, Vol. 8, Issue. 5, pp. 992-1005, October 2016. Materials from this paper is included within this thesis in Chapter 6.
- Amjad Ullah, Jingpeng Li, Yindong Shen and Amir Hussain. "Genetic optimization of fuzzy membership functions for cloud resource provisioning", *in Proceedings of 2016 IEEE Symposium Series on Computational Intelligence (SSCI)* (SSCI 2016), Athens, Greece: IEEE, December 2016 pp. 1-8. Materials from this paper is included within this thesis in Chapter 7.
- Amjad Ullah. "Towards Workload-aware Fine-grained Control over Cloud Resources: Student Research Abstract", *in Proceedings of the 31st Annual ACM Symposium on Applied Computing* (SAC 2016). New York, USA: ACM, April 2016 pp. 488-489.
- Amjad Ullah, Jingpeng Li, Yindong Shen and Amir Hussain. "A Control Theoretical View of Cloud Elasticity: Taxonomy, Survey and Challenges", *Cluster Computing*, Springer, (in-review), August 2017. Materials from this paper is included within this thesis in Chapter 3.
- Amjad Ullah, Jingpeng Li, Yindong Shen and Amir Hussain. "Optimising fuzzy membership functions for cloud resource provisioning using the

Taguchi approach", *Computing*, Springer, (in-review), June 2017. Materials from this paper is included within this thesis in Chapter 7.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

API	Application Programming Interface
ARX	Autoregressive Exogenous Model
AWS	Amazon Web Service
BG	Basal Ganglia
CPs	Cloud Providers
DOF	Degree of Freedom
EC2	Elastic Compute Cloud
FIS	Fuzzy Inference System
GA	Genetic Algorithm
IaaS	Infrastructure as a Service
IT	Information Technology
LLC	Limited Lookahead Controller
LPV	Linear Parameter Varying
LQR	Linear Quadratic Regulator
MIMO	Multi Input Multi Output
MMST	Multi-Modal Switching and Tuning
MPC	Model Predictive Controller
mRT	Mean Response Time
NIST	National Institute of Standards and Technology

OA	Orthogonal Array
PaaS	Platform as a Service
PI	Proportional Integral
PID	Proportional Integral Derivative
QoS	Quality of Service
RPD	Relative Percentage Deviation
rpm	Requests Per Minute
SaaS	Software as a Service
SID	System Identification
SISO	Single Input Single Output
SLA	Service Level Agreement
SLO	Service Level Objective
SPs	Service Providers
SSF	State-space Feedback
VMs	Virtual Machines

Part I

INTRODUCTION

1 INTRODUCTION

1.1 CONTEXT

The advent of cloud computing has reshaped the way Information Technology (IT) is used to design, deliver and operate software. Businesses or Service Providers (SPs) no longer require to purchase and maintain expensive hardware needed to run their software, rather they exploit the computational infrastructure provided by Cloud Providers (CPs). This model of computing provided by CPs facilitates the usage of computational infrastructures such as networking, storage and computation as a utility on a pay per usage basis, thus fulfilling the long-held dream of computing as a utility [2]. The SPs, by adopting such a model of computation, can reduce or eliminate their upfront computational infrastructure cost as well as maintenance costs. Moreover, it allows them to be more focused on their primary objectives [8].

The CPs are responsible for provision and management of the computational resources, whereas the SPs are consumers who rent these resources as required for their needs, e.g. deploying their applications (or services) for the end users, storage, computational requirements, etc. With the permeation of the Internet, the adoption of cloud computing has significantly and rapidly increased both in the industrial and academic worlds [9]. This increased shift towards cloud computing has resulted in the development of large-scale computing data centres to satisfy these growing needs. Such data centres contain an extensive number of computing machines that consume an enormous amount of energy, i.e. approximately 1.5% of the total electricity usage worldwide [10]. Moreover, they also release a large quantity of CO_2 into the environment [11], which was estimated as 0.6% of the global total in 2008 and this number is expected to rise to 2.6% in 2020 [12].

The key aim of a cloud computing model is to use the computational resources as efficiently as possible to reduce energy consumption as well as operational costs. However, this goal is still challenging as is evident from the low utilisation of most data centres, which is estimated as less than 30 percent [13]. The efficient use of cloud resources can be viewed from two perspectives: (1) The CPs aim to increase the efficiency of their under-utilised computational nodes by shutting down some servers and shifting their load to others, hence minimising energy consumption to reduce electricity costs as well as CO_2 emission; alternatively, CPs could oversubscribe their resources, hence maximising their revenue; (2) the SPs are concerned with the efficient use of their rented computational resources, so that they can release any under-utilised or unused Virtual Machines (VMs) to reduce their service operating costs.

This thesis explores the cloud resource management by focusing on the dynamic resource provisioning (elasticity) feature of cloud computing. The thesis goes on to investigate the current state-of-the-art elastic methodologies, highlighting the open issues and proposing new methods to contribute towards resolving the identified problems.

1.2 BACKGROUND AND MOTIVATION

With the widespread use of the Internet, the popularity of web applications such as social networking, wikis, news portals and e-commerce applications has increased significantly. Such applications are subject to unpredictable workload conditions that vary across time. For example,

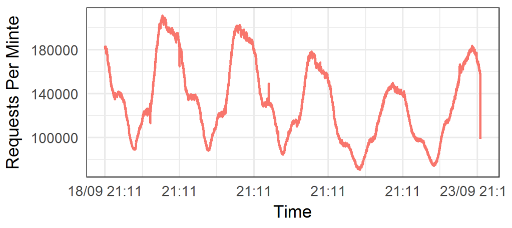
1. The higher than usual workload on e-commerce websites during festivals or promotional schemes such as Amazon's Christmas sale [14], China's recent 'singles day' sale [15], etc.
2. Facebook experienced a 10-fold increase in their active users within a span of three hours [16].
3. The diurnal pattern of web applications where the workload arrival rate during day time is higher than night, e.g. Wikipedia website traffic [17] (see Figure 1.1a).
4. The traffic on Al-Jazeera's news website observed an increase of 2,500% during the fourth day of the Egyptian revolution in 2011 [18].
5. The increase of workload on Animoto (an image processing web application) in April 2008 raised the Amazon based cluster size from 50 VMs to 4000 VMs within the span of just three days followed by a sharp decline back to normal [19].

Such applications are critical to each specific business, and the SPs do not want their applications to suffer from any performance related issues or service disruption. The degradation in the Quality of Service (QoS) of such applications has a direct consequence in the loss of customers and ultimately results in financial losses [20]. For example, every 100 ms of latency costs Amazon 1% in sales [21]. Similarly for Google, traffic drops by 20%, when page loading takes longer than 500 ms [20].

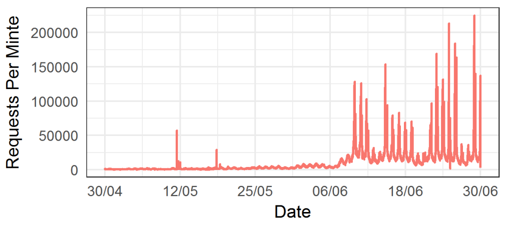
The examples mentioned above indicate that the real world web applications are subject to dynamically varying workload conditions that change from time to time. In some cases, the workload may follow a particular pattern, e.g. the diurnal pattern as in Wikipedia trace [17] given in Figure 1.1a, whereas in other cases, the pattern cannot be determined or predicted, e.g. the flash crowd behaviour in the case of the FIFA world cup trace [22] that can be seen in Figure 1.1b. The fluctuating workload conditions of such applications indicate different needs of computational resources at different times [23, 24].

The pool of virtually unlimited on-demand computational resources and many attractive features of cloud computing, such as pay-as-you-go pricing and on-the-fly re-adjustment of hired computational resources (elasticity), can be a perfect match to host web applications, as they are subject to fluctuating workload behaviours. The cloud's elasticity allows applications to dynamically adjust the underlying computational resources in response to the changes observed in the environment, thus enabling SPs to fulfil the changing application demands by paying only for the resources they are using [25]. The elasticity is of two types, i.e. *Horizontal* and *Vertical*. The *Horizontal* enables increases or decreases in the number of VMs, whereas, *Vertical* elasticity allows changes in the specification of existing VMs, e.g. the increase or decrease in CPU and/or memory capacity of one or a set of VMs.

The SPs have to provide an auto-scaling policy to exploit this type of elastic model. Such a policy is responsible for making runtime resource provisioning decisions. Over the years, researchers and practitioners have proposed many elastic methods using versatile techniques including but not limited to rule-based [26, 27, 28, 29, 30], control theory [31, 32, 33, 34], fuzzy logic [16, 35], optimisation [36, 37, 38] and machine learning [21, 39]. However, despite a large range of existing elasticity research work, the aim of implementing an efficient scaling technique that satisfies the actual demands is still a challenge to achieve [40, 41, 42]. This is evident from the low utilisation, estimated as 8% to 20%, of the server capa-



(a) Wikipedia website trace log



(b) FIFA world cup website trace log

Figure 1.1: Real workload examples

city purchased by the SPs [43]. The existing research literature on cloud elasticity differs in various aspects, e.g. triggering behaviour (Reactive/Predictive/Hybrid), scope (CPs/SPs perspective), dependency on metrics (CPU utilisation/Response time, etc.), and the implementation technique (Control Theory/Rule-based, etc.). Despite such differences most of the existing methods can generally be grouped into *Fixed* or *Adaptive* based on their design and working mechanism to analyse their pros and cons as a whole [44].

The *Fixed* class refers to the family of all elastic methods, which are designed off-line and remain fixed at runtime, whereas the *Adaptive* class refers to those methods which are equipped with an online learning capability that is responsible for adaptation at runtime in response to changes in the working environment. The *Fixed* approaches are simple, easy to design and better for systems with uniform workload behaviour, e.g. rule-based systems and fixed gain elastic feedback controllers. However, the performance severely affects for systems with variable workloads due to lack of adaptability at runtime. In contrast, the *Adaptive* approaches are more flexible, due to online learning capabilities, and perform better for systems with a slowly varying workload. However, they are also criticised for their additional computational cost caused due to the online learning [45], long training delays, its associated risk of reducing the quality assurance of the resulted system and the impossibility of deriving a convergence or stability proof [44]. Moreover, they are unable to cope with the sudden changes in the workloads.

In contrast to the families mentioned above, this thesis advocates a fixed-adaptive (also referred to as *Hybrid* by Gambi et al. [44]) approach, a method commonly associated with the biologically-inspired Multi-Modal Switching and Tuning (MMST). Using such an approach, an elastic method follows a *Fixed* design principle and also achieves certain level of adaptive behaviour at runtime. The review of existing state-of-the-art elasticity research (presented in Chapter 3) indicates that such an approach for implementing cloud elasticity has not received much attention. Another important consideration identified in the existing elasticity literature is the importance of addressing the uncertainty related issues, e.g. impreciseness in domain knowledge and noise in monitoring data. Jamshidi et al [16, 46] and Farokhi et al [47] stressed the importance of the uncertainty aspects to be taken into consideration while designing the elastic controller. However, despite the importance, the implementation of uncertainty in the context of cloud elasticity has not yet been well received [47]. This thesis is a step forward in this direction.

1.3 THESIS STATEMENT

This thesis addresses the horizontal elasticity problem and particularly focuses on contributing towards resolving the issues in the existing elasticity literature. The issues includes: (1) The lack of adaptability and static scaling behaviour whilst considering completely fixed approaches; (2) The burden of additional computational overhead, the inability to cope with sudden changes in workload behaviour and preference of adaptability over reliability at runtime whilst considering the fully dynamic approaches; (3) The lack of considering uncertainty aspects while designing auto-scaling solutions; (4) Lastly, the lack of providing qualitative elasticity rules to resolve the corresponding quantitative nature of the commonly used rule-based approaches.

There exist some limited elastic methodologies that address the aforementioned issues separately. In contrast, this thesis investigates the feasibility to address the aforementioned issues altogether using a single integrated approach. This research investigates the possibility of developing an elastic method using fixed-adaptive principle that incorporates the inherent uncertainty aspects present in the cloud environment and facilitates the composition of qualitative elasticity rules. For such a methodology, this thesis investigates the synergy between the biologically-inspired multi-controller approach and fuzzy control system to achieve the desired adaptability behaviour in the presence of uncertainty and reliability due to the statically designed nature of the control methodology.

Biological systems have the ability to identify and analyse a naturally occurred situation, and trigger an appropriate response action based on their existing knowledge of different behaviours particularly suited to various situations [48]. This results in an efficient and quick adaptation of the system to a changing environment. This phenomenon motivated methods like MMST [48] where adaptive behaviour is achieved using a repertoire of multiple models and the best model selection mechanism at runtime. Based on a similar principle to MMST, this thesis proposes a switched feedback control methodology to address the problem of horizontal elasticity. The switched controller utilises multiple controllers simultaneously, and the selection of a suitable controller is realised using an intelligent switching mechanism at runtime.

Each of the controllers is particularly designed to suit a different situation. A situation in this context represents the requirement of a scaling action. In other words, each controller determines the number of VMs needed at any point in time but at a different level of intensity, which responds appropriately to the changes in the environment. Each controller depicts a different elasticity policy and can be designed using the principles of fixed gain feedback controller approach. The selection of a suitable controller is achieved using a fuzzy system at runtime based on the system behaviour at that point in time. We will refer to this approach as *Hard switching* from this point onward. Using such an approach, we aim to achieve the required adaptive behaviour by switching among the controllers in response to the changes in the application workloads without the involvement of any online estimation technique, thus avoiding additional computational overhead.

The fuzzy system makes use of the latest status of different metrics including CPU utilisation, application Response time and workload Arrival rate to determine the suitable controller at runtime. The inclusion of these metrics into the decision-making mechanism covers three different aspects of the system: application performance, disturbance and system resource utilisation. Covering these aspects makes the proposed elastic methodology *Hybrid*, a term coined by Farokhi et al. [49] to differentiate from other methodologies that are either capacity-based (relying only on system metrics) or performance-based (relying only on application metrics).

The aim of using the fuzzy system is to capture the uncertainty aspects associated with the cloud environment, as a deployed application is subject to uncertainty related challenges [50]. The various cloud-related uncertainty aspects include impreciseness in domain knowledge, noise in monitoring data, inaccuracies in the performance model, and unpredictability in the workload. The explicit consideration of such uncertainty aspects is important whilst designing an elastic method. Otherwise, scaling decisions often result in unreliability as the available resources may fail to fulfil the requirements, or may not be cost-effective [47]. However, despite the importance, the implementation of uncertainty in the context of cloud elasticity has not yet been well received in the existing research [47]. Therefore this thesis utilises a fuzzy system due to its natural ability to deal with scenarios, where the system knowledge is imprecise, uncertain and highly dynamic [51]. The second important issue that the fuzzy system tackles is the provision of qualitative elasticity rules in contrast to the quantitative nature of the commercially available rule-based elasticity mechanisms. The existing elasticity rules require detailed knowledge of the system to set the different thresholds of the rule quantitatively.

The existing work on considering uncertainty and qualitative rules are very limited. Jamshidi et al. [16] initially highlighted and proposed a new fuzzy logic based methodology using a set of qualitative fuzzy rules. However, their approach itself suffers from the static scaling behaviour. This thesis compliments and extends their idea of incorporating the uncertainty prospects and qualitative elasticity rules mentioned above.

The proposed *Hard switching* methodology achieves the required adaptive behaviour using switching among the controllers at runtime. However, such a method is often criticised for an associated unwanted behaviour, commonly termed as bumpy transition, that could lead the system to an oscillatory state [52, 53, 54], where the cloud resources can be acquired/released periodically. The occurrences of bumpy transition can be due to an inappropriate switching or a larger change in the system state. Oscillation is undesirable, and therefore, it should be avoided. For this purpose, we also propose a *Soft switching* approach. Different to the *Hard switching* method, the *Soft switching* approach allows the possibility of selecting more than one controller. In such a case, the final decision will consider the output of all selected controllers. The *Soft switching* approach integrates a biologically-inspired Basal Ganglia (BG) based computational model. This model is built on the functional anatomy of BG, which is a central switching mechanism in animal's brain. The key aim of *Soft switching* methodology is to improve the possibility of bumpless transitions and to avoid any oscillatory behaviour.

1.5 CONTRIBUTIONS

This thesis proposes novel, biologically-inspired, elastic methodologies. For the evaluation of this research, we have used and extended where necessary a well-known cloud simulation environment called CloudSim [55]. Moreover, we have adopted real HTTP traces based on the commonly used workload patterns as test scenarios to demonstrate the suitability and effectiveness of the proposed methodologies for dynamic web applications. The key contributions of this thesis are summarised as following:

1. A state-of-the-art review is conducted to understand and analyse the existing control theoretical based elasticity solutions. The review is carried out considering a novel proposed taxonomy and classification that highlights aspects of both control theory and cloud elasticity domain. Its key purpose

is to fill the gap in the existing literature, where the existing reviews mostly focus on elasticity based classification. We review the existing solutions based on the various types of control methodologies and highlight the open issues and challenges.

2. A novel, multi-controller based feedback control methodology is proposed to address the problem of horizontal elasticity, where each controller works well in a different situation/operating region and the selection of a suitable controller is realised at runtime. There is no standard mechanism to partition a system among multiple sub-models/controllers [56]. Therefore in the absence of such a mechanism, this thesis utilises the domain experts' based classification of workload intensity as a criteria for the partitioning of the system among multiple controllers.

3. This thesis develops a fuzzy system based switching mechanism that determines the selection of a suitable controller at runtime. This mechanism accepts various system and application level metrics as inputs. The use of the fuzzy system is motivated due to its ability to deal with uncertainty that arises at runtime due to various factors, e.g. impreciseness in domain knowledge, and noise in monitoring data. Moreover, it facilitates the composition of qualitative elasticity rules.

4. A soft switching mechanism, where the selection process for suitable controllers is formulated as an action selection problem. This mechanism integrates a biologically-inspired BG based computational model of action selection [57, 58]. The soft switching approach aims at exploring the possibility to improve the bumpless transition and to avoid the oscillatory behaviour.

5. A detailed system evaluation is performed in comparison with the state-of-the-art approaches using various real workload traces, whereas the evaluation criteria consist of the various aspects including performance, cost, and oscillatory behaviour.

6. This thesis formulates the problem of designing the membership functions of the switching mechanism as a multi-objective optimisation problem. The key objective of this work is to obtain the near optimal parameter settings for the membership functions of the fuzzy control system using multi-objective GA.
7. The near optimal design problem of fuzzy membership functions is also addressed with an alternative, less known approach called the *Taguchi* method. The use of this approach is employed considering the scenarios where a larger exploration of search space, usually required by GA, is not feasible.

1.6 THESIS STRUCTURE

This thesis consists of 8 chapters. The summary of each chapter is provided below:

- **Chapter 1** introduces the thesis and explains the context, motivation and the contributions of this research work.
- **Chapter 2** explains the related concepts to establish a necessary understanding and foundation of this work. More specifically, it introduces and describes the technical view of cloud elasticity, target system and the basics of related implementation techniques, upon which the proposed work is based.
- **Chapter 3** reviews the state-of-the-art control theoretical based approaches for cloud elasticity. This chapter initially summarises the existing survey based research work to establish the need for reviewing the literature from a new perspective required for the understanding of control theoretical view of cloud elasticity. It proposes a novel taxonomy to carry out the review. An exhaustive review of the existing control theoretical approaches proposed for elasticity problem is carried out considering the proposed taxonomy.

Lastly, it summarises and discusses some open issues and challenges to lay the foundation for the focus of this thesis.

- **Chapter 4** presents the proposed *Hard switching* framework. This chapter starts by identifying the requirements based on the analysis of existing work that leads to the proposition of this framework. It then discusses the main concept of the framework, followed by the details and design of its components including the feedback control and the switching mechanism. The section on feedback control component includes the necessary details on the essential elements of the proposed control methodology, the SID experimentations and controllers' design. Whereas the section on switching mechanism presents the design of the proposed FIS including details on domain knowledge, membership functions and fuzzy rules.

- **Chapter 5** implements experimentations to evaluate the *Hard switching* framework against some state-of-the-art auto-scaling approaches. Firstly, it introduces the self-customised experimental environment, which utilises a well-known cloud simulation environment called *CloudSim* and an external Java-based library called *JFuzzyLogic*. Secondly, the chapter provides details about the various workload patterns and the real HTTP traces utilised for the experimentations. Moreover, the details on the evaluation criteria and methods used for comparison are provided. Lastly, the chapter presents the obtained computational result and discusses them in light of the pre-defined criteria to evaluate the performance of the proposed framework against the benchmark methods.

- **Chapter 6** covers the proposed biologically-inspired *Soft switching* approach. The chapter starts with the motivation of the soft switching method. It then introduces the basics of action selection, followed by a description of BG and the proposed BG based *Soft switching* mechanism. Finally, it reports the experiments of this enhanced method and discusses the obtained computational results in comparison with that of *Hard switching* approach.

- **Chapter 7** formulates the construction of the fuzzy membership functions, required for the implementation of switching mechanism, as a multi-objective optimisation problem to explore the near optimal parameter settings for their design. This problem is addressed using two different techniques including the commonly used multi-objective GA and an alternative, less known approach called the *Taguchi* method. The chapter starts with the motivation followed by the details of each employed technique in the context of the underlying problem. Finally, it reports the experiments carried out with both techniques and discuss the obtained computational results comparatively.
- **Chapter 8** concludes this thesis. It provides a summary of the research undertaken, reviews the contributions made and discusses how they are addressed. Finally, limitations of the research in this thesis and possible future research directions are discussed.

2

BACKGROUND AND FOUNDATION

This chapter provides background of the cloud elasticity and explains the related concepts to establish the necessary understanding and foundation of this thesis. Section 2.1 provides basic overview of cloud computing followed by a technical overview of the elasticity. Related concepts including an introduction to web applications, QoS and workload are explained in Section 2.3. Finally, Section 2.4 describes the elasticity implementation techniques, used to develop the methods proposed in this thesis.

2.1 THE CLOUD COMPUTING

2.1.1 *Historical View*

The term cloud computing was initially coined around 2006, referring to a new model of computation that provides on-demand pay-as-you-go based computational, storage and network services over the internet. It provides a practical shape to the idea of utility computing [2]. The idea of using computing as a utility dates back to 1961, when John McCarthy in his speech at MIT Centennial presciently stated *"If Computers of the kind I have advocated becomes the computers of the future, then computing may someday organise as a public utility just as the telephone system is a public utility."*. However, this idea did not attract much attention until the introduction of grid computing in mid-1990s.

Grid computing was initiated based on the concept of the electric power grid, which led to the formation of a computing grid, that could provide computing power on demand to consumers [59]. Such ideas were famous only in the scientific community, and as a result, various large-scale federated grid systems such as TeraGrid [60], Earth science grid [61] and Open science grid [62] were developed [59]. These grids provide computing power as well as various software services on demand. These systems, however, were predominantly used by the research and scientific community only. As such grid computing could not attract famous commercial giants for investment or adoption. Hence, no commercial grid computing providers have emerged thus far [59].

Subsequently, in 1999 a commercial company named *Salesforce* initiated a web-based service for the deployment of enterprise-level applications over the Internet. Amazon adopted the same idea in 2002 when they introduced Amazon Web Service (AWS) that provided access to computation and storage using the Internet. This release was followed by the launch of the first commercially available cloud infrastructure named Amazon Elastic Computing Cloud in 2006 [63]. Owing to its economic viability and commercial benefits, many big IT companies like Google and Microsoft immediately followed this paradigm shift of distributed computing and cloud enabled their existing hardware and software infrastructure [64].

2.1.2 *Definition and Key Benefits*

Cloud computing is defined in many different ways, and there is little consensus about it [65]. However, the following definition provided by National Institute of Standards and Technology (NIST) [66] is predominantly accepted both in the IT industry as well as in academia. *"Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction."*. This definition identifies the following five key characteristics of cloud computing [66]:

1. On-demand self-service: This facilitates provisioning of the computational resources on demand without human interaction.

2. Broad network access: The availability of computational resources that can be accessed through a standard mechanism over the Internet using any computing device, e.g. Laptop, Tablet and Mobile, etc.

3. Resource pooling: The computing and storage resources are pooled together that are further used by multiple users through the multi-tenant model.

4. Rapid elasticity: The computational resources are acquired and released dynamically. From the cloud consumer perspective, these resources appear to be unlimited and can be obtained at any point in time.

5. Services measurement: The cloud systems provide monitoring services that help to measuring resource usage. The cloud consumers have access to the resources measurement reports. These reports help in assisting further decision making process for both the stakeholders, i.e. cloud provider as well as the consumer of the service.

These characteristics of the cloud computing infrastructure entails the following key benefits from both the SPs and CPs perspectives:

- The on-demand and unlimited availability of computational resources provide a sense of relief to the SPs. Thus avoiding the risk associated with setting up the new business regarding infrastructure facilities needed for the deployment of any computational service.

- Whilst adopting the cloud computing model, the SPs can eliminate up-front infrastructure and maintenance cost required for the deployment and running of their services.

- The SPs by adopting this model of computing shifts the responsibilities to the CPs regarding infrastructure maintenance, backups, recovery and necessary support services. This helps the SPs to focus more on their actual pursuit that includes the improvement and maintenance of their applications and services.

- The pay-as-you-go pricing model of cloud computing enables to pay only for the resources used.
- The rapid elasticity reacts to the changes in demand of the resources. Hence it facilitates an efficient management of the hired computational resources at run time. The rapid elasticity helps in improving the performance of the system by acquiring more resources when there is an increase in demand and releases some of the resources if there is a decrease in demand. Thus it also helps in saving the overall operational cost of the hired resources.
- The monitoring services enable CPs to monitor the utilisation of cloud resources dynamically. This helps in the efficient management of the cloud resources, e.g. if the utilisation of computational nodes is low, the CPs shifts the load to a few computational nodes and switches off unnecessary nodes to reduce the energy consumption. Alternatively, the CPs can maximise their revenue by allocating resources to the customers based on their utilisation and renting out the free resources to more customers. Such a phenomenon is referred to oversubscribing. Similarly, the SPs can also gather monitoring reports and manage their rented resources as per their utilisation.

2.1.3 *Technical View*

From a technical point of view the cloud computing architecture can be viewed as a layered model, consist of three layers including Infrastructure, Platform and Application (also refers as Service). The main idea of such a model is shown in Figure 2.1 (borrowed from [1]). Each of these layers is responsible to provide a service and therefore, they are commonly known as Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). The short description of each of these services is as following:

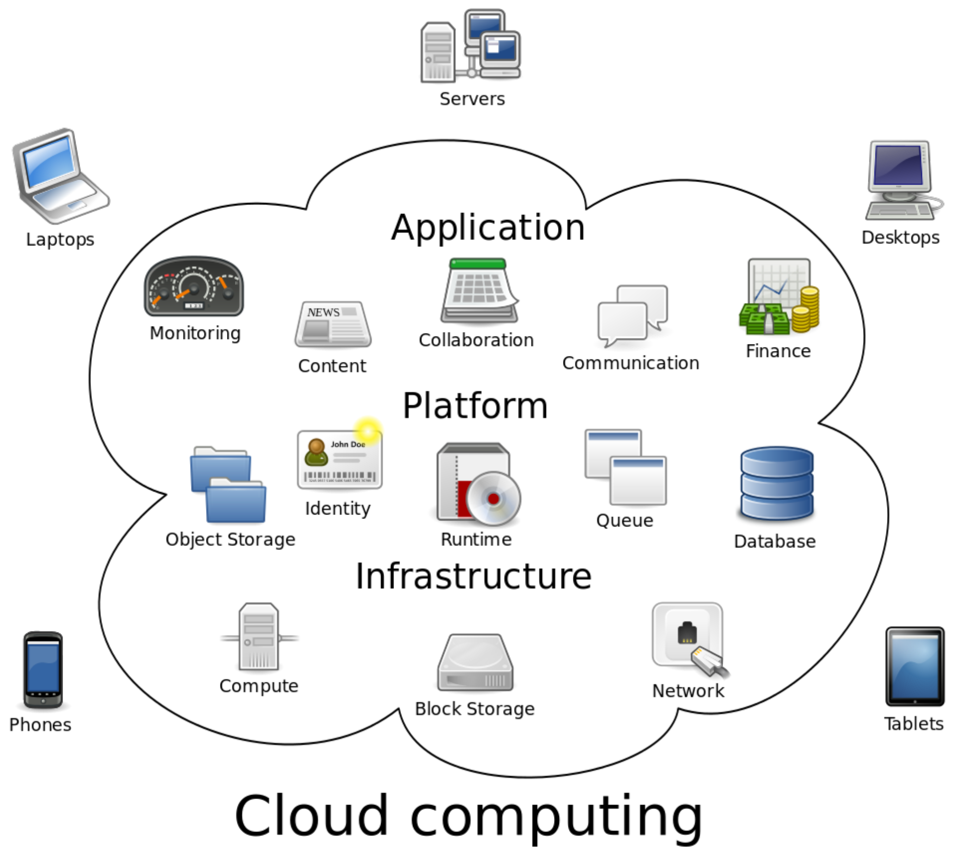


Figure 2.1: Cloud computing model [1]

1. IaaS refers to the service that provides infrastructural resources like storage, network and computational. Once these resources are made available, they are further used by the cloud consumers for the deployment of their own services or utilise them for any storage or computational purposes.

2. PaaS refers to the services that provide computational tools to the cloud consumers for the development and deployment of their applications (or services). Examples of these tools include the programming languages, libraries, database services and various other deployment tools.

3. SaaS (also known as Application as a Service) refers to the services delivered to the end user in the form of software (or application) that execute over a cloud infrastructure. The application itself is accessed using either a web-based or a programming interface.

The description of service models mentioned above hints three different kinds of cloud users. Figure 2.2 (borrowed from [2]) depicts their relationship from a web application perspective. The CPs provide all the infrastructure related services and the necessary software/services/utilities needed by the SPs and the end users. The SaaS provider use the services provided by the CPs to deploy and host their software/applications/services. The SaaS users are the end users that consume the services and applications of SaaS providers (also called SPs as in this thesis).

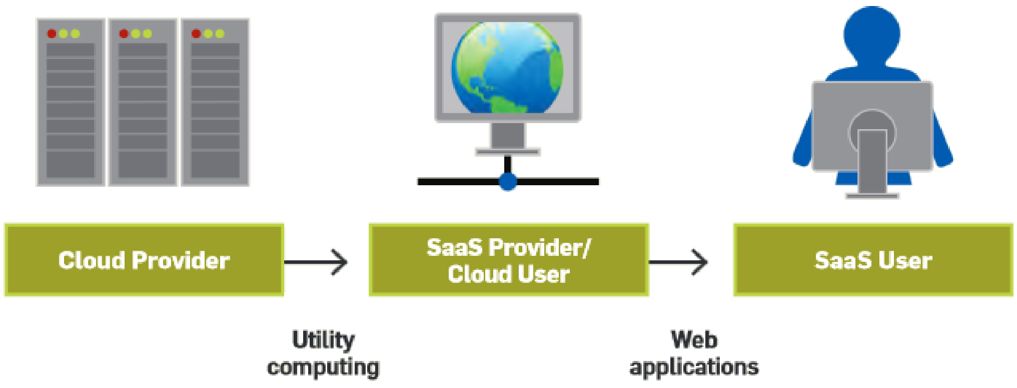


Figure 2.2: Various cloud users [2]

Another essential aspect of cloud computing is known as the deployment model, which refers to the type of cloud environment. According to NIST, there are the following four types of cloud deployment models available:

1. Public cloud: Such a cloud infrastructure owns and manages by a business organization, e.g. AWS [67] by Amazon. Such a cloud deployment model provides infrastructure services be used by the public.

2. Private cloud: A private cloud infrastructure is for the exclusive use of a single private organisation, e.g. business or institute specific. The management of such a cloud environment may handles by the private organisation itself or by a third party and it may exist in the premises of the private organisation or outside of its vicinity. Common examples of such a cloud model include SAVI [68] and the Ubuntu enterprise cloud [69], etc.

3. Community cloud: A community cloud infrastructure is for the exclusive use of a specific community that comprises of multiple organisations sharing some common or mutual interests, e.g. Government cloud [70].

4. Hybrid cloud: A hybrid cloud infrastructure is the composition of two or more types of clouds mentioned above. In this model, each individual cloud entity works independently, however, combinely they are using a standardized technology to enable collaboration at some specific situations or as when needed, e.g. netApp [71], etc.

2.2 CLOUD ELASTICITY

2.2.1 *Overview*

Cloud elasticity refers to the ability of a system to dynamically reconfigure the computational resources as a result of a change in demand. It is one of the most appealing feature of the cloud computing, and therefore, some refer to it with the combination of on-demand provisioning as a game-changing attribute for IT [72]. Herbst et al. in [25] defines cloud elasticity as *"the degree to which a system is able to adapt to workload changes by provisioning and de-provisioning resources in an autonomic manner, such that at each point in time the available resources match the current demand as closely as possible"*. This definition refers to the following two characteristics of elasticity: (1) Scalability - the ability to scale the infrastructure when there is a change in demand, and (2) Flexibility - performing the scaling action at runtime as and when needed.

The key objective of implementing cloud elasticity is to improve utilisation of computational resources whilst maintaining desired performance of the system and reducing its operational cost. This objective can be viewed differently by CPs and SPs. The CPs perspective of better resource utilisation is to improve performance of the system and reduce the operational cost of data centre, e.g. decrease in electricity consumption, increase in revenue generations using over-subscription, and reduction in the CO_2 emissions. From the SPs perspective, it is to reduce the operational cost of the services consumed whilst simultaneously maintaining performance and reliability of their deployed services. These different points of views promote the design of resource management methods for various purposes such as energy-aware resource allocation [8] and cost-efficient proactive scaling [34], etc. This thesis, however, focuses on the SPs perspective of cloud elasticity; therefore, we are proposing an application specific methodology. Thus, we are focusing on factors like performance and operational cost concerning evaluation and not considering other factors like the consumption of energy or revenue generation, etc. However, the methods proposed are generic and can be equally adopted by the CPs to provide elasticity services to their customers by configuring for each application.

2.2.2 *Technical View*

Figure 2.3 presents a functional block diagram of an autoscaling system. This system is equipped with an elastic policy, which is responsible for the required resource management. The policy, in general, is referred to "any type of formal behavioural guide" [73] provided as an input to the system. The elastic policy, in this case, determines when and how an elastic scaling action must be performed [4].

The elastic action is of two types, i.e. *Scale-up* and *Scale-down*. The *Scale-up* is an increase in the quantity of current computational resources by some quantity computed by the elastic policy. In contrast, the *Scale-down* refers to the decrease in current computational resources by some quantity. The *Scale-up* action is performed to improve the performance of the system in response to the increase in demands. Whereas, a *Scale-down* is carried out to reduce the operating cost when there is a decline in demand of the resources. The nature of an elastic action depends on the type of elasticity, i.e. *Horizontal* or *Vertical*. In the case of

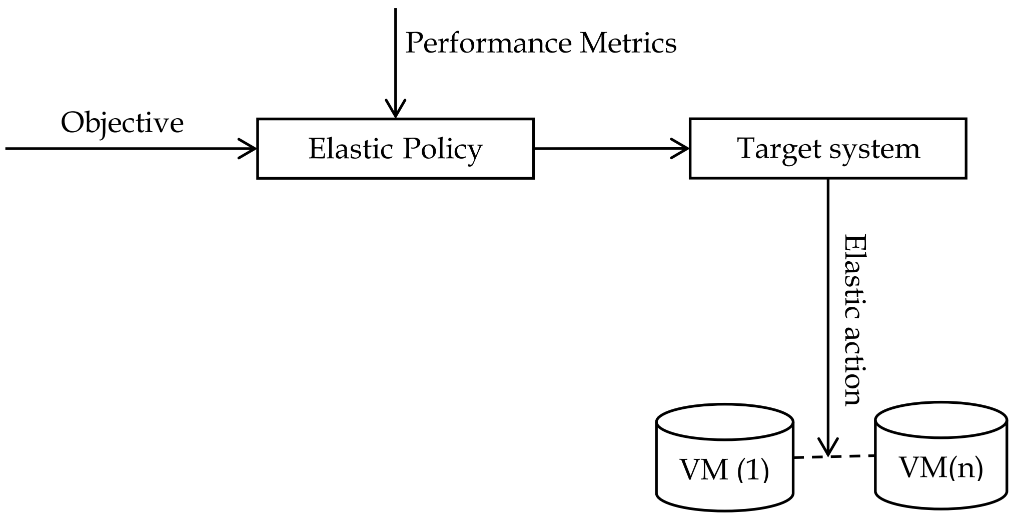


Figure 2.3: Block diagram of an autoscaling system

Horizontal elasticity, an elastic action increases or decreases the number of VMs. Whereas, in the case of a *Vertical elasticity*, the change occurs in the specification of existing VMs, e.g. the increase or decrease in CPU and/or memory capacity of a particular VM or the set of VMs.

The *Objective* in Figure 2.3, in general, refers to the input criteria that specify the goals of the system. The elastic policy makes use of these inputs and performs elastic decisions to satisfy or maintain the desired objective. The nature of such input criteria depends on the method of implementation. For example, such inputs specify the up/down thresholds for various metrics of the system as in the case of commonly used Rule-based methods (see Section 2.4.1 for details). Alternatively for feedback controllers (see Section 2.4.2 for details), the *Objective* is represented in the form of desired reference input.

The *Performance metrics* refers to the set of various system or application level parameters that inform the most recent status regarding the various aspects of the system. Common examples of such metrics at the system level include CPU utilisation and memory usage, whereas the application level include Response time and Throughput. Such metrics help in designing the elastic policies as the elastic actions are dependent on their values. The monitoring services provided by the CPs (see Section 2.1.2) are responsible to record the status of system level metrics continuously. The application itself is incharge of keeping track of the required application level metrics. An elastic policy can access the necessary metrics either by using the CPs monitoring service via Application Programming Interface (API) or by implementing a customized monitoring component of its own. A detailed list of different metrics used in implementing various elastic methods are shown in the tables provided in [4, 74].

The elastic policy reaches to a scaling decision by taking into account the objective and the latest measurement of the performance metrics. The target system subsequently takes the scaling decision and performs the actual operation using the CPs provided infrastructure management API. The triggering of scaling decisions itself is of three types:

1. Reactive: Such an approach performs the scaling decision in response to changes in the behaviour of the system. Such changes can be identified through a change in demand or measurement of the performance metrics.
2. Predictive: This approach anticipates future behaviour of the system and performs the scaling decision in advance. The prediction itself can be performed using forecasting the future workload behaviour and/or future utilisation level.
3. Hybrid: Such an approach combines both the *Reactive* and *Predictive* methods.

The Reactive and Prediction based approaches (i.e. both *Predictive* and *Hybrid*) have their own shortcomings. The *Reactive* approaches are criticised for the delay elapsed between the time of reaction to a change and the actual completion time of the reconfiguration process. Whereas, the performance of prediction based approaches (i.e. both *Predictive* and *Hybrid*) rely on the accuracy of predictions. They may fail or perform poorly in situations where the predictions are either not possible or not accurate, e.g. when there are abrupt changes in the workload [75]. Prediction based approaches are also criticised for the associated additional computational cost caused due to the on-line learning and analysis required for predictions [45]. This thesis, however, follows a *Reactive* approach as it avoids the complexity of incorporating a prediction method and it does not involve additional computational overhead. Moreover, the focus of this research is to demonstrate the effectiveness of biologically-inspired methods for the cloud resource provisioning problem rather than to introduce a new prediction based approach.

2.2.3 *Key Issues in Cloud Elasticity*

The key aim of an elastic method is to maintain the desired application performance at the lowest possible cost. The implementation of an auto-scaling method, irrespective of its underlying implementation technique, must consider avoiding the following commonly known generic issues of elasticity domain. More specific issues and challenges in the context of this research are listed in Section 3.4.

- Over-provisioning: The over-provisioning issue is referred to the scenarios, when more than required computational resources are running at a time. The key reason to over-provision the resources is to avoid performance violation considering peak workload scenarios [76, 77]. Such an approach provides better performance, however, at a higher price than necessary. A certain degree of over-provisioning may be unavoidable while considering the unpredictable nature of workload fluctuation at a smaller level [45]. However, in general, over-provisioning at a large scale should be avoided.
- Under-provisioning: The under-provisioning problem is referred to the scenario when the available resources are not sufficient enough to fulfil the requirements of the incoming workload. Such a situation produces bad performance and possibly results in the violations of Service Level Agreement (SLA) (explained in Section 2.3.2). Under-provisioning can sometime be

catastrophic as poor performance may results in the loss of customers and ultimately lead to the financial losses [21, 20]. Moreover, in some situations, the violation of SLA can cost financial penalties [78].

- Oscillation: The problem of oscillation refers to the scenarios of auto-scaling, where both the unwanted scenarios mentioned above (over and under-provisioning) occurs more frequently. Such a behaviour is usually caused when the scaling actions are performed either too quickly or too aggressively or even when the impact of the reconfiguration is not considered in the next scaling action [45].

2.3 TARGET SYSTEM

A *Target system* in Figure 2.3 refers to a system, which is equipped with auto-scaling behaviour. This thesis focuses on the Internet-based systems and exploits web applications as an example of the target system because of the following reasons: (1) The growing popularity of using cloud computing for hosting web-based applications and (2) A large quantity of existing resource provisioning methods are introduced and evaluated for such systems [79]. This section provides a brief overview of web applications and how it works in an elastic environment of the cloud. Moreover, it introduces related concepts including QoS, SLA, Service Level Objective (SLO) and Workload.

2.3.1 *Web Applications Overview*

The design and deployment of a web application follows a tiered architecture, where different parts of the application execute at different levels. A three tiers architecture is commonly adopted. Figure 2.4 shows the block diagram of such an architecture, which consists of three tiers known as the *Client*, *Business* and *Database*. The end-user at the *Client* tier accesses a web application through an Internet browser by generating an HTTP request. The *Business* tier at the server

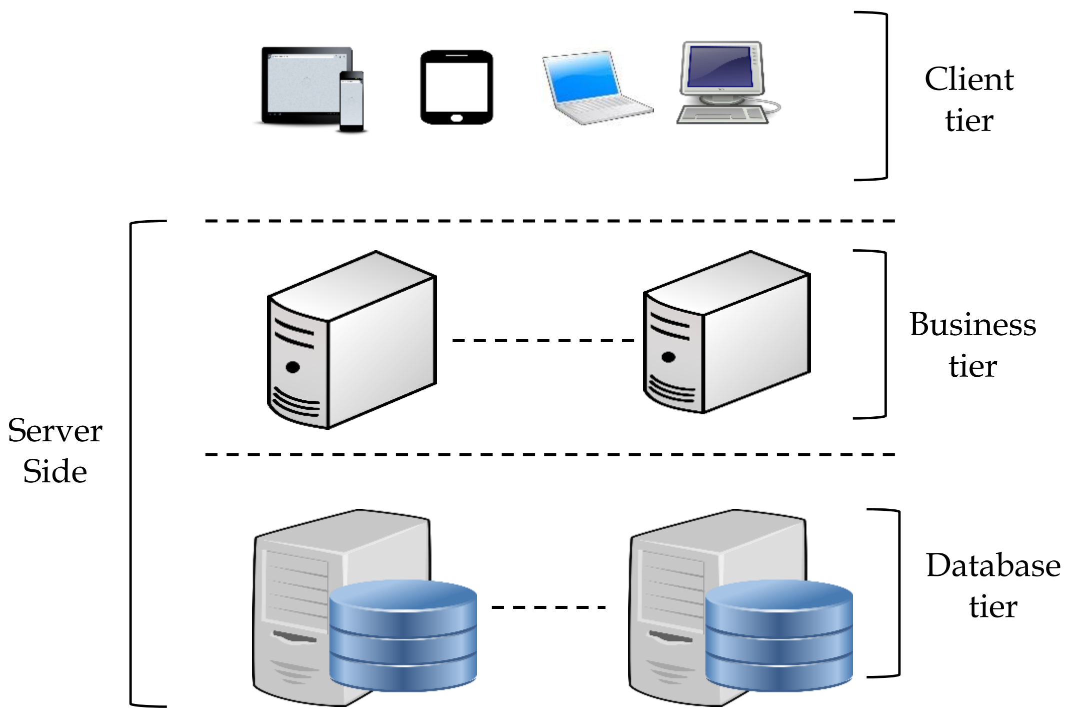


Figure 2.4: General architecture of web applications

side is responsible for the response to the end user requests by executing any necessary action required at the server side. The *Business* tier itself is often extended to multiple tiers depending on the complexity of the underlying application. The usual practice in such scenarios is to separate them between a *Web server* and *Application* tiers. A *Web server* tier is responsible to handle the incoming end-user HTTP requests, whereas the *Application* tier is used to handle the complex queries and database interactions. The *Database* tier at the server side is responsible for the data storage related functionalities.

Figure 2.5 (borrowed from [3]) shows the block diagram of the hosting of a web application in the AWS cloud environment. The *Web* and *Application* Servers in the figure under label 5 are the Elastic Compute Cloud (EC2) VMs instances, which hosts the *Web* and *Application* tiers respectively. The *Database* Servers on the other hand hosts the corresponding *Database* tier. Label 6 in the figure represents the *Auto-scaling* group, which governs and manages the *Web* and *Application* Servers. The *Auto-scaling* group implies that the underlying servers will automatically scale up and down based on the applied elasticity rules. Alternatively, auto-scaling can also be performed at individual tier level. Label 4 in the figure represents an *Elastic Load balancing* component that receives the incoming HTTP requests. It further distributes them amongst the running EC2 instances. The above description explains the working mechanism of how a web application works in cloud environment in integration with the auto-scaler and load balancer. The auto-scaling methodologies proposed in this thesis work similarly as the *Auto-scaling* component in Figure 2.5.

2.3.2 *Quality of Service and Service Level Agreement*

The QoS refers to the measurement that describes the performance or quality of an offered service. More specifically, it is used to describe the quality of network related services. The various aspects of a service considered for measurement depend on the nature of the service itself. In the context of this thesis, the commonly considered aspects to measure the quality of an Internet-based system deployed over cloud includes *Availability*, *Throughput*, and *Response time*, etc. *Availability* refers to the quantitative measure that describes the availability (up-time) of a service. *Throughput* determines the total number of completed requests in a

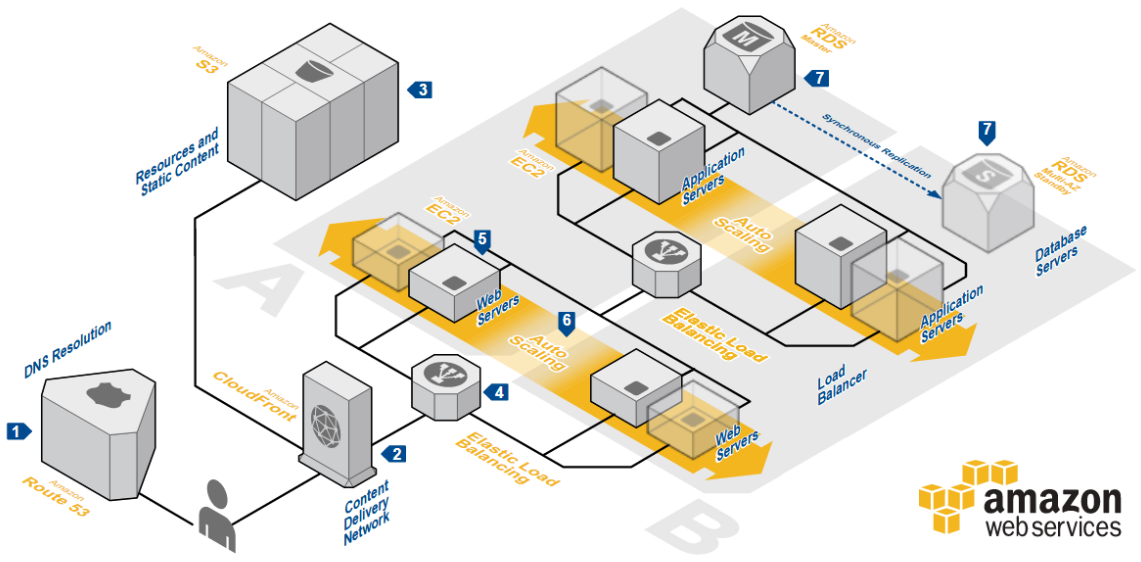


Figure 2.5: Architectural view of web application hosting at Amazon cloud [3]

specific time unit, whereas *Response time* defines the elapsed time between the arrival of the end-user request till completion.

In the cloud context, the QoS requirements of a service are recorded in a contract known as SLA [80]. The formal definition of an SLA is *"a document that includes a description of the agreed service, service level parameters, guarantees, and actions and remedies for all cases of violations"* [81]. SLA is an agreement between the provider and consumers of the service, and it contains all the details related to the agreed characteristics of the provided service regarding QoS, cost and responsibilities of both parties. The QoS related details are specified in the form of individual measurable units of SLA. The measurable unit is formally known as Service Level Objective (SLO), which is the combination of three distinct parts, i.e. a performance metric, numeric quantity and a relational operator. For example, a commonly used SLO for an Internet application is *"The Response time of 99% HTTP requests must be less than 1 second"*.

2.3.3 *Workload*

The term workload is often referred to any form of inputs provided to the e-infrastructure or benchmark application for processing purposes [82]. An e-infrastructure is referred to any digital based hardware or software facility. In the cloud computing context, the workload refers to the interactions/transactions made by end-users to the cloud-based services. Each interaction represents a job to be processed by the cloud service. The workloads in the cloud can be classified into two types: transactional and batch [4, 45]. A transactional workload is the most well-known type that consists of job requests. These requests include browsing or update queries to access or modify content like HTML pages, pictures, streams of videos and transactional data. These requests will return or execute contents that are either statically stored or dynamically rendered by the server applications. In contrast, a batch workload consists of time-consuming and resource intensive jobs submitted in a group or batch. The batch workloads are commonly used by the scientific domain or applications like large-scale video processing [45].

As mentioned earlier, this thesis focuses on web applications as a target system. Therefore, we only use the transactional workloads as test case scenarios for the evaluation purposes. The job arrivals in web related workloads follow two different models, i.e. an *Open* or *Closed* models [83]. The workloads based on *Open* models are those where every job request is independent of each other. In contrast, in the case of *Closed* models, new arrivals of jobs are dependent on the successful completion of the previous job in addition to the extra think time between the consecutive jobs. In this thesis, we only consider workloads based on *Open* model.

A workload is characterised mainly by the *Arrival rate*, which refers to the number of arrived jobs in a particular time period. The jobs of a workload arrive at different points in time and they vary from each other regarding *Service time* demands. A *Service time* specifies the amount of CPU time required for a job to complete its execution. The different workloads utilised in this thesis for evaluation purposes consider both these attributes. More importantly, we use the *Arrival rate* as one factor in the decision-making process. In some research works [84], workloads are characterised based on various patterns and then different policies are used for various kinds of workloads. Such a characterisation of workload is beyond the scope of this thesis. However, this thesis makes use of diverse real workloads' examples that follow different workload patterns for the evaluation purposes (see Section 5.2 for details).

2.4 RELATED IMPLEMENTATION METHODS OF ELASTICITY

The existing cloud elasticity proposals are implemented using versatile techniques ranging from rule-based systems to complex optimisation and machine learning based methods. This section, however, provides an overview of the following three methods due to their relevance to this thesis.

2.4.1 *Rule-based Auto-scaling*

The *Rule-based* auto-scaling method facilitates the collection of elasticity rules that follows an *"if Condition then Action"* based pattern. The *Condition* in the rule specifies the criteria of interest and is designed using a system provided metrics based on the QoS requirements. A detailed list of the various metrics is shown in the tables provided in [4, 74]. The *Action* to perform as a result of evaluation of a rule specifies the decision. The action may be the acquisition or release of VMs in the case of *Horizontal* elasticity or the increase/decrease of VM level specification (memory size, for example) in the case of *Vertical* elasticity. The increase or decrease in the quantity of resources in a scaling action part of the rule is a constant number or percentage based on the existing resources. The process of setting up the scaling rules does not involve any systematic method, and is mostly based on the empirical judgements of SPs [79]. An example of such a rule is given below [45]:

> if $x_1 > thrU_1$ and/or $x_2 > thrU2$ and/or ...
> for durL seconds then
> n= n + s and
> do nothing for inL seconds

The rule mentioned above specifies a *Scale-up* action. In the case of a *Scale-down* action, "$n = n + s$" is replaced by "$n = n - s$". The condition part may consist of one or multiple metrics of interests, e.g. x_1 and x_2 in this case. The parameters $thrU1$ and $thrU2$ indicate their target threshold levels respectively. These threshold values represent the desired objective. The $durL$ parameter represents the time duration on which the condition is based. The term s represents the number of additional VMs in the case of *Horizontal elasticity*, whereas it represents the increase in VM capacity in the case of a *Vertical elasticity*. A *Cooldown* period of certain time, e.g. inL in this case, is usually applied, where the auto-scaling mechanism does not take further scaling action. The key reason for restricting this period is to avoid oscillation. Based on the above format of the rule, a real

Horizontal elasticity rule becomes like the following: *"If Average CPU utilisation* $>$ 60% *for 5 minutes then add 2* VMs *and do nothing for 5 minutes"*.

Figure 2.6 (borrowed from [4]) presents the architecture of a *Rule-based* auto-scaler. The *Rules* encode the QoS targets (or expectations) in the format described earlier, whereas the *Rule Engine* is responsible for the execution of the rules that determine the scaling actions. The *Monitoring* component of the auto-scaler provides the up-to-date values for the metrics used in conditions, e.g. x_1 and x_2 in the example rule mentioned above. The elastic application carries out the scaling actions using the management API provided by CPs.

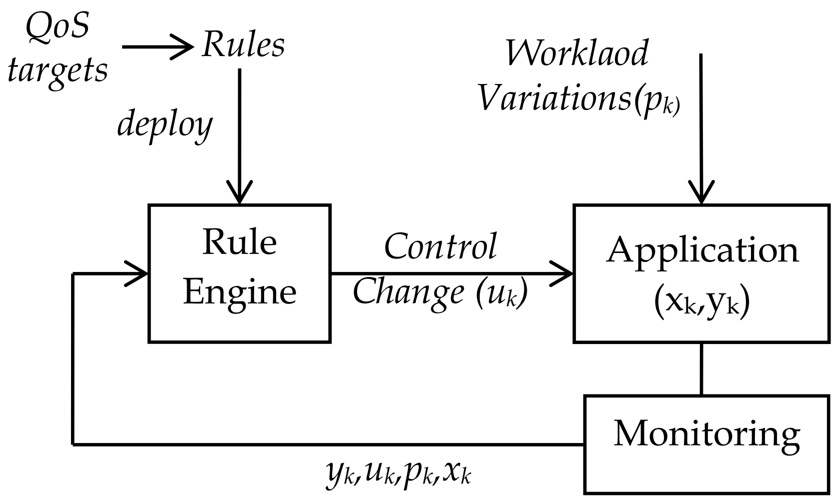


Figure 2.6: Architectural view of *Rule-based* auto-scaler [4]

The *Rule-based* auto-scaling techniques are more popular and used by commercial CPs such as Amazon [26]. Moreover, there are also some third-party solutions available that provide *Rule-based* auto-scaling facilities, e.g. Rightscale [27]. The key reasons behind the popularity of such an approach are the commercial availability and their simplistic nature for designing the elasticity rules. This thesis uses the Rightscale [27] method as one benchmark methodology (See Section 5.3.2 for further details).

2.4.2 *Elastic Feedback Controllers*

A *Control system* is responsible to manage and regulate the behaviour of a system using control loops. In the context of cloud elasticity, the main objective of a control system is to automate the resource management task of a target system. This is achieved by maintaining the value of system output, e.g. *Response time*, close to a desired value by changing the value of system input, e.g. number of VMs. There are the following three main types of control systems [45].

1. Open Loop: The *Open Loop* system computes the value of system (or control) input using only the system model and current state of the system without considering the system output as a feedback signal. Hence, such systems are independent of the system output.

2. Feedback: The *Feedback* control system, in contrast to the *Open Loop* system observes and measures the system output for taking into consideration the deviation of the system output from the desired value.

3. Feed-forward: The *Feed-forward* control system predict the behaviour of the system using a model to anticipate control error and correct any deviation in advance before its occurrence.

From this point onward, we focus on *Feedback* control systems due to (1) their frequent use in cloud elasticity literature and (2), the methodologies proposed in this thesis are based on feedback loop. This section briefly explains the basics of feedback loop in the context of cloud elasticity, whereas the classification of various control strategies such as *Classic*, *Optimal*, *Advanced* and *Intelligent* are further provided in Section 3.2.

The use of feedback control approaches were well received even before the invention of the cloud computing and various systems were using it for self-adaptivity purposes such as web servers [85, 86], database servers [87] and cache storage systems [88]. The key idea of the feedback loop model is to use the measurement of a system output and adjust the control inputs to achieve a particular goal. For example, the cruise control system of a vehicle achieves a target speed by readjusting the accelerator based on the speedometer's measurement. Similarly, an elastic feedback controller maintains the output of a system, e.g. Response time, to a desired value by adjusting the control input, e.g. number of servers. Such a control system can be used to satisfy a constraint or guarantee an invariant on the outputs of the system [4]. More specifically, the controllers are particularly designed for a specific purpose commonly referred to as a control objective, e.g. to maintain an overall average Response time of less than t seconds. The commonly used control objectives are one of the following types:

- Regulatory control: A feedback controller developed for regulatory purposes maintains system output close to the desired reference value. For example, the average CPU utilisation of the *Cluster* must be 60%.

- Optimisation: Such a controller solves an optimisation problem to ensure the optimum value of the system output in the presence of certain constraints. For example, minimisation of system's response time with the lowest possible cost.

- Disturbance rejection: Such a controller is used to manage and adjust the level of disturbances, e.g. *Admission control system*. It only allows enough workload that does not affect the performance of the system.

Figure 2.7 (borrowed from [5]) depicts the architecture of a feedback loop model, where a controller observes the system output to correct any deviation from the desired value. This figure shows, the design of a feedback control system consist of various essential elements. A short description of these elements is as follow:

- Reference input: It refers to the desired value of the system output that the controller is required to maintain. For example, the overall average CPU utilisation of all acquired VMs (*Cluster*) must be 60%.

- Control error: This refers to the difference between desired *Reference input* and the measured value of a system's output.

- Control input: It refers to the dynamic parameter computed by the controller that affects the behaviour of the target system to achieve the *Reference input*. For example, the number of VMs used by the *Target system*.
- Actuator: This component executes the decision made by the controller.
- Target system: This refers to the underlying system managed by the controller. For example, the corresponding *Elastic application* in this case.
- Sensor: A sensor measures the values of metrics needed by the controller for making the next scaling decision. For example, to measure the CPU utilisation of VMs.
- Controller: A controller is the mechanism that computes the values for the control input required to achieve the desired objective value (*Reference input*) by taking into account various measurements, e.g. the *Control error* mentioned in the figure.

The *Disturbance* in Figure 2.7 refers to the workload, whereas the *Measured output* is the latest measurement of the system output. The design of a control system

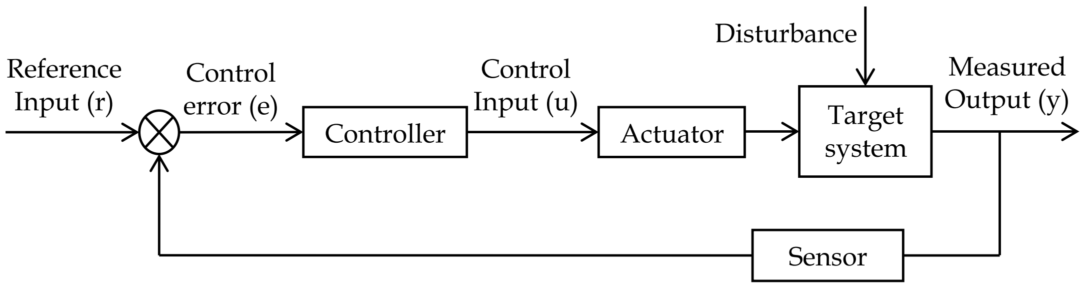


Figure 2.7: Block diagram of a feedback control system [5]

is composed of the following two steps. Firstly, the formal construction of a system model that determines the relationship between inputs and outputs of the system. Secondly, the design of the controller using the obtained model. The implementation of a controller varies in various aspects consisting of modelling approach, controller type and the architectural complexity. A detailed state-of-the-art review is undertaken in Chapter 3 which provide a comprehensive survey of the research regarding the existing feedback controllers. This review highlights and analyses the various aspects of control-theoretical approaches used to implement auto-scaling.

2.4.3 *Fuzzy Control System*

The fuzzy control system can be viewed as an advance form of a *Rule-based* system, where a collection of rules are implemented using the incorporation of human knowledge. Such a system is based on fuzzy inferencing rather than a classical decision mechanism, where a rule executes to a true or false value. More specifically, a fuzzy control system refers to a control methodology based on the use of *Fuzzy logic* theory. The *Fuzzy Logic* initially introduced by Zadeh [89] is a computing approach based on the notion of degree of truth rather than an exact truth, i.e. true or false. It is one kind of many-valued logic that relies on approximate reasoning rather than fixed reasoning like *Boolean logic*. In contrast to the *Boolean logic*, a variable in *Fuzzy logic* can take a value in the range from 0 to 1 rather than a true or false. This value determines the degree of membership in the fuzzy set.

The use of *Fuzzy logic* theory is employed for those control and decision-making problems, where system knowledge is imprecise, uncertain and highly dynamic [51]. Moreover, it is used in situations, where a mathematical illustration of the system may not be viable. Another important aspect of the use of *Fuzzy logic* theory is that it enables the qualitative decision-making process by designing the rule based system. The rules are constructed from meaningful words (also called labels) and therefore they are easily understood by humans.

The construction of a fuzzy control system consists of the following three ingredients: (1) The *Domain knowledge* includes the identification of system related knowledge, i.e. inputs and outputs. The inputs and outputs are represented by a fuzzy set using different linguistic terms; (2) The *Membership functions* defines the degree of crisp input against its linguistic variables in the range [0 to 1]; (3) Rules can be created using a combination of linguistic terms of fuzzy sets. Once these ingredients are ready, a fuzzy control system works according to the block diagram provided in Figure 2.8. As can be seen from this figure, the *Fuzzification* accepts the *System's input* in the crisp form and converts it to the corresponding Fuzzy values using the *Input membership functions*. The Fuzzy Inferencing is responsible for executing the fuzzy rules and generating the fuzzy output values, which are then converted to *System's output* values in the crisp form using the *Defuzzification* process.

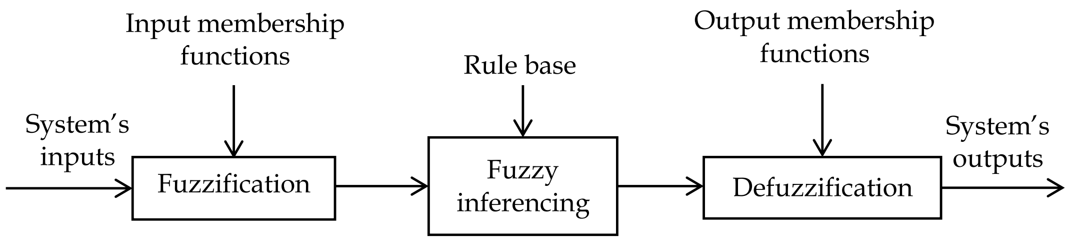


Figure 2.8: Architectural view of a fuzzy system

The use of a fuzzy system as an auto-scaler follow the structure provided in Figure 2.8, or it can also be adapted as a feedback loop model. In such a case, the *Controller* part of Figure 2.7 is replaced with a fuzzy system. The block diagram of such a structure is provided in Figure 2.9. The various fuzzy based elasticity proposals are reviewed and presented in Section 3.3.4. This thesis uses a fuzzy control system for the implementation of the switching mechanism of the proposed method (see Section 4.5 for details).

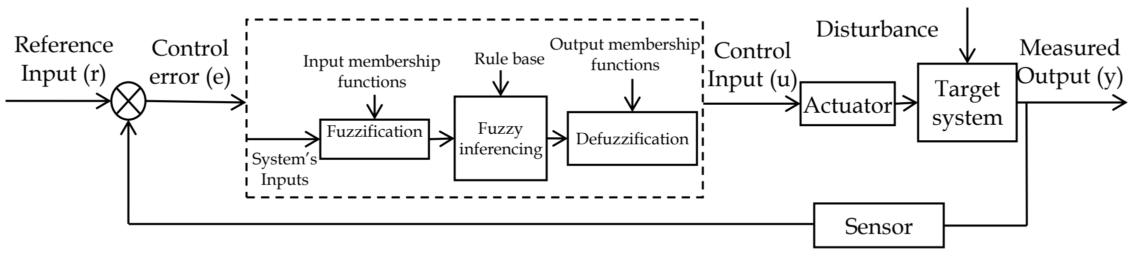


Figure 2.9: Architectural view of a fuzzy system as a feedback loop model

Part II

CONTRIBUTIONS

3 CONTROL THEORETICAL VIEW OF ELASTICITY: TAXONOMY, SURVEY AND CHALLENGES

This chapter discusses the state-of-the-art control theoretical view of cloud elasticity, summarising initially the existing survey-based research works in Section 3.1. This establishes the need for reviewing the literature from a new perspective required for the understanding of control theoretical view of cloud elasticity. The chapter then introduces a novel taxonomy in Section 3.2, which includes aspects from implementation technique perspective (i.e. control theory) as well as from application domain perspective (i.e. cloud elasticity). An exhaustive review, following the proposed taxonomy, of the existing control theoretical approaches proposed for elasticity problem is carried out in Section 3.3. Lastly, this chapter summarises and discusses some open issues and challenges in Section 3.4 to lay the foundation for the focus of this thesis.

3.1 OVERVIEW

With the rise in popularity of Internet-based applications, the notion of providing better elasticity management has considerably increased. This has a proportional effect on cloud elasticity literature, where researchers and practitioners make use of various techniques to exploit cloud elasticity. Many research undertakings are carried out on cloud elasticity, and its different aspects are explored. Moreover, there are also different survey papers available that provide a concise review on elasticity research. However, the scope of all such papers are broad. They have focused on the high-level view of overall elasticity research rather than specific details on one implementation technique to better understand the differences amongst related approaches. Hence, this thesis focuses on the control theoretical approach of cloud elasticity, therefore a standalone review of control theoretical methodologies are provided.

The review here is carried out with a novel taxonomy and classification. Therefore, this section briefly describes the current survey papers as related work and provides a brief explanation of how the review conducted in this chapter is different. For this purpose, we classified the relevant review papers in the following three categories based on their primary strengths.

1. Cloud resource management: The review articles in this category mainly cover an extensive range of cloud resource management related problems such as provisioning, allocation, scheduling, mapping, adaptation, discovery and brokering. Amongst these problems, cloud elasticity (or dynamic cloud resource provisioning) approaches are covered either partially or in a limited capacity. Singh and Chana [42] focused on autonomic computing with a particular emphasis on QoS-aware management of resources; Jennings and Stadler [90] used resource management functions as a classification method; Mustafa et al. [91] reviewed the literature based on the metrics used and discussed the underlying research problems. Manvi and Shyam [92] classified the literature into problem specific categories such as resource provisioning, and allocation. Whereas Singh and Chana [93] targeted resource provisioning in general, wherein elasticity is considered as a trait of resource provisioning mechanism.

2. Adaptability using control theory: The review papers in this category are related as their primary focus is on the use of control theory in similar context, e.g. QoS management or adaptation in general. However, none of them have considered the elasticity proposals. Yfoulis and Gounaris [94] briefly investigated the control theoretical perspective in cloud computing context with a focus on SLA management. More relevant but brief discussions on the use and suitability of feedback controllers in larger cloud computing domain for performance management is carried out in [95]. Whereas, Gambi et al. [44] focused on the assurance and adaptability perspective of cloud

controllers. However, their scope is wider and also includes other techniques such as Rule-based and Machine learning. Patikirikorala et al.[96] carried out a systematic survey of the design of self-adaptive systems using control solutions. They presented a quantitative review based on a taxonomy consisting of attributes such as target system, control system and validation mechanism. The scope of their research, however, is much wider, i.e. general adaptive systems rather than cloud elasticity. Moreover, they only presented a quantitative analysis of the existing research works rather than a detailed review.

3. Cloud elasticity: A comprehensive survey on cloud elasticity is carried out in [45], where the authors classified the elasticity literature based on the underlying implementation techniques. However, they only focused on cloud application provider based approaches. Whereas, PaaS systems are focused in [97, 98]. Galante and DeBona [99] classified the existing literature based on infrastructure and application level. A taxonomy consisting of features like scope, purpose, decision-making mechanism, action type and evaluation is proposed in [75]. A similar taxonomy is also adopted in [100, 101]. The authors of [78] focused on strategy, action type and architecture perspective. Whereas, an adaptability view of computational resources with a larger scope including concepts like node adaptation and VM migration is provided in [102]. They, however, used adaptation techniques as one of the dimensions to review the literature. Elasticity functions such as reactive migration, resizing and proactive replication are used as a means of classification in [74]. At last, Galante and da Rosa Righi [103] reviewed cloud elasticity in the scientific application perspective.

All the survey papers mentioned above are very innovative and mostly overlapped regarding essential elasticity features, e.g. type (*Reactive/Proactive*), action (*Horizontal/Vertical*), scope (CPs/SPs), etc. However, their scope is wider, i.e. overall elasticity research and apart from [45], they lack details on the underlying implementation techniques. Moreover, the exhaustive review of the proposals of each implementation technique is missing. This chapter is a step forwards toward a technique specific (control theoretic in this case), up-to-date and exhaustive review of cloud elasticity solutions.

3.2 PROPOSED TAXONOMY

Figure 3.1 presents the proposed taxonomy, which consists of characteristics from both control theoretical as well as cloud elasticity perspective. The brief explanation of some of the attributes of this taxonomy is covered already in Section 2.4.2, whereas the description of the remaining attributes is provided below.

3.2.1 *Control Solution View*

3.2.1.1 *Controller*

As mentioned earlier in 2.4.2, the systematic design of a feedback control solution consists of the following two steps. Firstly, the formal construction of a system model. Secondly, the implementation of a control mechanism. Based on this description, the *Controller* is divided into the following two subcategories.

1. Model: Generally, the various modelling approaches used in the design of control systems are categorized into the following three main classes [104, 105, 106]:
    - White-box: Such models are used when it is possible to construct the model based on the prior knowledge and the availability of the physical insight about the system. White-box modelling derive mathematical models based on the use of first principles.
    - Black-box: Such models are data driven and no physical insights or prior knowledge of the system is required. Statistical methods are used to derive the model based on the measurement of data using well

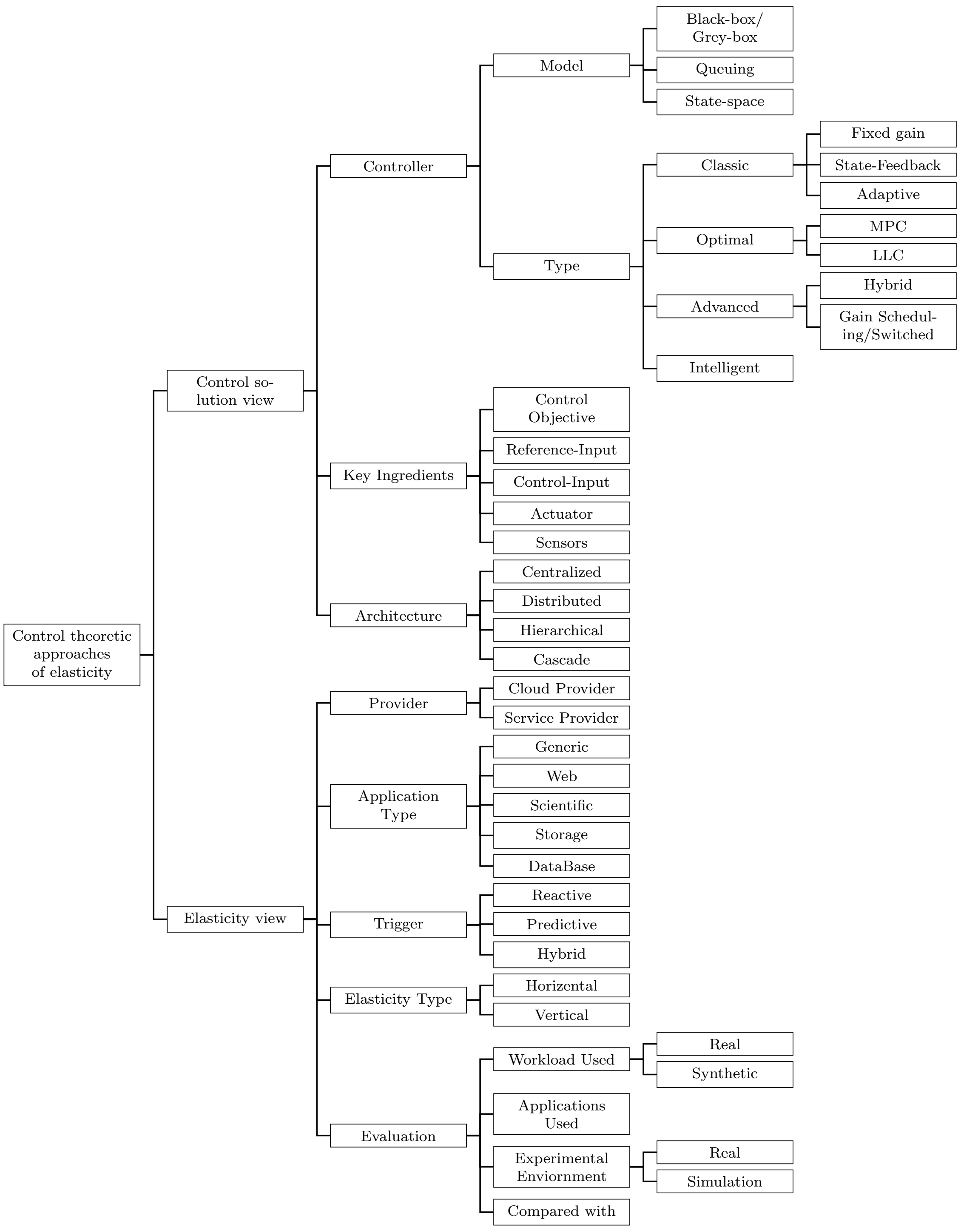


Figure 3.1: Proposed taxonomy for control theoretical approaches of auto-scaling

designed experiments, where the underlying system is considered as black-box.

- Grey-box: Such models are hybrid in nature and are used in situations where some physical insights or prior knowledge about system is available, however, certain parameters are required to be derived from observed measurements.

The use of white-box modelling approaches in the context of cloud elasticity are rare. However, in certain few cases *Queuing Theory* and *State-space* modelling approaches are utilized. Thus, in the context of cloud elasticity, we found the following types of modelling techniques are used in the construction of control systems to address dynamic resource provisioning problem:

- Queuing Theory: The elastic system is considered as a queue so that different analysis can be performed such as prediction of queue length, average service rate, and average waiting time.
- Black-box/Grey-box: As earlier described, such methods are used when the detailed knowledge of the target system is not available. Such approaches involve the construction of SID experiments, where well-designed system input signal is generated to record outputs of the system. Statistical techniques are then utilised to infer system input and output relationship.
- State-space: The target system using such an approach is characterised and represented using a set of state variables to express their dynamics. For detail description of these modelling techniques, please refer to [53].

2. Type: There are various types of controller used for the implementation of elasticity. We have clustered them into the following four groups inspired by the controller types used in [96]:

- Classic: This category contains the most commonly used controller types that are comparatively simpler in nature. This includes the following three types: Fixed gain, Adaptive and State-space feedback. The **Fixed gain** refers to that class of controllers, where the tuning/-gain/model parameters are estimated off-line and then remains fixed at runtime. For example, the most commonly used controller of this category is the Proportional Integral Derivative (PID) or its different variants such as Proportional Integral (PI) or only Integral (I). The **Adaptive** refers to the group of controllers that have the ability to estimate the parameters at runtime thus adjusting itself to changes in the environment such as self-tuning PID controllers [95]. The **State-space feedback** controllers are used for systems that are modelled using state space [53].
- Optimal: This class of controllers consist of an optimisation base control strategy and includes the following two types: Model Predictive Controller (MPC) and Limited Lookahead Controller (LLC). The **MPC** is based on a twofold concept [107]. Firstly, it uses an internal dynamic model to predict future system behaviour, and optimise the forecast to generate the best decision at the current time. Secondly, it uses the previous moves of the controller to determine the best possible initial state of the system as the current move of the optimal control depends on it. For further details on MPC, refer to [107]. The **LLC** follows the similar concept as MPC, where the next action of the controller is determined using the projected behaviour of the system over a limited look-ahead horizon [108]. However, MPC deals with the systems operating in continuous, whereas LLC deals in discrete input-output domains [109].
- Advanced: This category clustered all those control solution methodologies that either combine multiple control methods into one or have some notion of runtime adaptive behaviour. However, the adaptation mechanism is not based on runtime parameter estimation, which is

described earlier for the *Adaptive* type in the *Basic* category. This family of controllers includes the following types: **Hybrid** refers to all those control solutions that combine more than one controllers and they all are active at the same time; **Gain scheduling/Switched** refers to all those control methods, where multiple models/controllers/gain parameters are used simultaneously. Such methodologies are accompanied with an associated reconfigurable/switching/gain scheduler layer to select the suitable model/controller/gains at runtime. Our proposed elastic methods also fall in this category.

- Intelligent: This set of controllers includes all those control systems, which are based on the use of various artificial intelligence techniques such as knowledge-based fuzzy controller, neural models based fuzzy control, neural networks, etc.

3.2.1.2 *Key Ingredients*

The attributes of this section include all the essential elements of the control methodology earlier explained in Section 2.4.2. These attributes can be seen in Figure 3.1.

3.2.1.3 *Architecture*

The architecture of a control system refers to the pattern of how a particular control methodology is implemented. The most common patterns observed in cloud elasticity research include *Centralised* and *Decentralised* (also known as *Distributed*). However, there are also few cases, where *Cascade* and *Hierarchical* patterns are used as well. The brief description of each of these patterns is as follow:

- Centralised: The control system following this architecture is implemented as one unit, which is responsible for managing the control objective from a central place, e.g. at a global system level.

- Distributed: The control system adopting distributed pattern implements at sub system level. Thus the control method is responsible for achieving the control objective at sub system level, e.g. the implementation of the controller at each VM of the cluster.
- Hierarchical: The control system in this case is implemented at two levels: lower and upper level. At the lower level, the distributed controllers manage sub-systems, whereas, at the upper level, another controller mediate distributed controllers to achieve the control objective at the global scale.
- Cascade: Using such an approach, multiple controllers work simultaneously in a way, where the decision of one controller becomes input for the next one.

3.2.2 *Elasticity View*

This section cover the attributes from the cloud elasticity perspective that defines the different aspects of an autoscaling approach. The brief description of each attribute is as follow:

1. Provider: An elasticity proposal targets the aims of a particular stakeholder, e.g. the proposed methods in this thesis is for SPs. Other possibilities include CPs or may be for both, i.e. CPs and SPs.
2. Application type: Various auto-scaling approaches are proposed for different kinds of applications. This attribute refers to the nature of the application for which the elasticity method is proposed. Possible types can be seen from Figure 3.1.
3. Trigger: This represents the triggering behaviour of an elasticity method. The possible types include *Reactive*, *Predictive* or *Hybrid*.
4. Type: The type of resource scaling can be either *Horizontal* or *Vertical*.

5. Evaluation - This attribute of taxonomy highlights how the assessment of a particular approach is carried out. This consists of the following specification:
   - Workload used: The workload used for the evaluation can be either real or synthetically generated. This attribute represents the nature of the workload and its brief description.
   - Applications used: It includes the details of any application used either for the generation of workload or experimentation purposes.
   - Environment: This attribute includes the particulars of the experimental set-up.
   - Compared with: This attribute specifies the approaches or scenarios used for comparison purposes.

3.3 STATE-OF-THE-ART REVIEW

This section provides the state-of-the-art review of existing cloud elasticity approaches that are implemented using control theory. The review is carried out by classifying the relevant proposals using controller types. The following subsections briefly discuss the proposals, whereas their details in accordance with the characteristics listed in the proposed taxonomy are provided in respective tables.

3.3.1 *Classic*

3.3.1.1 *Fixed Gain*

A generic proportional threshold based control policy using an Integral controller is proposed in [31] that adjusts the number of VMs to maintain the CPU utilisation within a desired range. The same control policy has been further utilised to maintain the performance (Response time) of storage tier of a multi-tier application in [114]. Whereas, a PID controller is proposed in [32] to adjust nodes of

Table 3.1: Fixed gain controllers

	[32]	[33]	[110]	[111]	[112] and [34]	[113]	[31]	[114]
Type	PID	PID	PID	PI	PD	Fixed gain	Integral	Integral
Model	State-space	Queuing	-	Grey-box	-	Black-box	Black-box	Black-box
Architecture	Centralized	Distributed	Centralized	Centralized	Centralized	Centralized	Centralized	Centralized
Control Objectives	99th % Read operation latency	Application SLO (response time) at a pre-defined level	Maintain a desired response time	Ensure service time constraints	Optimal number of VMs avoiding under/over utilized scenarios	Desired memory utilization	Desired CPU utilization	Maintain a desired response time
Reference Inputs	Service time	CPU utilization	CPU utilization	Service time	Server load and memory utilization	Memory utilization	CPU utilization	CPU utilization
Con Input	Number of Voldmart nodes	Number of VMs	Number of VMs	Number of map-reduce nodes	Number of VMs	Memory allocation	Number of VMs	Number of VMs
Monitoring Metrics	Read latency without round-trip time	Mean CPU utilization of tier's VMs	Mean CPU utilization of cluster	Service time and number of clients	Server load and memory utilization	CPU and memory usage, page fault rates	CPU load, arrival rate	CPU load, arrival rate
Actuators	API provided by Voldmart rebalancing tool	Used of AWS SDK	-	Linux bash script	Cloud provisioner using Amazon APIs	Balloon driver in Xen	Using Automat (a test-bed architecture) interface	Using Automat interface
Sensors	Used Voldmart server sensor	Amazon cloud watch	Hogna (a deployment framework for cloud) and cloud's API	Linux bash script	Local controller using Amazon APIs	/proc interface	Hyperic HQ [115]	Hypric SIGAR
Provider	Service provider	Service provider	Service provider	-	Cloud provider	-	Service provider	Cloud provider
Application Type	Storage	Web (n-tier)	Web (1-tier)	Data Intensive	Web (Application tier)	Generic (Web)	Generic (Web)	Storage
Trigger	Reactive	Reactive	Reactive	Reactive	[112]: Reactive, [34]: Predictive	Reactive	Reactive	Reactive
Elasticity Type	Horizontal	Horizontal	Horizontal	Horizontal	Horizontal	Vertical	Horizontal	Horizontal
Workloads	Synthetic: Generated using YCSB	Real: FIFA	Synthetic	Synthetic: used via MapReduce Benchmark Suite	Synthetic	Real: Obtain from HP lab	Synthetic	Synthetic: Using Cloudstone
Applications used	YCSB	-	Hogna Framework	MapReduce Benchmark Suite	-	httperf, MemAccess	-	Cloudstone
Environment	Real: Voldmart	Real: Amazon	Real: Amazon and SAVI [68]	Real: Grid5000 [116]	Real: Amazon	Real: Custom (4 HP Proliant servers) + Xen 2.6.16	Real: ORCA [117] + Xen as Hypervisor	Real: ORCA [117] + Xen as Hypervisor
Compared with	None provided	Threshold based, Proportional controller	None provided	None provided	[112]: None, [34]: compared with [112]	None provided	Static threshold, integral control	Static provisioning

a cloud-based storage system (named Voldmart [118]) to maintain the desired service time.

Gergin et al. [33] and Barna et al. [110] also adopted PID approach for maintaining the Response time of web applications using CPU utilisation as a reference input. In both papers, the design of the controller is similar but the adaptation is different. More specifically, Barna et al. [110] only considered one tier, whereas Gergin et al. [33] considered n-tiered transactional application and proposed a distributed architecture, i.e. using multiple controller in parallel but one for each tier. Ashraf et al. [112, 34] focused on the dynamic scaling of the application tier using a Proportional Derivative (PD) controller. This approach is not dependent on any performance model and is therefore different from all other approaches mentioned earlier, i.e. [31, 114, 33]. A PI feedback controller was introduced in [111] considering big data application that adjusts the nodes of a map reduce cluster to guarantee a desired service time and the authors of [113] focused on the readjustments of memory and CPU resources at VM level to comply with the SLO requirements.

The gains of the controllers mentioned above are estimated off-line either using a trial-and-error methods or performance model [31], application specific model [112] or using a standardised method such as Ziegler Nichols, root-locus. However, they remain fixed at runtime. Table 3.1 summarises the characteristics of the proposals reviewed in this category.

3.3.1.2 *SSF*

Li et al. [119] proposed an integrated three-layer approach. The three layers include VM, node and cloud level, amongst which the method of VM level is only relevant. Where they proposed a Multi Input Multi Output (MIMO) based SSF controller responsible for determining VM resource requirements including CPU power, memory and I/O to achieve application SLO requirements consisting of Throughput and Response time. Their method uses an on-line model estimator to capture input-output relationships at runtime. Similarly, Moulavi et al. [120] also used SSF, however for horizontal scaling of distributed cloud storage system's computational nodes.

Both approaches use Linear Quadratic Regulator (LQR) method to obtain the gains (Proportional and Integral) of their controllers. However, in [119], the gain is adaptive and may change at runtime, whereas the gain remains fixed in [120]. Moreover, Moulavi et al. [120] also used an additional fuzzy controller, which is responsible for allowing or discarding the control decision considering the standard deviations of CPU loads. The purpose of the additional controller is to avoid unnecessary fluctuations. However, no description is provided for the design of fuzzy controller. Their approach also considers cost as one of the reference input. However, it is not clear how it influences the decision of the controller. A similar approach is utilised in [121] with a difference of using Throughput as one of the reference input rather than cost. The details of the papers mentioned in this category are provided in table 3.2.

3.3.1.3 *Adaptive*

Farokhi et al. [122] proposed a performance-based adaptive feedback controller focusing on the business tier of cloud applications to adjust the memory size of a virtual machine. Their method uses an on-line linear regression method to capture the effects of controller over the performance (Response time) and to estimate a model parameter. The approach is further extended in [49], where memory utilisation is considered as a monitoring metric during model construction. Zhu et al. [130], in contrast, focused on CPU allocation using a nested control design comprising of two feedback loops. Their inner loop is an adaptive Integral controller that maintains a target CPU utilisation by adjusting the CPU allocation of VM. Whereas, the outer loop is also an Integral controller that maintains the target QoS goal (Response time) by adjusting the target CPU utilisation of the inner loop. Their research, however, lacks details on the model construction of the work.

Table 3.2: State-space Feedback and Adaptive approaches

	[120]	[121]	[119]	[122]	[123]	[40]	[124]	[125]
Type	SSF + FC	SSF	SSF (Adaptive)	Adaptive	Adaptive + hill climbing	Adaptive	Adaptive	Adaptive (Integral)
Model	State-space + LQR	State-space	Black-box + LQR	Black-box (Linear regression)	Queuing	Queuing	Queuing	Black-box
Architecture	Centralized	Centralized	Distributed	-	Centralized	Centralized	Centralized	Arbiter: Centralized, Utilization:Distributed
Control Objectives	To meet SLO (CPU load, response time and Bandwidth)	To meet performance (CPU load, response time and Throughput)	To achieve applications' SLO (Throughput and response time)	To guarantee target performance(response time)	To achieve application performance (Latency)	To guarantee SLA (max number of requests handled per time unit)	Macro level analysis to evaluate controller performance	To achieve application level QoS (response time, Throughput)
Reference Inputs	CPU load, response time, Cost	CPU load, response time, Throughput	response time, Throughput	response time	Latency	Un-handled service request in past interval	Over- and under-provisioned systems	Response time and Throughput
Control Input	Number of storage nodes	Number of VMs	CPU, Memory and I/O allocation	Memory size	Virtual CPUs	Number of VMs	Number of VMs	CPU entitlement
Monitoring Metrics	Average CPU load, Average response time, Average Bandwidth per download,total cost	CPU usage, Response time, Throughput	CPU & Memory usage, I/O Consumption	Response time	ArrivalRate, Application performance, CPU Usage, Queue length	Arrival requests, Service rate	Service capacity, Arrived requests, buffer size, Processed requests	Per VM CPU usage , Response time, Throughput, Workload Arrivals
Actuators	-	Bash scripts (in Linux)	Linux components (Scheduler, memory manager, NUMA)	-	-	-	-	CPU Scheduler of hypervisor
Sensors	-	/proc/stat (in Linux)	proc and top (in Linux)	httpmon	Hyperic HQ + custom tool	-	-	Hypervisor provided sensor
Provider	Service provider	-	Service provider	Service provider	Service provider	Cloud provider	Cloud/Service provider	Cloud provider
Application Type	Storage	Generic (Web)	Generic (Web)	Web	Generic	Generic	Scientific	Web
Trigger	Reactive	Reactive	Reactive	Reactive	Reactive	Hybrid	Hybrid	Reactive
Elasticity Type	Horizontal	Horizontal	Vertical	Vertical	Vertical	Horizontal	Horizontal	Vertical
Workloads	Synthetic	Synthetic: Generated via httperf	Synthetic: Generated via httperf	Real: FIFA, Wikipedia	Real: FIFA	Real: FIFA	Real: FIFA and Google cluster traces	Synthetic: Generated via RUBIS and TPC-W
Applications used	-	httperf	httperf, RUBiS, TPC-W	RUBBoS	Zimbra load generator	-	RUBiS , TPC-W	
Environment	Simulation: EstoreSim	Real: Eucalyptus private cloud	Real: Custom (Linux machines) + KVM	Real: Custom (Linux machine) + KVM	Real: Custom (Linux machines + Zimbra [126] server)	Simulation: Python based discrete event simulator	Simulation: Discrete event simulator [127]	Real: Custom (5 HP Proliant Servers)
Compared with	scenario without any auto-scaling	None provided	scenario without any controller	Non-adaptive (Details not provided)	Threshold-based utilization controller	Regression based controller [128]	Reactive controller [129]	None provided

Padala et al. [125] also proposed a two layered hierarchical controller to adjust CPU allocation of VMs that host individual tiers of multi-tier applications. At the VM layer, they introduced a utilisation controller (i.e. an adaptive Integral controller like the one in [130]) that maintains a desired CPU utilisation level, while adjusting the CPU entitlement thus focusing on the resource utilisation perspective. Whereas, at the top (node) layer, an arbiter controller (using fixed Integral controller) is proposed to allocate the CPU share based on the requested CPU entitlements by the utilisation controllers and the target performance objectives of multi-tiered applications sharing the same physical node. Padala et al. further enhanced this approach in [131], where they adjusted not only CPU entitlement but also Disk I/O allocation. Moreover, they changed the adaptive Integral controller with an optimiser controller aiming to minimise a cost function comprise of performance cost and control cost while obtaining the values of required resource allocations for next interval.

In contrast to CPU allocation approaches mentioned above, Kalyvianaki et al. [135] exploited the use of Kalman filter as a performance modelling technique by integrating it with feedback controller, to adjust the CPU allocation for multi-tier applications. They proposed three Kalman filter based control solutions including a Single Input Single Output (SISO), MIMO process noise covariance controller (PNNC) and an adaptive MIMO PNCC controller. The SISO is responsible for the CPU allocation of individual VMs that hosts a single application tier. MIMO PNNC is responsible for the CPU allocation of all VMs of a multi-tier application. However, both their controllers (i.e. SISO and MIMO PNNC) are fixed, where the gain of the Kalman filter does not change at runtime. The adaptive MIMO PNNC is the adaptive counterpart of MIMO PNNC where the gain of the controller is obtained at runtime. Spinner et al. [123] focused on a different perspective of CPU entitlement, wherein they adjusted the number of virtual CPUs of VMs to meet target latency of an application using queuing theory based on layered performance modelling approach in collaboration with a runtime model estimation component. The details on adaptivity are, however, not very clear. Moreover, the

Table 3.3: Adaptive approaches

	[131]	[132]	[133]	[130]	[134]	[135]	[49]
Type	Adaptive (Optimal)	Adaptive PI + pole placement	Adaptive PI	Nested Adaptive (Integral)	MIMO Adaptive PI + RL	SISO, MIMO and Adaptive MIMO	Adaptive
Model	Black-box (ARMA 2nd order)	Black-box (1st order AR model)	Least square regression	-	ARMAX (2nd order) + SVM regression [136]	Kalman filter	Linear regression (1st order)
Architecture	Distributed	-	Distributed	Cascade & Distributed	Centralized	Centralized	Centralized (Node Level)
Control Objectives	To achieve application level QoS (response time)	To achieve application level QoS (response time)	To obtain target job progress to meet deadline	To achieve target QoS goal(response time)	To maximize application benefit (QoS) within Time and budget constraints	To maintain CPU allocation right above the CPU utilization	Adjustment of memory size to achieve desire application response time
Reference Inputs	per application response time	Response time	Target job progress	Outer Loop (response time), Inner Loop (CPU utilization)	Application specific benefit function, Execution Time, resource cost	CPU Utilization	Desired response time
Control Input	CPU entitlement and disk IO allocation	CPU entitlement	CPU share	CPU Allocation	Application Adaptive parameters	CPU allocation	Memory size allocation
Monitoring Metrics	CPU Usage, Response time, Disk Usage	Response time	Job progress, Milestone	CPU Utilization, Measured response time	Application adaptation parameters, CPU and Memory usage	CPU Utilization	Measured response time, Memory utilization
Actuators	Xen CPU credit Scheduler, IO Scheduler	CPU Scheduler	-	HP-UX WLM	-	CPU Scheduler (in Xen)	Xen based APIs
Sensors	Xen xm and iostat	httperf	-	HP-UX WLM	-	-	/proc/meminfo
Provider	Cloud provider	-	Service provider	-	Service/Cloud provider	-	Service provider
Application Type	Web	Web	Scientific	Generic (Web)	Generic	Web	Web (Business tier)
Trigger	Predictive	Reactive	Reactive	Reactive	Predictive	Predictive	Reactive
Elasticity Type	Vertical	Vertical	Vertical	Vertical	Vertical	Vertical	Vertical
Workloads	Synthetic	Synthetic	Synthetic: Batch jobs	R: SPECweb99 [137] data set	-	Synthetic: BrowsingMix and BiddingMax Patterns	Real: FIFA and Wikipedia
Application Used	RUBiS, TPC-W, Custom: Secure media server	httperf	ADCIRC, OpenLB, WRF, BLAST and Montage	httferf	Great Lake nowcasting & forecasting, Volume Rendering	RUBiS	RUBBoS, httpmon
Environment	Real: 2 Custom test-beds (HP C-class blades and Emulab [138])	Real: Custom (2 machines i.e. HP9000-R server and Pentium III machine)	Real:Custom (8-core AMD server + Hyper-V and Xen)	Real: Custom (2 machines i.e. HP9000-L server and HP LPr Netserver)	Real: 2 private clusters (Each with 64 nodes)	R: Custom (3 machines with Xen 3.0.2 Hypervisor)	R: Custom (32 cores and 56 GB memory based machine + Xen hypervisor)
Compared with	Two cases: Work-conserving and static allocation	Fixed PI controller	Feedback approach of [139]	Single loop QoS controller, utilization controller	Work conserving, Static scheduling	None provided	Capacity based [140] & Performance based [141] memory controllers

final output (i.e. scaling decision) of the controller is fixed where at each control interval only one virtual CPU can be added or removed.

There are several papers that propose the adaptive version of PI controller. For example, Liu et al. [132] used it to maintain the desired Response time of web server by adjusting the CPU entitlement of a shared resource container. Park and Humphrey [133] focused on the scientific domain to achieve a target progress of high-performance computing jobs adjusting the CPU entitlement using an adaptive PI. Whereas, Zhu and Agrawal [134] focused not only on CPU but also on memory size adjustment by proposing an adaptive version of MIMO PI controller in collaboration with a reinforcement learning component. This approach, however, is different in two aspects from any other approaches mentioned in this section. Firstly, it does not directly control resources but rather change some adaptive parameters of the cloud applications. Secondly, it aims to maximise the application specific benefits QoS within a pre-specified time and budget constraints.

In contrast, to the above approaches, Ali-Eldin et al. [40] proposed an adaptive hybrid controller for horizontal scaling using queuing theory as a modelling technique. The queuing based model determines the total service capacity required per control interval while considering the arrival rate of the concurrent requests. The output of the controller is dependent on a dynamic gain parameter. The value of the parameter itself is obtained at runtime using the change in demand on the past time unit and the necessary service capacity. The same approach is applied in [124] with an enhanced model and controller design, which also considers buffer size, the delay caused due to the VM start process, allocated capacity and changing request service rate of the VM. Table 3.2 and 3.3 summarize various aspects of the adaptive approaches mentioned in this section.

3.3.2 *Optimal*

3.3.2.1 *MPC*

The readjustment problem of data centre capacity with a focus on energy saving perspective is address using an MPC based approach in Zhang et al. [36]. They consider and include aspects like cluster reconfiguration cost (due to saving/-loading/migration of systems' state), electricity price fluctuation (as rates vary at different time of the day in some countries, e.g. the USA). Their work, however, assumes that (1) all data centre machines have the same computational capabilities and (2) the instances of incoming workload share similar characteristics, e.g. task length, resource requirements. Their work is further extended in [37], where they considered heterogeneous hardware and workload behaviour. They approached the heterogeneity issue using MPC controller coupled with a k-means clustering algorithm, which is used to cluster the tasks into various groups based on the identical characteristics, i.e. performance and resource requirements.

The MPC based approach in [38] decides a sequence of resource reservations actions for N steps rather than a single decision for next control input. The paper, however, lacks details about the results obtained. Whereas, Cerf et al. [142] focused on the idea of reducing the number of elasticity decisions for big data cloud systems using an MPC controller coupled with an event triggering mechanism. The event triggering mechanism serves as an additional layer to determine whether the MPC decision will be carried out or not. The mechanism is based on the optimal cost function rather than the state of the system or any form of control error mechanism.

In contrast to MPC approaches mentioned above, Lama et al. [143] proposed a distributed MIMO control solution to address vertical elasticity. They used multiple MPC to manage the allocation of resources (CPU and memory) to achieve the target performance requirements of the co-located multi-tier web applications deployed on shared computational nodes of a data center. Each MPC handles one application and controls the allocation of resources of all their respective VMs whilst considering that each tier of the application deployed on a separate VM, which may reside in different computational nodes than other tiers' VMs. They used neural network based fuzzy models and considered variables of the local controller as well as the neighbour controller, which manages VMs of other applications that share the underlying physical resources.

A few other MPC based proposals include reconfiguration of storage system [144], resource management of multiple client classes in shared environment [145], performance optimisation using power control [146], and dynamic resource allocation using an integrated approach of fuzzy model and MPC [147].

3.3.2.2 *LLC*

Kusic and Kandasamy [148] utilised an LLC mechanism for enterprise computing system by formulating the resource provisioning problem as a sequential optimisation problem. They approached the problem by considering multiple, i.e. fixed three client classes, each with different QoS requirements. A different cluster is used to manage one client class focusing on reducing operating cost regarding switching cost and minimisation of energy consumption. The distinct feature of their approach is the consideration of provisioning decision risk as a factor and encoding it into the cost function while considering the variability of workload patterns. Each of their decision determines not only the number of machines in each cluster but also the operating processing frequencies associated with different pricing regarding power.

The same approach is adopted in [109] for virtualised environments with the following modifications. The controller decision concludes the allocation of CPU share of each virtual machine for each cluster rather than specifying operating frequencies, the indication of active host machines and the share of workload to be assigned to each virtual machine. Bai and Abdelwahed [149] demonstrated the use of artificial intelligence based search methods on a case study of processor power management to address the problem of computational overhead caused while using LLC.

3.3.3 *Advanced*

3.3.3.1 *Hybrid*

Dawoud et al. [156] proposed a hybrid scheme, which consists of three feedback controllers to dynamically allocate CPU and memory resources to VMs. Each of the controller is responsible for CPU allocation, memory allocation and application performance tuning respectively. All of the three controllers run in parallel albeit independent from each other. Therefore, it is not clear how and why the decision of one controller does not have any effect on others. Similarly, Farokhi et al. [141] focused on both CPU and memory allocation of VMs by using two controllers. Their approach, however, consists of a fuzzy based coordination control that determines the contribution of CPU and memory allocation based on the application performance.

In contrast to the above approaches, Xiong et al. [157] only focused on the CPU share allocation to individual virtual machines using a two-level hybrid control scheme. This approach consists of an inter-dependent optimal and PI feedback controller to allocate CPU budget to each VM hosting distinct tiers of an N-tier web application. The PI controller is responsible for calculating the required CPU budget for each VM while considering the application performance target. Similarly, the optimal controller decides the actual (to be allocated) CPU share of each VM by solving an optimisation problem. This approach, however, considers the total CPU budget for an application is fixed, and each VM will get a share of the total budget. Rao et al. [158] approached the problem of multi-objective resource management using a two-level approach. At the first level, an individual fuzzy controller per objective decides the resources needed to maintain

Table 3.4: Optimal approaches

	[143]	[38]	[142]	[36] and [37]	[148]	[109]	[149]
Type	Distributed MIMO MPC	Stochastic MPC	MPC + Event triggering	[36]: MPC, [37] MPC + k-means clustering	LLC	LLC	LLC + Search methods
Model	Fuzzy logic + Artificial Neural Network	Queuing + Kalman filter	G	ARIMA [150]	ARIMA + Kalman Filter	Kalman Filter	ARIMA + Kalman Filter
Architecture	Distributed	-	Centralized	Centralized	Centralized	Hierarchical	Centralized
Control Objectives	To achieve end-to-end desired response time of co-located web applications	To minimize application response time and cost	To reduce cluster reconfiguration decisions	To obtain optimal trade-off between energy saving and reconfiguration cost	To maximize profit and minimize power consumption	To maximize profit and minimize power consumption and SLA	To obtain a desired response time while reducing power consumption
Reference Inputs	per application response time	-	-	SLA (Average task scheduling delay)	Response time of each client class	Response time of each client class	Response time
Control Input	CPU and Memory allocation	Number of VMs	Map-reduce cluster size	Number of machines	Cluster size, operating frequencies	Cluster size, CPU Share, Active host machines, Workload share of VM	Operating frequency
Monitoring Metrics	CPU and Memory usage of local and neighbour VMs, Energy usage, Measured response time	Resource usage, Measured response time and cost, Service rate of cluster, Queue length, Arrival rate	Number of clients, Service time, Availability	Queue length, SLA, CPU and Memory usage, Arrival rate	Arrival rate, Measured response time	Queue length, Arrival rate, Processing rate, Measured Average response time	Arrival rate, Queue length, Power consumption
Actuators	Vsphere API's (ReconfigVM_Task method)	-	-	-	-	ESX server control APIs + Linux shut down command	-
Sensors	Vsphere APIs, XML-RPC Daemon	-	-	-	-	-	-
Provider	Cloud provider	Service provider	-	Cloud provider	Cloud provider	Cloud provider	Cloud provider
Application Type	Web	Generic	Big Data systems	Generic	Generic	Generic	Generic
Trigger	Predictive	Predictive	Reactive + Predictive	Predictive	Predictive	Predictive	Predictive
Elasticity Type	Vertical	Horizontal	Horizontal	Horizontal	Horizontal/Vertical	Horizontal/Vertical	Vertical
Workloads	Real: FIFA	Real: FIFA	Real:Taobao website trace [151]	Real: Traces of a production cluster at Google [152]	Real : FIFA	Real : FIFA	Real: http traces of a Washinton DC Internet server [153]
Application Used	RUBiS and custom web service application	Convex optimization solver	Map-reduce framework	-	-	IBM's trade6 benchmark, Httperf	-
Environment	Real: Custom (Dell PowerEdge R610 servers + VMWare enviornment)	Simulation: Custom simulator	Simulation: Matlab Simulink	Simulation: Trace driven simulation used in ([36]), matlab in ([37])	Simulation: Matlab simulation	R: Custom (Dell PowerEdge servers + VMWare ESX)	Simulation: Matlab
Compared with	Perfume [154], PAC [155]	None provided	Time based control mechanism, Error based event mechanism	[36]: None provided, [37]: heuristic based provisioning	A system without LLC	A system without LLC	Comparison among various search methods

the desired performance level, whereas, at the second level, a gain scheduler component aggregates the individual requests and forms a final decision using the dynamically computed gains (weights) based on the control errors.

Other hybrid approaches use the combination of a feedback and feed-forward methods, e.g. [159, 160, 161, 162]. The idea behind such merging is to exploit feed-forward to control the large spikes in workload using any model predictive method, whereas, the gradual changes in the workload, as well as the rectification of modelling errors, using the feedback approach. Al-Shishtawy and Vlassov [159] used such an approach to perform horizontal scaling of cloud-based key-value stores. According to their methodology, the scaling decision will be either carried out with feed-forward or feedback and it depends on the intensity of the workload. Their approach, however, is different in general than any other related horizontal approaches as they use average throughput per server as the control input in contrast to the commonly used number of servers.

Wang et al. [160] focused on vertical elasticity to adjust CPU allocation of virtual containers that hosts different tiers of an application. Their feed-forward method estimates optimal CPU utilisation level, whereas the feedback controller further tunes the utilisation target for individual containers that are maintained by the distributed utilisation controllers. In the case of [159], either feedback or feed-forward control executes at a time as their control interval is the same, whereas, in the case of [160], all controllers run at different control intervals. Their approach is further extended in [161] to manage the performance of multiple applications. The extension includes the consideration of hybrid controller of [160] as the application controller, which is responsible for calculating required CPU entitlement necessary to obtain the desired performance target. Furthermore, the addition of a node level controller (Integral) to manage the CPU entitlement of the respective VMs on a given node. Kjaer et al. [162] combined a PI and a Proportional based feed-forward method. Their feed-forward approach is, however, different as it uses an on-line performance model in contrast to other related hybrid approaches mentioned in this category which use off-line performance model.

3.3.3.2 *Gain Scheduled/Switched Controllers*

Grimaldi et al. [167] proposed a PID gain scheduling approach to dynamically adjust the number of VMs to maintain the desired CPU utilisation. The gain scheduler is based on the minimisation of a cost function to obtain optimised values of the controller gains in a particular operating region (characterised by different workload and timeslots) with an objective to reduce the control error close to zero. Certain details such as modelling aspects and how CPU utilisation is related to the end-user performance are missing. A Linear Parameter Varying (LPV) modelling approach is followed in [168] to guarantee web server performance (Throughput and Response time) by dynamically adjusting the number of VMs. They utilised a gain scheduled LQR design approach, where the CPU utilisation of VMs is used as a scheduling parameter. In contrast, Qin and Wang [169] used $\mathrm{LPV} - H_\infty$ controllers as their design approach for calculating the aggregate CPU frequency needed to maintain a target Response time, which was then used to compute the number of VMs. They used arrival rate of the workload and average service rate as the scheduling parameter for the characterisation of time-varying operating conditions. Similarly, Tanelli et al. [170] also used arrival rate and effective service rate per application as their scheduling variables in their proposed MIMO LPV approach. However, they focused on the dynamic allocation of CPU capacity to individual VMs that share the same physical system.

Patikirikorala et al. [171] proposed a multi-model switching control system to dynamically allocate CPU capacity to the VMs of a physical machine to maintain response time objectives of an individual application. Their approach distributes the overall system in two operating regions using a threshold level of Response time. A specific model is then designed to represent the behaviour of the system in each operating region. Their approach uses multiple fixed PI feedback controllers, which on runtime change based on the operating region using if-else

Table 3.5: Hybrid approaches

	[156]	[159]	[160] and [161]	[162]	[157]	[141]	[158]
Type	Adaptive integral + Fixed gain + Heuristic	Feedback (PI) + Feed-forward (MPC)	[160]: Feedback + Feed-forward, [161]: I + controller of [160]	Feedback (PI) + Feed-forward (P)	Optimal + Feedback (PI)	Adaptive + Fuzzy controller	Multiple Fuzzy controllers
Model	Black-box	Black-box	Queuing + Transaction mix [163]	Queuing	Queuing + ARMA	Fuzzy models + Black-box (Linear Regression)	Fuzzy Models
Architecture	Centralized	Centralized	Hierarchical and Distributed	Centralized	Centralized	Centralized	Centralized
Control Objectives	Desired application performance and reduction of SLO violations	To regulate target performance of key-value store	Desired response time of ([160]: one)/([161]: multiple) multi-tier applications	To minimize CPU capacity allocation per application, and maintain desired response time	To obtain optimal partitioning scheme that reduce round trip time	To maintain a target response time	Adaptive allocation of multi-objective resources and service differentiation
Reference Inputs	Response time	Desired 99th percentile of read latency (R99)	[160]: response time, [161]: response times of applications	Average response time	round trip time	Response time	Response time, Throughput, play rate
Control Input	CPU Capacity, Memory allocation, MaxClient value	Average throughput per storage server	CPU Capacity	Shared CPU capacity	CPU budget	CPU and memory allocation	CPU, Memory, Disk bandwidth
Monitoring Metrics	CPU and Memory Usage, Response time, throughput, Number of Apache processes	R99, Average throughput per server	Transaction Mix, Measured response time, CPU Utilization	Arrival rate, Response time, CPU usage	Arrival Rate, Average round trip time, CPU usage, Average resident time	Average CPU & memory usage, mean response time	Response time, Throughput, play rate
Actuators	Xen credit scheduler, Balloon driver	Voldemart rebalancing tool	Credit scheduler	-	-	Xen API	Xen Credit Scheduler, memset, 1sof command
Sensors	xentop, memset	-	xen Hypervisor provided interface	-	-	Xen API	Via customization of Workload generators
Provider	Service provider	Service provider	Cloud provider	-	Cloud provider	Service provider	-
Application Type	Web	Data tier	Web	Web	Web	Web (Business tier)	Generic
Trigger	Predictive	Hybrid	Hybrid	Hybrid	Reactive	Reactive	Predictive
Elasticity Type	Vertical	Horizontal	Vertical	Vertical	Vertical	Vertical	Vertical
Workloads	Synthetic	Synthetic	Real: Transaction traces of VDR [164]	Real: Sweden based News portal traces	Real: FIFA	Synthetic: based on open and close models	Synthetic
Application Used	httperf, autobensh tool	Voldemart (version 0.91)	Customized RUBiS + Custom workload generator	CRIS Tool	RUBiS	RUBiS, RUBBoS, Olio and httpmon	TPC-W, key-value store, video streaming
Environment	Real: Custom (Intel Quad Core i7 + Xen 3.3)	Real: OpenStack (Cluster consist of 11 nodes)	Real: custom (HP ProLiant servers + xen 2.6)	Simulation: Java simulator, Real: Custom (Linux server)	Real: Custom (Linux based Pentium 4 machines + xen 3.0.3)	Real: Custom (1 machine with 32 cores + 56 GB memory) + Xen	Real: Custom (2 Dell servers + Xen)
Compared with	Static scenario without reconfiguring resources	None provided	Fixed entitlement, Utilization only, Nested control, Feed-forward + Utilization	Modified versions of [165, 166]	Schemes like Equal shares and Equal utilization	None provided	Kalman Filter, Adaptive PI, ARMA

based switching mechanism. In contrast, Saikrishna et al. [172] used ten distinct operating regions and arrival rate as the switching signal.

3.3.4 *Intelligent*

Jamshidi et al. [16] proposed a fuzzy controller to address the uncertainty in cloud environments and to provide horizontal elasticity rules using qualitative specification in contrast to the commercially provided quantitative if-else rules. They designed the membership functions and rules using the knowledge (consisting of the intensity of workload and Response time) extracted directly from the domain experts. However, they followed a static scaling approach, where the number of VMs to be added or removed as a result of scaling action is pre-defined fixed numbers. Their approach is further extended in [35, 174], where they make use of fuzzy Q-Learning to learn the best elastic policies (fuzzy rules) at runtime considering scenarios of systems, where the domain knowledge may not or partially available.

In contrast to the approaches mentioned above, Xu et al. [175] focused on vertical elasticity. They proposed two fuzzy based methods (i.e. modelling and prediction as referred by them) to dynamically estimate the computational resources of a VM needed by an application. Their modelling approach builds a fuzzy model at runtime by directly monitoring/learning the relationship between application workload, resource usage and performance. Whereas, their adaptive prediction technique only considers the observations of resource usage to estimate future resources. A similar approach (adaptive fuzzy modelling) is followed in [176] that estimates multi-type resources (i.e. CPU capacity and disk IO bandwidth) of VMs specifically for database systems.

The approach in [177] on multi-tier applications for guaranteeing 90th percentile end-to-end delay by computing the number of servers to application tiers. This method considers control error (i.e. the difference between measured and the

Table 3.6: Gain Scheduling/Switched approaches

	[167]	[168]	[169]	[171]	[172]	[170]
Type	GS (PID)	GS (SSF + LQR)	GS ($LPV - H_\infty$ + LQR)	Multi-model PI	SSF + pole placement	LPV controller
Model	-	Grey-box + Least square regression	ARX + LPV-ARX	Black-box	State-space + Grey-box	State-space LPV models
Architecture	Centralized	Distributed	-	Centralized	Distributed	Centralized
Control Objectives	To maintain a desired CPU load	To maintain web server performance(response time and throughput)	To dynamically control the aggregate CPU frequency to maintain target response time	To optimize share of CPU capacity among all VMs to meet desired response time	To maintain a desired response time	To maintain target response time for each application
Reference Inputs	CPU Load	Response time	Response time	per application/VM average response time	Response time	Response time
Control Inputs	Number of VMs	Number of VMs, Admission Control	Aggregate CPU frequency	CPU Capacity	Number of VMs, Admission control	per VM CPU capacity share
Monitoring Metrics	CPU Utilization, Number of VMs	Utilization of VMs, Throughput, Measured response time, Service rate	Number of jobs, Queue length, Throughput, Mean response time	per application response time	Utilization of VMs, Throughput, response time	per application (Arrival rate, effective service time, response time)
Actuators	-	KVM provided API	-	Credit scheduler	-	-
Sensors	Amazon Cloud watch	Customized httperf, bash script	-	Authors-written Java program	-	-
Provider	Service provider	-	Cloud provider	Cloud provider	-	Cloud provider
Application Type	Generic	Web	Generic	Generic (Web)	Web	Web
Trigger	Reactive	Reactive	Reactive	Reactive	Reactive	Reactive
Elasticity Type	Horizontal	Horizontal	Horizontal	Vertical	Horizontal	Vertical
Workloads	Synthetic: Stable and highly variable patterns + Real	Synthetic	Real: web traces from [173]	-	Synthetic	Real: on-line banking application web traces
Applications Used	Httpmon	httperf	-	RUBiS	httperf	Customozied Apach Jmeter, Micro-benchmarking web service
Environment	Real: Amazon	Real: Eucalyptus + KVM hypervisor	Simulation: Customize CSIM simulation	Real:Custom (3 machines) + xen 2.6	Real: Eucalyptus + KVM hypervisor	R:Custom (8 VMs) + Xen 3.0
Compared with	Fixed gain PID	None provided	Linear models, Queuing theory	Single model controllers	None provided	None provided

desired end-to-end delay) and rate of change in error as performance metrics and not rely on any workload related attribute. An alternative mode of their approach involves the use of an additional optimisation-based component. In such a case, the output of the fuzzy controller becomes an adjustment parameter to the output of the optimisation component. The approach in [178] have focused on vertical elasticity and adjust the CPU capacity assigned to VMs by considering the differences between (1) measured and target performances, and (2) the current CPU allocation and the actual utilisation. This approach is further extended in [179] to also take into account the memory readjustment in collaboration of CPU capacity. Moreover, they provided a cascade based coordination mechanism at the time of scaling decision to consider the joint effect of both kinds of resources. Such coordination mechanism has not been considered by many authors for the same problem, e.g. [154, 156, 147] except in [141]. Some other examples of fuzzy approaches include neural fuzzy control [180], fuzzy logic based feedback controller [181], fuzzy model coupled with a performance prediction model [182] and multi-agent fuzzy control [183].

3.4 ISSUES AND CHALLENGES

Irrespective of the considerably wide range of control theoretical approaches to implement cloud elasticity, there are still various issues and challenges that have not receive much attention. Based on our analysis, we list the following such issues and challenges that require further attention:

1. Lack of adaptation: The fixed approaches are designed off-line. They do not involve any on-line learning mechanism to adapt at runtime in response to changes. Therefore, they often get credit for (i) their simplistic design, (ii) computationally economical behaviour due to the absence of any computational overheads. Such approaches are better for systems subject to uniform workload behaviour. However, the performance of the system largely suffers, in situations when workload fluctuates at different times.

Table 3.7: Intelligent approaches

	[16]	[175]	[176]	[177]	[178]	[179]	[35]
Type	Fuzzy type-2 Controller	Fuzzy controller	Fuzzy controller	Fuzzy + Optimization	Fuzzy controller	Fuzzy controllers	Fuzzy self-learning controller
Model	Fuzzy model	Zero-order Sugeno-type Fuzzy model + Clustering	Fuzzy model + Classification and Clustering	Fuzzy model + Queuing	Fuzzy model + stochastic approximation algorithm [184]	Fuzzy model + t-digest [185]	Fuzzy + Q-Learning
Architecture	Centralized	Distributed	Distributed	Centralized	Distributed	Distributed and Centralized	Centralized
Control Objectives	To maintain a target response time	To adjust per VM CPU capacity to meet performance goals	To adjust per VM CPU & disk IO to meet performance goals	To guarantee 90th Percentile end-to-end delay in multi-tier systems	To maintain performance SLOs of an application	To maintain performance SLOs of an application	To maintain a desired response time
Reference Inputs	95th percentile of response time	-	-	90th Percentile end-to-end delay	Response time, Throughput	Response time and Throughput	Response time
Control Input	Number of VMs	CPU capacity	CPU Capacity, Disk IO bandwidth	Number of servers	CPU capacity allocation for each tier	CPU and memory allocation for each tier	Number of VMs
Monitoring Metrics	Measured response time, Arrival rate	CPU usage, throughput (reply rate), Workload	CPU and disk IO Usage, Average query throughput, Measured response time	end-to-end delay	Mean CPU usage per tier/VM, Measured SLO	Mean CPU & memory usage per tier/VM, Measured SLO	Arrival rate, Measured response time, Throughput
Actuators	Azure management service	-	Xen CPU Scheduler, Linux's dm-ioband	-	Xen credit scheduler	Xen credit scheduler, balloon driver	Azure and OpenStack API
Sensors	Performance counters	Using Java code	xentop, iostat	-	libvirt daemon	libvirt daemon, /proc/meminfo command	Azure and OpenStack API
Provider	Service provider	Service provider	-	-	Cloud provider	Cloud provider	Service provider
Application Type	Generic (Business tier)	Generic (Web)	Database	Web	Generic (Web and database)	Generic (Web, database, in-memory data store)	Generic
Trigger	Reactive	Predictive	Predictive	Reactive	Predictive	Predictive	Reactive
Elasticity Type	Horizontal	Vertical	Vertical	Horizontal	Vertical	Vertical	Horizontal
Workloads	Synthetic: Patterns adapted from real workloads (e.g [173])	Synthetic + Real:FIFA	Synthetic + Real: FIFA	Synthetic	Synthetic: generated using RAIN, YCSB	Synthetic: generated using RAIN	Synthetic: six patterns generated using ElasticBench
Application Used	Jmeter	Java Pet store, httperf	TPC-H, RUBiS	Three tier application	RUBiS, Olio, Cassendra	RUBiS, RUBBoS, Olio, Cassendra, Redis	ElasticBench
Environment	Real:Microsoft Azure	Real:Custom (16-CPU IBM x 336 based cluster) + VMWare ESX 3.0	Real: Custom (Quad-core Intel Q6600) + Xen 3.3.1	Simulation: Simulation (No information provided)	Real: Custom (2 Fujitsu Servers) + Xen 4.2	Real: Custom (2 Fujitsu Servers) + Xen 4.4	Real: Azure and OpenStack
Compared with	Over/Under-provisioning scenarios	None provided	Peak load based allocation scheme	With/Without optimization scenarios	DynaQoS [158], AutoControl [131]	APPLEware [143], FMPC [147]	Fuzzy controller without learning, Azure rule-based

2. Computational overhead: The lack of adaptability issue in fixed approaches is addressed with the use of adaptive control methodologies, where adaptive behaviour is obtained by exploiting machine learning algorithms and constant monitoring. Such methods result in comparatively better performance and efficient decision making. However, this only applies, where the workload behaviour is predictable or when it changes slowly. The achievement of better performance comes at the cost of being computationally expensive due to the use of on-line learning algorithms or predictive modelling. Moreover, such methods also fail when workload behaviour is unable to predict, and they are therefore unable to cope with sudden changes in workloads.

3. Oscillation: The design of elastic feedback controllers requires careful attention and detailed evaluation because badly designed controller may result in oscillation and instability [5]. The control methodologies based on multiple models/controllers are criticised specifically for the phenomenon like bumpy transitions that leads to oscillatory behaviour. In such a situation, the cloud resources are acquired and released periodically. The existing researches on elastic feedback controllers lack on providing an explanation and evaluation on how a proposed method deals with the undesirable oscillatory behaviour.

4. Over-provisioning and resource usage: Over-provisioning is commonly used to avoid performance violation considering peak workload scenarios [76, 77]. However, this results in the wastage of resources. It is therefore undesirable and should be avoided. In the existing research works, the authors mostly provide the evaluation of the proposed methodologies regarding achieving performance objective. However, an explanation or comparative analysis of the methodology in the prospect of minimising the computational resources is missing. A comparative cost analysis of the state of the art approaches is needed to justify that a proposed method is not costly and do not acquire more than needed resources to improve system performance.

5. Uncertainty: An elastic method depends and collaborates with other components of the system irrespective of the nature of underlying implementation technique. For example, the dependency of an elastic method on monitoring component of the system to obtain the latest status of system metrics to make scaling decision. Thus a system, with a lot of dependencies on other parts/components of the system is likely to face challenges related to uncertainty. Some examples of such uncertainty behaviour include inaccuracy in monitoring information, delays caused due to actuator operation, failure of a VM, noise in input data, the unpredictability of workload, inaccuracies in performance model [47, 186, 46]. The existing research works on cloud elasticity, in general, has not paid much attention to consider such aspects while designing auto-scaling system.

6. Evaluation and Benchmarking: The performance of a control methodology is sensitive to the changes in workload. Therefore, an extensive evaluation of the control methodology is required. However, most of the existing methods have been evaluated only using less than three workload scenarios [187]. Moreover, as can be seen from tables (3.1 to 3.7), the majority of papers also lack on providing details of comparative evaluation. Furthermore, the unavailability of benchmark frameworks makes it difficult to evaluate the related approaches. A recent development in this regard is the performance evaluation framework proposed in [187]. However, this framework is in general for an auto-scaling strategy. There is also a need for specific control theoretic benchmarks that have the ability to facilitate analysis of control theoretic specific characteristics, e.g. SASO (Stability, Accuracy, Short settling and Overshoot) properties as discussed and suggested in [188].

7. Scalability: It is observed, during our analysis that most of the control methods are either designed or tested for web applications. The evaluation and analysis by the authors of respective approaches are performed at small scale, i.e. using a workload spanning of hours/days or fewer number of VMs [187]. However, experimentation and description on the suitability of

control methodology at larger scale considering realistic enterprise level web applications is missing.

8. Heterogeneity: In the case of horizontal scaling, most of the existing approaches consider VMs with same computational specifications. Similarly, in the case of vertical scaling, the focus of most approaches are either on the dynamic adjustments of one computational resource, i.e. either memory, CPU or I/O. Alternatively, if in the case of multiple resources, the management of each computational resource is independent of each other. However, renting homogeneous servers or management of one computational resource is not always pragmatic. Considering heterogeneous servers for horizontal elasticity creates challenges for building efficient and accurate performance models because of their different computational capabilities. Similarly, controlling multiple computational resources at fine-grained level requires methods to identify and establish better coordination protocol between controllers at runtime and with the environment.

4

TOWARDS A MULTI-CONTROLLER BASED HARD SWITCHING ELASTICITY FRAMEWORK

This chapter proposes a novel, multi-controller with fuzzy switching based horizontal elasticity framework. The chapter starts with a brief definition of the horizontal elasticity problem in Section 4.1. Section 4.2 identifies the key requirements for the development of the proposed framework. Section 4.3 elaborates the proposed framework. The key components of our proposed framework including the feedback control loop and switching mechanism are explained in Sections 4.4 and 4.5 respectively.

4.1 PROBLEM DEFINITION

This thesis considers that a web application is deployed in a cloud environment using a pool of N VMs. For the sake of simplicity we also assume that all VMs are identical regarding their hardware specification. A load balancer is used to support the deployed web application by managing the assignment of incoming requests to the available VMs. The load balancer has an access to the updated information of all the VMs and follows a specified scheduling policy such as a *First come first serve*, *Round robin* or a *Priority based*. The detail explanation of scheduling policies or determining their impact is beyond the scope of this thesis. All the experimentations in this thesis are conducted using a *Round robin* policy where the arrived jobs are assigned to VMs in a circular order and without considering any kind of priority. The proposed multi-controller elasticity framework is responsible to govern the dynamic scaling of the deployed web application. Thus, whilst considering the horizontal elasticity problem, the proposed framework dynamically determines an optimal number of required VMs, sufficient enough to achieve a targeted performance objective at a lowest possible cost.

Over the last decade, researchers and practitioners have introduced a large number of auto-scaling solutions to cover different aspects of resource management aiming to achieve better performance [75, 78, 45]. A subset of these solutions using control theoretical techniques are reviewed in Chapter 3. Based on this review, we have identified the following three shortcomings that have not received much attention in the existing research.

1. The nature of existing methods varies from each other in different dimensions including but not limited to the triggering behaviour, scope, purpose, dependency on metrics and implementation techniques [45, 99, 74, 91]. Despite such differences they can be mainly grouped into *Fixed* or *Adaptive* based on their design and working mechanism to analyse their pros and cons as a whole [44].

    - Fixed: This family of elasticity approaches are designed off-line and remain fixed at runtime such as the auto-scaling approaches developed using *Rules-based* [27, 28, 29, 30], *Fuzzy logic* based approaches (reviewed in Section 3.3.4) and *Fixed gain* feedback control approaches (reviewed in Section 3.3.1.1). The approaches following a *Fixed* based design methodology are (1) simple and easy to design, (2) they do not involve additional computational overhead and (3) they are better for systems with uniform workload behaviour. However, the performance severely affect systems with variable workloads due to lack of adaptability at runtime.
    - Adaptive: The Adaptive on the other hand include those approaches that involve some sort of an on-line learning algorithm responsible for the reconfiguration at run time due to the change of system behaviour. For example, the elastic methods developed using machine learning techniques [21, 189, 190], adaptive control theoretical approaches (reviewed in Section 3.3.1.3) and optimisation based methods (reviewed

in Section 3.3.2). In general, such adaptive control methodologies have the ability to modify themselves to the changing behaviour in the system environment that makes it suitable for the systems with changing workload conditions. However, such approaches are also criticised for an additional computational overhead caused by the on-line analysis required for predictions [45], e.g. the on-line estimation of gain parameters in case of adaptive feedback controllers [45], the long training delays in the case of reinforced learning [45], the brute-force search time in the case of optimisation based approaches [109]. Furthermore, they are also blamed for their associated risk of reducing the quality assurance of the resulted system, and the impossibility of deriving a convergence or stability proof [44]. The methodologies developed using an Adaptive approach are predictive in nature and perform better in comparison to *Fixed* approaches. However, they fail or perform poorly in situations where the predictions are not possible or not accurate [75], e.g. when there are abrupt changes in the workload behaviour.

Based on the two-way classification mentioned above and considering their pros and cons, this thesis advocates the idea of adopting the best of both worlds, i.e. an elastic methodology which can be developed using fixed design approach but which also inherit an adaptive behaviour at run time. Such an approach will have the inherent advantage of ease in design with no additional computational overhead. Some of such existing proposals, e.g. [167, 168, 169, 170, 171, 172] are reviewed in Section 3.3.3 and will be further discussed in Section 8.1 comparative to the proposed method.

2. The auto-scaler in the cloud environment face many challenges raised due to uncertainty aspects present in the cloud environment. Some examples of such uncertainty behaviour include inaccuracy in monitoring information, delays caused due to actuator operation, failure of the VMs themselves, noise in the input data, the unpredictability of workload and inaccuracies in the performance model [47, 186, 46]. The implementation of uncertainty

related aspects in the existing elasticity research work has not received much attention, therefore, the research work carried out in this thesis advocates the use of fuzzy control system to take into account the uncertainty related aspects into auto-scaling decisions.

3. Another important consideration is the popularity of Rule-based method explained earlier in Section 2.4.1. The key benefit of such an approach is its simplistic design. However, it suffers mainly from the following two shortcomings i.e. (1) A detailed knowledge of the system itself is required to set the thresholds quantitatively; (2) it is unable to cope with unpredictable events because of its static scaling behaviour. Thus, whilst considering the drawbacks of static scaling and quantitative rules, this chapter complements and extends the idea of qualitative elasticity rules initially proposed by Jamshidi et. al [16].

In light of the above discussion, this chapter proposes a multi-controller with fuzzy switching based elasticity framework aiming to address the above mentioned points. The simultaneous use of multiple controllers with switching ability facilitates to achieve the adaptive behaviour using a fixed design approach. Whereas, the implementation of fuzzy based switching mechanism helps to address the uncertainty related challenges.

4.3 MULTI-CONTROLLER WITH FUZZY SWITCHING FRAMEWORK

The proposed multi-controller with fuzzy switching framework consists of the use of an array of controllers, where each controller is particularly designed to achieve better performance in a different situation and the selection of a suitable controller is realised at runtime. The architectural diagram of the proposed control methodology is given in Figure 4.1 that extends and builds on the feedback loop model. A detailed explanation of feedback control loop model is provided in Section 2.4.2. The key components of the proposed control method are shown in Figure 4.1 that consists of the *Feedback control loop*, a *System Monitor* and

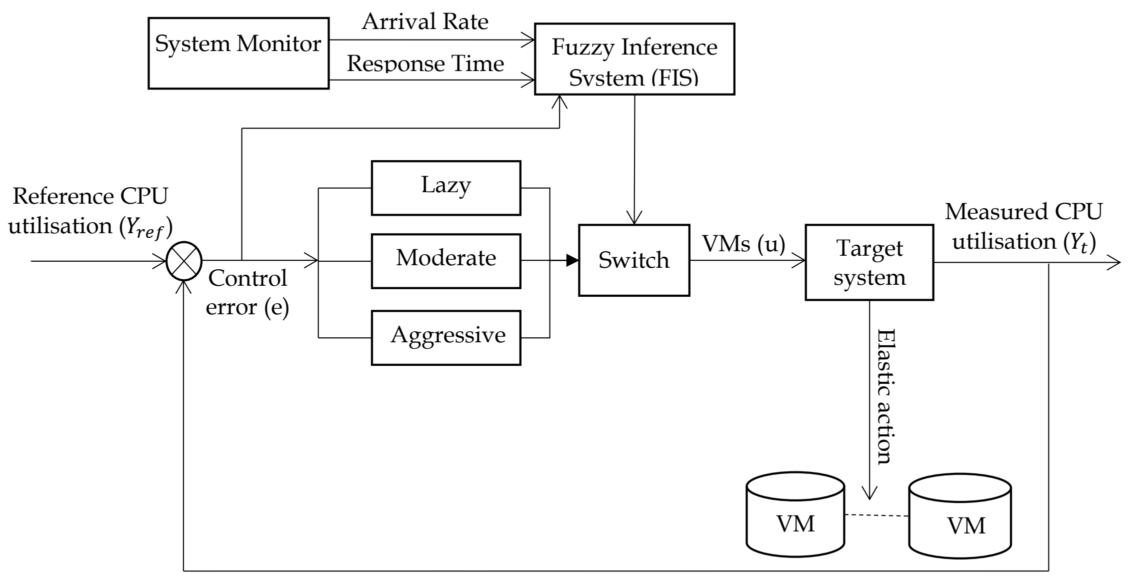


Figure 4.1: Architectural view of the proposed cloud elasticity framework using multiple controllers with fuzzy switching

the FIS. The individual design details and working mechanism of *Feedback control loop* and *Switching mechanism* are discussed in Sections 4.4 and 4.5 respectively.

The key idea behind the proposed framework is to divide the complexity of the overall system by constructing multiple fixed gain controllers, where each controller depicts a separate elastic policy that carries out scaling actions at different intensity level. The switching based multiple controller approach with some variations is analogous to other popular methods like *MMST* and *Gain scheduling*. We treat these methods similarly and therefore, all the existing auto-scaling methods based on any of these mechanisms are considered as related works. Such proposals are reviewed in Section 3.3.3.2 and are further discussed in comparison to our proposed method in Section 8.1.

Designing the proposed methodology (or any switched method in general) involves the following two key challenges including: (1) how to partition the system among multiple controllers? (2) How to switch (or formulate) the final decision? Due to the lack of a standard approach for partitioning the system among sub controllers [56], this research realises the use of expert-oriented distribution of workload intensity into various categories such as low, moderate and high. For each category, a system model is constructed, based on which a controller is designed. Whereas the final decision is carried out by the selection of a suitable controller at runtime using the FIS.

The proposed control method is responsible for the readjustment of the number of VMs to maintain the average *CPU utilisation* of hired VMs running at that time. We will refer *Cluster* to represent the running VMs at a time from this point onward. The proposed methodology incorporates three *Fixed gain* controllers termed *Lazy*, *Moderate* and *Aggressive*. In theory, the number of controllers depends on the adaptation and application scenario. Increasing the number of controllers facilitates more fine-grained control over cloud resources, however, it also increases the design and complexity of the elastic method. Each controller depicts a different elasticity policy, and theoretically they can be implemented using any suitable technique. However, this thesis focuses on the utilisation of three controllers to demonstrate the effectiveness of the proposed methodology, and we consider the types of controllers as *Fixed gain*.

The incorporation of fixed controllers with switching ability enables the adaptive behaviour of the system to respond appropriately to the needs of the system in case of changes in workload without the need of any on-line learning algorithm. Each of the controllers is designed to react differently in the various situation. In this case, as their name specifies, they indicate three different scenarios, i.e. to perform scaling action at slower, moderate and aggressive levels of intensity. The selection of one of this policy depends on the behaviour of the system at that point in time. The behaviour of the system is identify using the latest status of the following three aspects including application performance, workload arrivals, and resource utilisation. These aspects are represented as *Response time*, *Arrival rate* and *Control error* respectively in Figure 4.1.

The *System Monitor* component of the proposed methodology is responsible for obtaining the latest status of the three parameters mentioned above. These measurements (as shown in Figure 4.1) are provided to the FIS. The FIS then decides using the collection of elastic fuzzy rules (Section 4.5), what level of intensity is needed for the readjustment of resources (VMs) to meet the desired performance objective (explained in Section 4.5.2.1). The output of the FIS is one of the employed controller. The controllers are responsible for making the scaling decisions.

The simultaneous use of multiple policies in the existing cloud elasticity research, apart from the related methods discussed in Section 3.3.3.2, exists as follows:

1. Combination of reactive/predictive: This class of methodologies include the use of different triggering behaviour, i.e. *Reactive* and *Predictive* for different scaling actions. A commonly followed pattern is to use a *Predictive* approach to perform a *Scale-up* operation, whereas a *Reactive* approach is used for the *Scale-down* operation [128, 40].

2. Correction of prediction error: This class of methods include the use of a *Predictive* approach to take a scaling action in advance for a longer period. Whereas a *Reactive* approach used for the correction of any prediction error, and to handle unpredictable changes [159, 160, 161, 162].

Other methodologies include like [21], where the *Scale-up* action follows an aggressive approach by acquiring more than required resources to avoid performance degradation followed by the *Scale-down* actions to release the unwanted resources, if required.

In contrast to the classes of methodologies mentioned above, the proposed method consists of multiple policies, and they are independent of scaling action or trigger type association. Moreover, we aim to acquire the exact amount of resources, i.e. sufficient to satisfy the demands rather than buy more than required. This helps to avoid over-provisioning as described in Section 2.2.3. Further details on the components of the proposed framework are provided in the following sections.

4.4 FEEDBACK CONTROL

This section discusses the design and development of the feedback control part of our proposed methodology. We follow the process flow proposed by Antonio et al. [6] for the design of control system. The diagrammatic representation of this process flow is shown in Figure 4.2 (borrowed from [6]), which consists of six steps distributed in three levels. The first level involves defining the goal of the control methodology followed by the identification of control input and devising of system's model at the second level. Finally, the development, deployment and evaluation of the control methodology are part of the last level. The details of the control system goal, control input, system model and control design of our proposed methodology are provided in the following subsections. Whereas, the deployment and evaluation are discussed in Chapter 5.

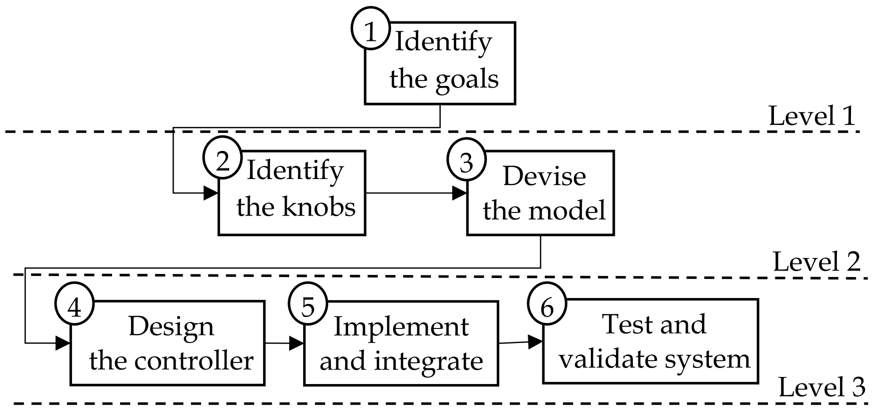


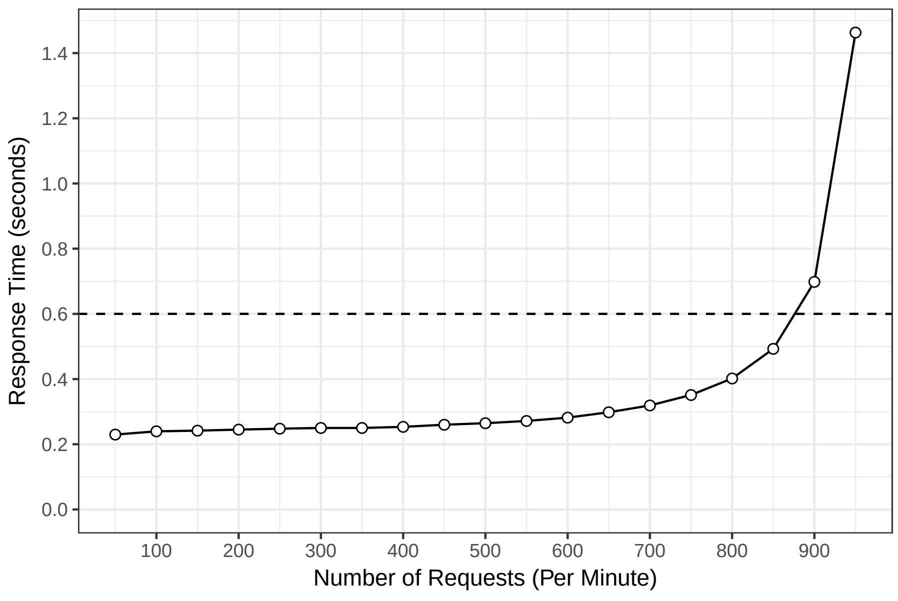
Figure 4.2: Design and development process steps of a feedback control methodology [6]

4.4.1 *Goal of Control Methodology*

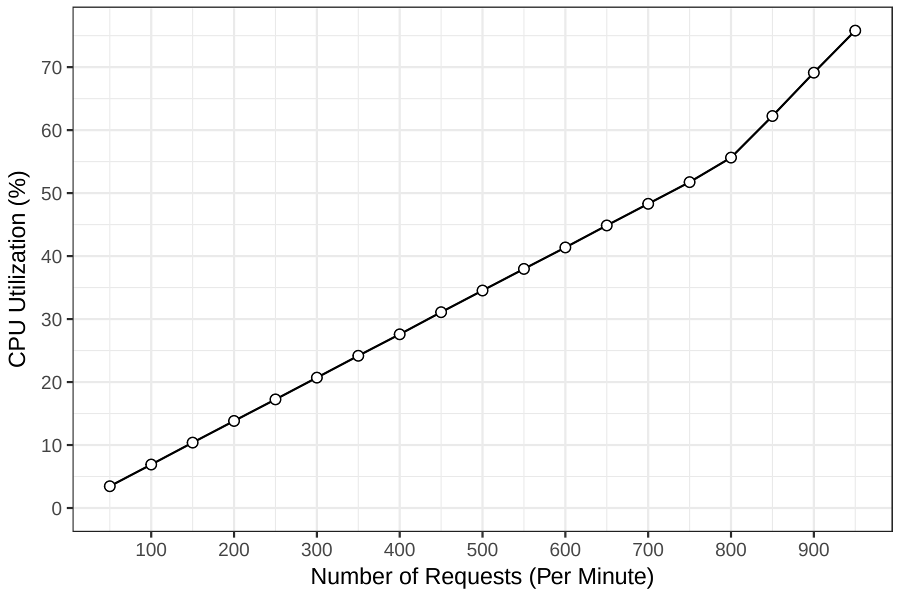
The goal of the proposed method is to adjust the number of VMs dynamically in response to changes in workload to maintain the CPU utilisation of the *Cluster* at a desired reference value, which reflects the target performance level. This description indicates that the CPU utilisation is the *Measured output* of the system and we have to identify the *Reference input*, i.e. the target CPU utilisation, which results in achieving the desired performance level. We consider mRT as the QoS parameter to measure the application performance and accept a value of (mRT $\leqslant$ 0.6 seconds) as the desired performance measurement. Thus, we need to identify the corresponding CPU utilisation level that maintains the mRT $\leqslant$ 0.6 seconds.

The key reasons for using CPU utilisation as the system output are the following: (1) The CPU utilisation is directly obtained from the monitoring API provided by CPs. Hence it does not require application level monitoring or efforts. (2) It is a system specific metric and no runtime relation identification between application metric, e.g. *Response time*, is required. Hence it does not involve additional overhead at runtime. (3) More importantly with respect to our methodology, we have already catered application level metric (i.e. Response time) for decision-making. Thus using CPU utilisation as another metric strengthens the decision-making mechanism by taking into account the system's resource utilisation perspective. Hence, the proposed methodology becomes hybrid in contrast to most of the existing methods that either rely on application or system level metrics.

The measurement for *Reference CPU utilisation* can be obtained using SID experiments (explained briefly in Section 3.2.1.1) by establishing a relationship between VM CPU utilisation versus performance. This experiment and all other such SID experiments are conducted using an extended version of a well-known cloud simulation tool named CloudSim [55]. A brief description of this simulation environment is provided in Section 5.1.



(a) Workload requests vs Response time



(b) Workload requests vs CPU Utilisation

Figure 4.3: VM performance

The SID experiment records the measurement of CPU utilisation and mRT against several workloads that differentiate regarding the number of incoming requests ranging from 50 rpm to 950 rpm. Each measurement of CPU utilisation and mRT against the specified rpm is obtained from sub experiment, where the corresponding number of rpm are sent for 30 minutes to the system, which consists of one VM. The arrival time of job requests in a minute and the service time of each request is randomly assigned. This whole experiment is repeated 100 times and the average for each measurement is recorded. The obtained results are presented in Figure 4.3.

It is evident from Figure 4.3a that the increase in the number of rpm makes the mRT slower. The dashed line in 4.3a represents the desired performance measurement, and we are interested in the maximum rpm measurement for which the obtained performance is less than the desired target, i.e. (mRT $\leqslant$ 0.6 seconds). This criterion is satisfied by 850 rpm. However, in this case, there were 13% SLO violations observed, which is not acceptable as per the employed performance objective (explain in Section 4.5.2.1). Therefore, we do not select the 850 rpm and consider the next measurement (i.e. 800 rpm), which satisfies the criterion mentioned earlier. This means that on average one VM can fulfil maximum 800 rpm on a per minute basis, while obtaining the desired performance level. Analogously to mRT, the number of rpm has similar effect on CPU utilisation, i.e. the increase in rpm results in an increase in the CPU utilisation as well. For the *Reference input*, we record the corresponding measurement of CPU utilisation from Figure 4.3b against the 800 rpm, which is 55%. Thus the control methodology is responsible to maintain the measurement of 55% as the *Reference CPU utilisation*.

4.4.2 *Control Input*

We use the number of VMs as the Control input. This choice is obvious considering horizontal elasticity and is also evident from the review conducted in Chapter 3, where all of the existing horizontal auto-scaling approaches use the number of VMs as the Control input. Moreover, we also performed an experiment to demonstrate the impact on mRT with a change in *Cluster* size. Figure 4.4 shows the obtained results from this experiment, which demonstrates that the increase in the number of VMs reduces the system's mRT.

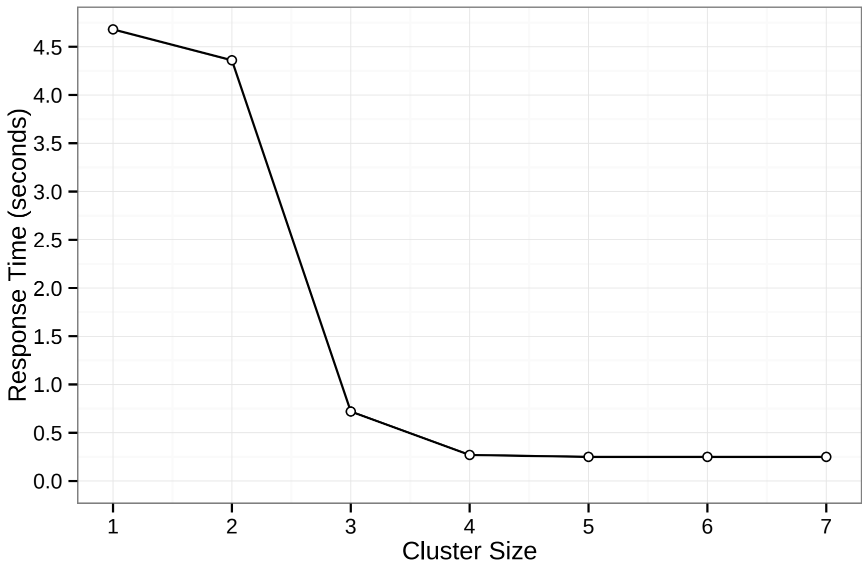


Figure 4.4: mRT vs cluster size with constant workload (2800 rpm)

4.4.3 *System Modelling*

This section identifies the relationship between input (number of VMs) and output (CPU utilisation) of the system. Such a relationship refers to the system model that describes how a change in input effect the corresponding output of the system. We followed the black box modelling approach (briefly explained in Section 3.2.1.1) to identify the system model. The black-box approach consists mainly of the following three steps to derive the model [191]: (1) designing and executing SID experiments to obtain training data; (2) building a model through the use of training data; and (3) validating the model. If the model is not valid as per the validation process, then the above steps are repeated to obtain another set of training data and different model. The following subsections describe the details of the steps carried out to obtain the model:

4.4.3.1 *SID Experiments Design*

The SID experiments record the training data consisting of the pairs of input and output of the system by changing the control input in a systematic way during the experiment. The design of such an experiment ought to take into account the following three considerations [53]: (1) Range of control input values; (2) Coverage of values; and (3) Richness that result in exciting system's dynamics. Considering these guidelines and following the recommendations provided in [53] on experimental design related to the adopted model type, i.e. 1^{st} order Autoregressive Exogenous Model (ARX) model (explain in Section 4.4.3.2), this thesis implements a discrete Sine wave equation to change the value of control input during an experiment.

The Sine wave consists of the specification of the following three ingredients: the time period, mean and amplitude. It satisfies the three considerations mentioned earlier as it changes the input signal within a defined range in an oscillatory style, starting from the mean, covering all possible values within the range. The coverage of the input values are evident from Figures 4.5a, 4.5c and 4.5e and the corresponding variation of changes in system's output are provided in Figures 4.5b, 4.5d and 4.5f respectively.

In the following part of this section, we explain the experiments conducted for recording training data. However, it is noted that we assume that the historical information related to system workload is available and on that basis, we use domain experts based distribution of workload into three categories namely *Low*, *Moderate* and *High*. Using these categories and following principles of *Gain scheduling* technique where workload-specific models are developed [53], we conducted three workload category specific experiments. Each experiment implements the following sine wave equation:

$$y(t) = m + A * sin(t) \tag{4.1}$$

In the above equation, m represents mean, A represents amplitude, where the time period for each experiment is 540 minutes long. The difference between each experiment is the use of the different pair of (mean, amplitude) values and the use of different workload. The settings of each experiment are provided below:

- Experiment 1: $mean = 18$, $amplitude = 15$ and workload specification is $rpm = 14400$
- Experiment 2: $mean = 37$, $amplitude = 20$ and workload $rpm = 29400$
- Experiment 3: $mean = 59$, $amplitude = 30$ and workload $rpm = 46800$

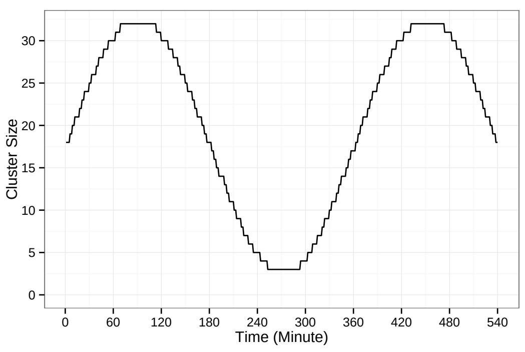
The workload in each of the above scenario is the average rpm for each respective category, and the mean is assigned the values (number of VMs) that roughly results in 55% CPU utilisation for each category respectively. The arrival time of jobs on a per minute basis and the service time of each request are randomly assigned. Each experiment is repeated 10 times, and the average results are recorded. These results are provided in Figure 4.5. The sub figures pairs (4.5a, 4.5b), (4.5c, 4.5d) and (4.5e, 4.5f) represents the input and output signals for *Experiment 1*, *Experiment 2* and *Experiment 3* respectively.

4.4.3.2 *System Model and Evaluation*

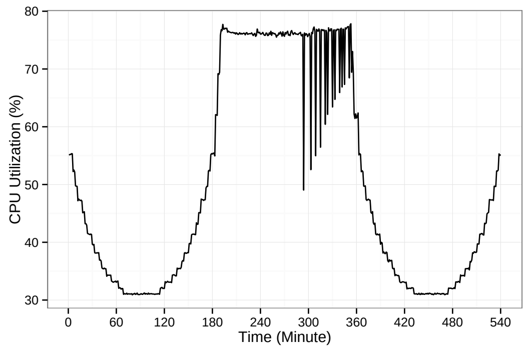
The ARX models are employed commonly for systems that use the black box modelling approach [53, 191]. This thesis also uses the ARX approach to describe the relationship between the number of VMs and CPU utilisation. The following equation (4.2) represents the general form of ARX.

$$y(k+1) = a_1 y(k) + ... + a_n y(k-n+1) + b_1 u(k) + ... + b_m u(k-m+1) \quad (4.2)$$

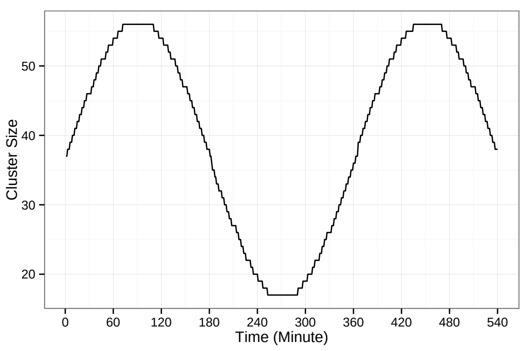
The equation above represents a single input, single output system. The u and y represent the input and output of the system respectively. According to this equation, the output in next time unit $(k+1)$ depends on the n number of previous output values and the m number of previous input values. The a_k and b_k are the constant coefficients values for each output and input value. Whereas



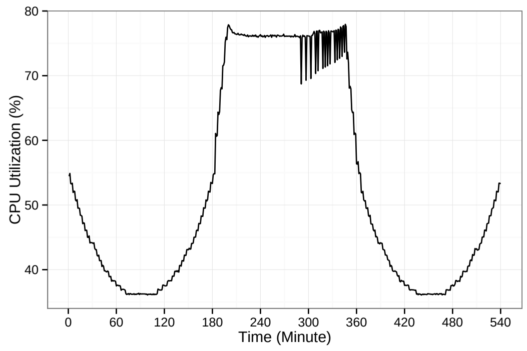
(a) Control input (Cluster size)



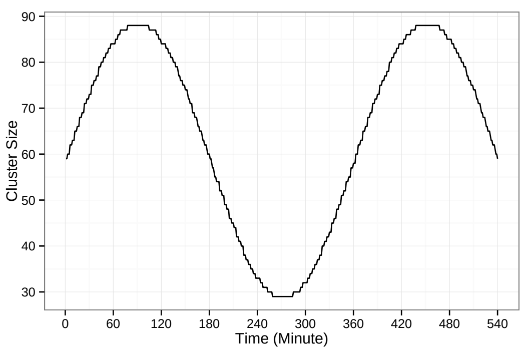
(b) Control output (CPU Utilisation)



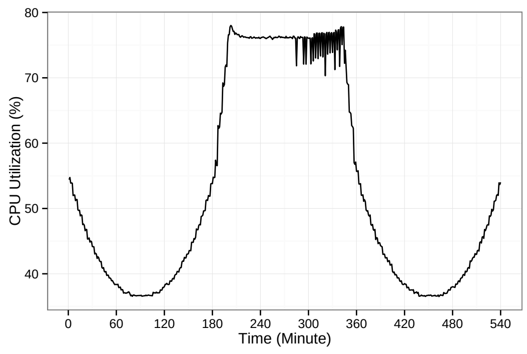
(c) Control input (Cluster size)



(d) Control output (CPU Utilisation)



(e) Control input (Cluster size)



(f) Control output (CPU Utilisation)

Figure 4.5: SID experiments: each pair presents the relationship between cluster size and CPU utilisation

the m and n represents the order of the model. We use a 1st order ARX model of the following form that can be derived from Equation 4.2 by setting $m = n = 1$.

$$y(k+1) = ay(k) + bu(k) \tag{4.3}$$

The 1st order model, in contrast to m & n order model, relies on the input & output from the previous time unit only. The key reason of selecting 1st order model is its simplistic nature and the ability to avoid over-fitting [53]. We have to find values for parameter a and b of the above equation from the training data obtained from the experimentation (described in the previous section). For this purpose, we employ the commonly used least square regression method to estimate the model parameters for all the three experiments mentioned in the previous sections, and the outcome is in the following equations:

$$y(k+1) = 0.89y(k) - 0.18u(k) \tag{4.4a}$$

$$y(k+1) = 0.93y(k) - 0.07u(k) \tag{4.4b}$$

$$y(k+1) = 0.95y(k) - 0.03u(k) \tag{4.4c}$$

These models after validation can be used to design controllers. The following two approaches can be followed to design controllers from these models. Firstly, each model could be used to design a different controller as they are obtained based on the average rate of each workload category and thus can be treated as workload-specific models. Secondly, one model could be used to design different controller where each differs from others based on the controller properties. We use the second approach because we do not only rely on workload arrival as criteria to differentiate between operating region but use the level of scaling action intensity required at run time to determine the suitability for a controller. We use the model of Equation 4.4a for the designing of controllers (explain in Section 4.4.4).

The next step is to evaluate the model. The purpose of the evaluation is to quantify the accuracy of the model. For this purpose, we employ a widely used method known as the coefficient of determination (denoted by R^2). The R^2 indicates the variability explained by the model [53]. R^2 can be calculated using the following equation,

$$R^2 = 1 - \frac{var(y - \hat{y})}{var(y)} \tag{4.5}$$

The y in above equation represents the actual recorded output value, where $\hat{y}$ indicates the predicted value computed using model. Thus, $var(y - \hat{y})$ is the variance of the difference between actual and predicted output, whereas $var(y)$ is the variance of the actual output. The value of R^2 lies in the range of 0 to 1. The magnitude of this value indicates the quality of the model. A higher value suggests better fit. The value of $R^2 \geqslant 0.8$ is an acceptable range [53]. In the case of the model in Equation 4.4a, the R^2 value is 0.96, which indicates a good fit. However, according to Hellerstein et al. [53], a larger value of R^2 can also be misleading in cases where data points are grouped together around extreme values. Therefore, to confirm the accuracy of the model, residual analysis plots are often recommended. Such a plot, in the context of our model, can be seen in Figure 4.6 where the actual values of the output signal are plotted against the predicted values. It is evident from this plot that apart from few points, all other points are grouped around the diagonal line, which indicates the better accuracy of the model.

4.4.4 *Controller Design*

The goal of the controller design step is to select the control law and any required parameters for the *Controller* component of the feedback control methodology. The control law determines the structure of *Controller* component and describes how it will operates [192]. In this thesis, we adopt the Integral control law for each of the three employed controllers, i.e. *Lazy*, *Moderate*, and *Aggressive*. The key reasons behind the Integral control law selection are its simplistic nature and its extensive use for similar problems, e.g. [31, 114, 125, 130, 156].

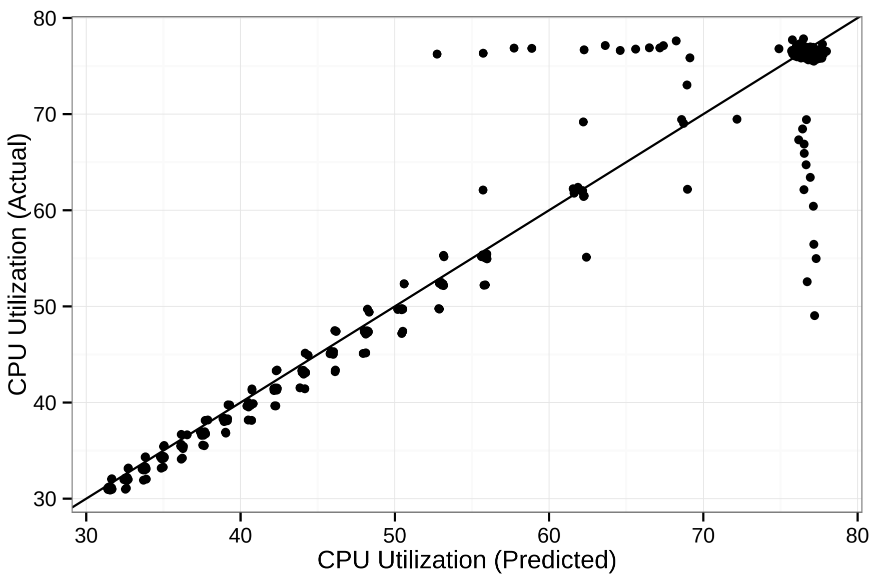


Figure 4.6: Comparison of actual (measured) output vs predicted output

The integral law can be defined using the following equation:

$$u(t) = u(t-1) + K_i e(t) \tag{4.6}$$

The $u(t)$ represents the new value for control input in time t, the $e(t)$ is the control error that represents the difference between the desired and measured output, i.e. $e(t) = y_{ref} - y_t$, and K_i is referred to the integral gain parameter. In the context of this thesis, the number of VMs is the control input, whereas CPU utilisation is the measured output. The control error represents the difference between the desired CPU utilisation (i.e. 55%) and the measured CPU utilisation.

The integral gain parameter indicate the aggressiveness of the controller that determines how fast the system will respond. The higher this value, the faster the system will react. However, careful attention is required while deciding the gain of the controller as higher value of the gain parameter could cause oscillation and may lead the system to instability. All the three employed controllers adopt the same integral law specified by Equation 4.6. However, their integral gain parameter is different. The following equations represent each employed controller:

$$u^{L}(t) = u(t-1) + K_{i}^{L} e(t) \tag{4.7}$$

$$u^{M}(t) = u(t-1) + K_{i}^{M} e(t) \tag{4.8}$$

$$u^{A}(t) = u(t-1) + K_{i}^{A} e(t) \tag{4.9}$$

The gains K_i^L, K_i^M, and K_i^A are derived using the standard procedure of Root-locus, which provides a systematic method to analyse and design feedback controllers. The Root-locus method require the transfer function of the feedback control system. Such a transfer function can be obtained by the corresponding transfer functions of the different components of the feedback loop. In our case, the different components include the integral controller (represented by Equation 4.6) and the target system (represented by one of the model earlier described in Section 4.4.3.2). The corresponding transfer function of integral controller is given in Equation 4.10, whereas the transfer function of the system model of Equation 4.4a is provided in Equation 4.11. Based on these equations, the transfer function of the entire feedback loop [53] is provided in Equation 4.12.

$$C(z) = \frac{zK_i}{z-1} \tag{4.10}$$

$$G(z) = \frac{0.18}{z-0.89} \tag{4.11}$$

$$F_R(z) = \frac{0.18K_i z}{z^2 + (0.18K_i - 1.89)z + 0.89} \tag{4.12}$$

Using the Root-locus method by taking into account the transfer function of feedback loop (Equation 4.12), we finalise the following values −0.06, −0.2, and −0.5 for K_i^L, K_i^M, and K_i^A gains respectively. The analysis performed using Root-locus indicate that the system remains stable (always reach to equilibrium) and accurate (steady-state error reach to zero) using all the selected gains. The finalised value has a settling time of less than 10 time interval, whereas, the maximum overshoot is recorded less than 15%.

4.5 THE SWITCHING MECHANISM: A FUZZY CONTROL SYSTEM

4.5.1 *Overview*

The deployed application over cloud environment automatically inherits, the uncertainty related challenges associated with the cloud environment [50]. Hence the underlying elastic method, which is responsible for the resource management of the application, has to deal with these challenges. However, handling uncertainty aspects in the existing elasticity research has not received much attention [47]. The uncertainty, in general, can be either *Stochastic* in nature or *Subjective* [193]. The *Stochastic* refers to the randomness in the environment such as the failure of VMs, and workload unpredictability. Whereas *Subjective* caused due to impreciseness in domain knowledge, e.g. lack of expertise to appropriately set optimal values for configuring a particular application. This section briefly discusses some of such uncertainty aspects in light of cloud elasticity. These aspects are summarised from the limited research works that highlight and address auto-scaling related uncertainty challenges [47, 186, 46, 50, 16]:

1. Impreciseness in domain knowledge: The design of an elastic policy needs a careful consideration by the domain experts to define various threshold values of the specific performance metrics. The elastic policy is thus depending on the specification of these thresholds values, hence the performance of policy remains subjective to the accuracy of experts knowledge, which is usually prone to impreciseness [47].

2. Noise in monitoring data: An elastic method has to rely on a monitoring component that continuously monitors the latest status of metrics needed for decision-making mechanism. This monitoring could involve gathering the following information: (1) the status of application-level metrics such as Response Time, and Throughput using the application or 3rd party sensor; (2) the status of resource utilisation such as CPU utilisation, memory consumption using cloud provider specific sensors. In both cases, the collected

monitoring data is associated with sensory noise [46]. For example, the hosted application may return different *Response Time* values at different times, or the cloud sensor returns slightly inaccurate estimate of resource utilisation measurement.

3. Inaccuracies in performance model: A performance model interprets the behaviour of the underlying system. The elastic controller as specified earlier can be designed using the performance model, which is subject to inaccuracies. Specifically, in the cloud environment, the VMs of one application may share the underlying physical resources with the VMs of other applications. Such phenomenon can lead significant variation in the performance of VMs [194, 46], which usually not considered at the time of designing performance models.

4. Delay caused due to actuator operation: The elastic controller takes a scaling decision, however, its execution is not immediate. The elastic controller has to interact the cloud provider service to initiate the execution. This process may take up to several minutes before the new VMs are set-up and ready to accept the load of the application. Thus the delay between the decision and the actual execution could cause some problems for the underlying application [46].

5. Unpredictability in workload: As mentioned earlier, the workload behaviour changes at different times (Figures 1.1b and 1.1a) and it can be difficult to predict across time due to unexpected spikes [50].

Jamshidi et al [16, 46] and Farokhi et al [47] stressed the importance of the uncertainty aspects to be taken into consideration while designing the elastic controller. Otherwise, scaling decisions often result in unreliability as the available resources may fail to fulfil the requirements, or may not be cost-effective [47]. However, despite the importance, the implementation of uncertainty in the context of cloud elasticity has not yet been well received [47].

A step in this direction is the work of Jamshidi et al in [16], where they proposed a fuzzy control system. They focused on the following two issues: (1) The quantitative nature of the *Rules-based* method by introducing the idea of qualitative elasticity rules; and (2) The lack of consideration regarding uncertainty raise due to noise in monitoring input data. Their fuzzy controller introduces elasticity rules of the following nature:

IF *workload* IS *high* AND *responsetime* IS *slow* THEN add 2 VMs

The elasticity engine executes such rules at runtime and make decision, based on *Arrival rate* and *Response Time*. The output of their controller is the number of VMs to be added or removed. Their approach facilitates a dynamic response based on the aforementioned two parameters by making a scaling decision with different intensity level, and consequently it improve the static scaling issue of the Rule-based approaches. However, the output (number of VMs) are pre-defined range of constant integers, and these numbers are set-up based on the experiences of the experts rather than rely on a well-founded design approach. In contrast, the proposed approach rely on the systematic method of control theory to compute the number of VMs. Moreover, the proposed approach is hybrid in nature, i.e. it incorporates both the performance and capacity based metrics as opposed to their performance based approach only.

We compliment and extend the work of Jamshidi et al. [16], and aim to develop a fuzzy control system to implement the switching mechanism of the proposed framework. The key reasons of selecting fuzzy logic theory as an implementation method are two-fold. Firstly, It is well known for its use in scenarios, where the system knowledge is imprecise, uncertain and highly dynamic [51]. Secondly, it facilitates qualitative decision-making by designing the rule-based system thus enabling us to provision qualitative elasticity rules.

4.5.2 *The Design Process*

The design process of constructing a fuzzy system as described in Section 2.4.3 consists of the following three steps: establishing domain knowledge, designing membership functions and composing fuzzy rules. The details of each of these steps in the context of our switching mechanism are provided below.

4.5.2.1 *Domain Knowledge*

Domain knowledge is concerned with the identification of inputs and outputs of the system. The inputs specify those factors of the system which are important to be considered for decision-making purposes. Earlier in Section 4.3, we mentioned the consideration of the three aspects of the system for decision-making. These aspects are the inputs of the fuzzy system. Their brief description is provided below:

1. Response time: It indicates the performance level of the deployed application and is measured as the percentage number of SLO violations (i.e. when *Response time* of a job request > 0.6 seconds) in last time unit.
2. Arrival rate: It indicates the workload behaviour in the last time unit and is measured as percentage number of job arrivals. The *System Monitor* component of the proposed method records the number of arrivals in last time period to identify the intensity of the workload.
3. Control error: The inclusion of *Control error* as an input is the consideration of resource utilisation level into the decision-making. The *Control error* is the difference between the measured and desired CPU utilisation.

The inclusion of inputs mentioned above into the decision-making mechanism covers the following aspects of the system, i.e. performance, disturbance and resource utilisation. Such an elastic methodology is referred to as *Hybrid*, a termed coined by Faroki et al. [49] to represent methodologies in contrast to either capacity-based or performance-based approaches. The capacity-based methodologies rely on system level resource utilisation metrics such as CPU utilisation, and memory utilisation [167, 33, 110]. Whereas the performance-based methodologies rely only on application level metrics such as Response time, and Throughput [171, 16]. The output of the fuzzy system is one of the employed controller that will be used to compute the scaling decision. In contrast, the fuzzy controller of Jamshidi et al in [16], directly produce the pre-defined constant number of VMs that has to added or removed as the output. Whereas in our case, the scaling decision is computed by the controller using the *Control error* at that point in time.

The next step is to define fuzzy set for each input and output (commonly known as fuzzy variables). The fuzzy set of each variable comprises of defining linguistic terms and assigns ranges of values to them. Table 4.1 provides the definitions of all the linguistic terms for each fuzzy variable and their corresponding ranges, whereas their brief description is as following:

- The linguistic terms and the corresponding ranges for the *Workload* variable are adopted from the work of Jamshidi et al in [16], where the knowledge base is constructed using domain experts, i.e. architects and administrators. They have constructed a fuzzy set of five linguistic terms for *Workload* variable including *Very low*, *Low*, *Medium*, *High* and *Very high*. We reduce them to three to minimise the number of rules, hence reduce the complexity. However, more fine-grained control over resources can be obtained by increasing the number of workload categories or the number of controllers.
- The linguistic terms of *Response time* variable reflect the overall performance objective of the application that can be defined by the SPs. In Table 4.1, we use symbols β_1, β_2, β_3 and β_4 to represent the customisable aspect of these parameters. Jamshidi et al. [16] in contrast, distributes the *Response time* into five categories with the values obtained from domain experts. However, considering that the application performance measurement for different application is different and therefore, the values of the linguistic terms of

Response time are customisable that reflect the desired performance objective and has to be defined by the SPs. In the current settings of this thesis, we adopted the following values for evaluation purposes, i.e. $\beta_1 = 3\%$, $\beta_2 = 5\%$, $\beta_3 = 8\%$, and $\beta_4 = 10\%$.

- The linguistic terms of *Control error* are obtained by distributing the *Control error* measurement into five categories. The increase in these categories can provide more fine-grained control. However, it will also increase the complexity of the proposed method. The ranges of these linguistic terms are obtained using trial and error method, where various experiments were carried out using different ranges.
- The linguistic terms of *Controller* variable are the possible outcomes. These terms depend on the number of controllers, which in this case are three. Moreover, we also consider one more output, i.e. *No-scaling*, which specifies that none of the controllers are selected and no scaling decision is required. The ranges of these linguistic terms are set based on the approach adopted in [54], where no overlapping of the range is required because the final decision represents a range that indicates a single output rather than a numerical value.

4.5.2.2 *Membership Functions*

The next step is to define the membership functions that convert the crisp inputs into the corresponding fuzzy values. The membership function defines the degree of the crisp input against its linguistic variables in the range of 0 to 1. In this thesis, we design the membership functions following the approach adopted in [16], where they have used only triangular and trapezoidal types of function for their construction. The triangular and trapezoid functions have the advantage of being simple and efficient in comparison with other types of membership functions [195]. Figure 4.7 represents the membership functions of our fuzzy control system.

Fuzzy variable	Set member	Range
Workload(Arrival Rate)	Low	0 — 48.9
	Medium	30.7 — 67.94
	High	56.41 — 100
Response time	Desirable	0 — β_2
	Okay	β_1 — β_3
	Bad	β_4 — 100
Control error	Stronger Negative (stNeg)	-20 — -100
	Weaker Negative (weNeg)	-5 — -30
	Normal	-10 — 10
	Weaker Positive (wePos)	5 — 30
	Stronger Positive (stPos)	20 — 100
Controller	No scaling	0 — 10
	Lazy	11 — 20
	Moderate	21 — 30
	Aggressive	31 — 40

Table 4.1: Ranges for fuzzy variables

4.5.2.3 *Fuzzy Rules*

The fuzzy rules describe the relationship between the inputs and outputs of the fuzzy control system. The fuzzy rules, in this case, determine the type of the controller that will be used to make the elastic decision. The fuzzy rules are in the form of *if-then* and are made of using fuzzy logic statements. The *if* part of the rule refers to antecedent and the *then* part is called the consequent. The fuzzy rules of the switching mechanism are made using the linguistic terms of the fuzzy variables explained earlier in Section 4.5.2.1. An example of such a rule can be seen from Figure 4.8. In this example rule, a *Lazy* controller is selected based on the values of *Arrival rate*, *Response time* and *Control error*. Such rules for an application scenario can be designed using the combination of linguistic terms

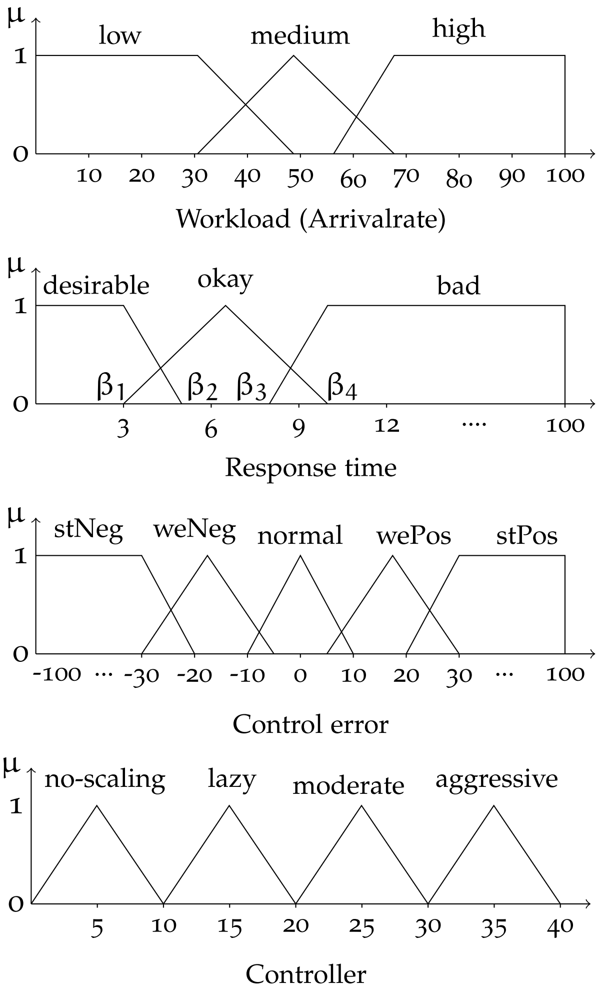


Figure 4.7: Membership functions

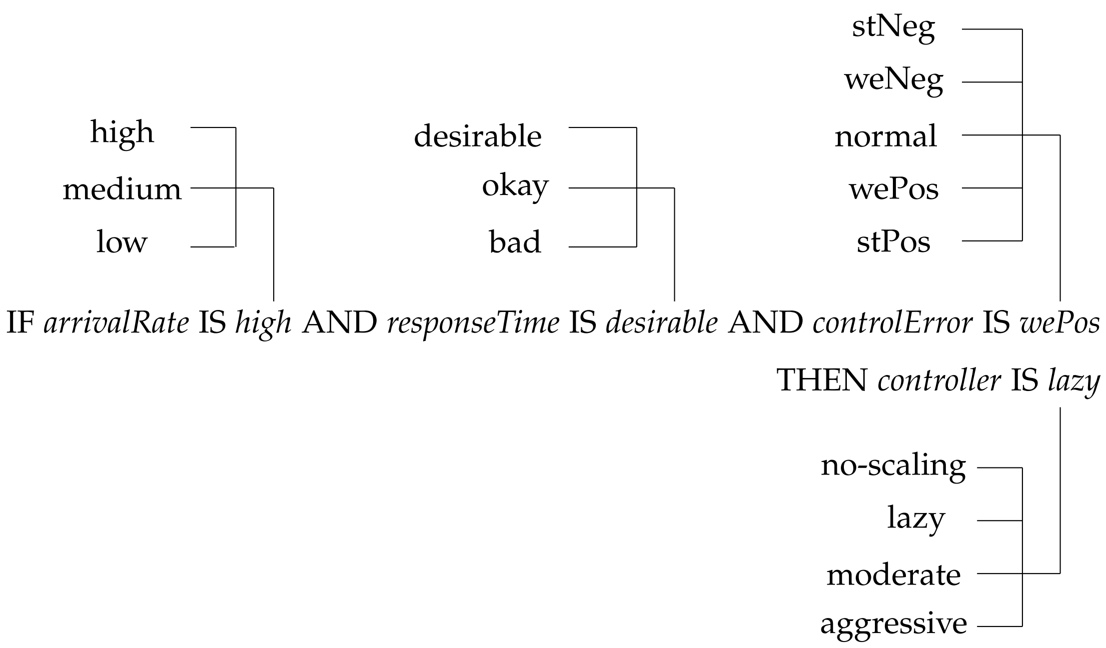

stNeg
weNeg
high
medium
low
desirable
okay
bad
normal
wePos
stPos
IF *arrivalRate* IS *high* AND *responseTime* IS *desirable* AND *controlError* IS *wePos*
THEN *controller* IS *lazy*
no-scaling
lazy
moderate
aggressive

Figure 4.8: Example of switching elastic rule

provided for each parameter in the rule (or see Table 4.1). Such rules can also be tuned for different situations using optimisation approaches. The full list of the rules employed for the experimentation conducted in this thesis are provided in Table 4.2. These rules are designed using the following considerations: (1) Select those rules, which can react quickly if the application performance is poor; (2) If the application performance is desirable then aims to reduce cost; (3) Aim to maintain the CPU utilisation around the desired reference value.

Workload	Response time	Control error	No-scaling	Lazy	Moderate	Aggressive
high	desirable	wePos		✓		
high	desirable	stPos			✓	
medium	desirable	wePos			✓	
medium	desirable	stPos				✓
low	desirable	wePos			✓	
high	desirable	stPos				✓
high	desirable	weNeg		✓		
high	desirable	stNeg			✓	
high	okay	weNeg				✓
high	okay	stNeg				✓
high	bad	weNeg				✓
high	bad	stNeg				✓
medium	desirable	weNeg		✓		
medium	desirable	stNeg			✓	
medium	okay	weNeg				✓
medium	okay	stNeg				✓
medium	bad	weNeg				✓
medium	bad	stNeg				✓
low	desirable	weNeg		✓		
low	desirable	stNeg			✓	
low	okay	weNeg		✓		
low	okay	stNeg			✓	
low	bad	weNeg				✓
low	bad	stNeg				✓
-	-	normal	✓			

Table 4.2: All switching elasticity rules

4.6 EXECUTION FLOW SUMMARY

Sections 4.4 and 4.5 explain the key components of the proposed framework. This section briefly summarises the execution flow of the framework explaining how the proposed framework works as a unit at each iteration.

1. The *System Monitor* component of the framework is responsible for gathering the latest status of the performance metrics. This monitoring include, the incoming workload, measuring the application performance and collecting the CPU utilisation measurement from cloud provided sensors, e.g. Cloud watch service provided by Amazon.

2. The FIS obtains the latest measurements of inputs from *System Monitor* component.

3. The input values are converted to the corresponding fuzzy values using the defined membership functions.

4. The FIS then evaluates the rules and identifies the output, i.e. *Controller*.

5. The *Switch* component of the framework then enables only the selected controller to compute the number of VMs to be added or removed.

6. The target system (i.e. elastic application) then executes the scaling decision by triggering the cloud provided API to add or remove the VMs, computed in the previous steps.

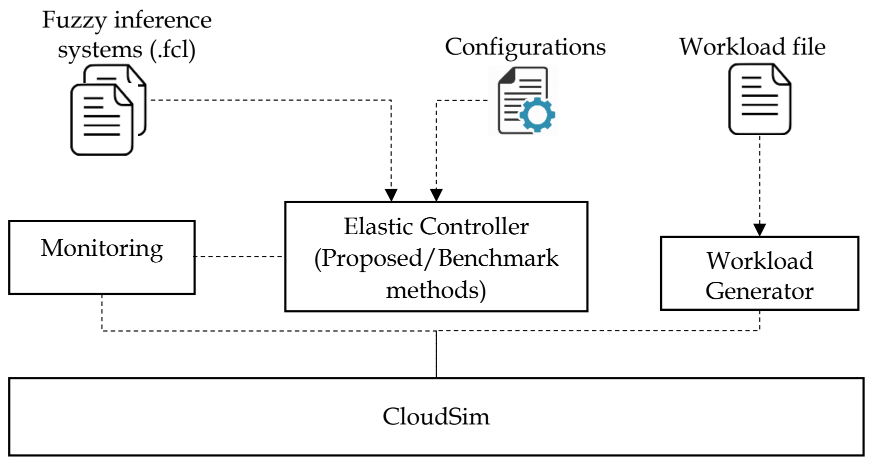
5 EXPERIMENTATION AND PERFORMANCE EVALUATION

This chapter presents and discusses the experimentation carried out for the evaluation of *Hard switching* framework. Section 5.1 introduces the self-customised experimental environment that integrates a well-known cloud simulation environment called *CloudSim* and an external Java-based library called *JFuzzyLogic*. The details regarding the various workload patterns and the real HTTP traces employed for evaluation purposes are provided in Section 5.2. The benchmark methods and the employed evaluation criteria are presented in Section 5.3 and 5.4 respectively. Finally, the obtained computational results are presented and discussed in Section 5.5.

5.1 EXPERIMENTAL SET-UP

The experimental environment used for the evaluation of *Hard switching* framework is developed in Java language. This experimental environment integrates a well-known cloud simulation framework called *CloudSim* [55] and an external Java-based library called *JFuzzyLogic* [196]. The block diagram of this experimental set-up is shown in Figure 5.1, which consists of the following four key components. This section provides an overview of these components and further explanation of how it works collectively:

1. Workload Generator: It is responsible to read the *Workload file* containing details of the incoming job requests. Each job request is uniquely identified with a *Job ID*, and its arrival time and required service time are noted. The *Workload Generator* creates a particular Java-based object to represent each job request and passes it on to the *CloudSim* (explain shortly in this section)


Fuzzy inference systems (.fcl)
Configurations
Workload file
Monitoring
Elastic Controller (Proposed/Benchmark methods)
Workload Generator
CloudSim
Figure 5.1: Architectural view of the experimental set-up

at its specified arrival time. The *CloudSim* is responsible for the execution of the job.

2. Elastic Controller: The *Elastic Controller* component is the implementation of our proposed elastic method and all other benchmark methods (see Section 5.3). Various experimental settings required to run the elastic methods are provided at runtime using a configuration file. This file contains all the parameter settings of the experiment such as the number of initial VMs, which method to execute and desired *Response time*, etc.

3. Monitoring: The *Monitoring* component is responsible to gather the latest information of the employed metrics. This includes the monitoring of the *Arrival rate* and *Response time*. Whereas, the CPU utilisation is obtained from the *CloudSim*.

4. CloudSim: A key component of this experimental set-up is the use of CloudSim [55]. *CloudSim* is extensively used in the cloud related research activities for modelling and simulation of cloud computing systems and applications [8, 197, 198, 199]. For the sake of these experiments we have extended *CloudSim* with features of elasticity and other related concepts such as load balancing strategies and necessary features for SID experiments.

CloudSim provides some important entities that can be extended to simulate the real world cloud objects. In this research, however, we have focused on the following four of them because of their importance and usage in our experiments. These include the *Data centre*, *Host*, *VM* and the *Cloudlet*. The *Data centre* entity depicts the Infrastructure as a Service (IaaS) layer, which manages the *Host* entities. A *Host* entity represents a physical server, which can be possibly shared by several VMs. The *VM* reflects a virtual machine, hosted by one of the *Host*. The *Cloudlets* simulates an individual task unit or a job with a pre-defined processing length (service time) measured in *MIPS* (millions of instruction per second) and an arrival time.

CloudSim allows the creation and deletion of VMs, which are used as a result of the scaling decision. Once a VM is created, it must be assigned to a *Host*. This assignment is governed by an allocation policy. Each host also implements a scheduling policy that determines how the assigned VM must share the available resources. These policies must be defined before running the simulation. Such configurations are related to the allocation and scheduling of the related problems (see for details [9]) and are beyond the scope of this thesis. Therefore, only the default allocation and scheduling policies concerning *VM* and *Host* related assignment and execution are used in this research.

Analogous to the *VM* and *Host* related allocation and scheduling policies, there are some other application level allocation and scheduling policies that control the assignment of *Cloudlets* to the already available VMs. For this purpose, we have implemented a round robin policy to assigned the incoming jobs (i.e. *Cloudlets*) to the available VMs. In the case of a *Scale-down* operation, the *VMs* with the minimum number of jobs are deleted.

The flow of an experiment using the environment mentioned above is as follows: A specific *Workload file* and *Configuration settings* represent one experiment, which are provided to the environment. The *Workload generator* reads the *Workload file*, transforms each job request to *Cloudlets* and keeps sending them to *CloudSim* at their pre-specified arrival times. The *CloudsSim* executes each job as per their service time requirements. The *Monitoring* component gathers the performance metrics information at different time intervals during the entire experiment and feed them into the *Elastic controller* component. The scaling decisions, if required, are performed by the *Elastic controller* and execute using *CloudSim* API.

5.2 WORKLOADS

The commonly used approach to test an auto-scaling methodology is to evaluate its performance based on certain desirable criteria against different workloads. The existing research on cloud elasticity [122, 123, 40, 49, 21] have made used of various publicly available internet traces, e.g. Wikipedia, FIFA world cup, etc. as input workloads. Moreover, the use of synthetically generated workloads based on different patterns is also quite common [45]. The various workloads used in the existing research are shown in the Tables 3.1 to 3.7.

It is quite common to use various real workloads to evaluate an elastic method. However, Gandhi et al. in [200] and Jamshidi et al. in [16] evaluated their proposed elastic methods by using workloads that follow different patterns. The key reason of using such an approach is to evaluate and analyse the performance of an elastic method in different scenarios. The workload patterns that they have used include *Quickly varying*, *Slowly varying*, *Dual-phase*, *Tri-phase*, *Big spike* and *Large variations*. Similarly, Mao and Humphrey [201] used *Stable*, *Cyclic*, *Growing* and *On-off* set of patterns. Each of these patterns represents a different class of applications [45].

This research also adopts the patterns mentioned above to analyse the performance of the proposed method regarding different classes of applications. We have used seven different workloads as shown in Figure 5.2. Each of these workloads represents a single or multiple patterns. Amongst these workloads, six are real and one of them is synthetically generated. The real workloads are derived from the following Internet-based sources, i.e. *Wikipedia*, *FIFA World Cup* and *WITS*. One of them, however, is generated synthetically. A brief description of each of these sources and the details of the corresponding workloads are provided below:

- Wikipedia: The *Wikipedia* has made available its entire HTTP traces target to their servers from *September 2007* to *January 2008* in [202]. The *Wikipedia* traces as input workload scenarios are amongst the most used for the evaluation of elastic methods. Some prominent examples of the relevant research undertaken that utilised *Wikipedia* traces include [122, 49, 21]. We have extracted the following two workload scenarios using *Wikipedia* traces: (1) The three days (starting from 18th September to 20th September) is horizontally scaled to one day long. The derived trace is shown in Figure 5.2e and is an example of the *Cyclic* pattern, and (2) The second workload consists of the original one day (18th September) trace. This trace is shown in Figure 5.2f and is an example of a *Slowly varying* workload pattern.

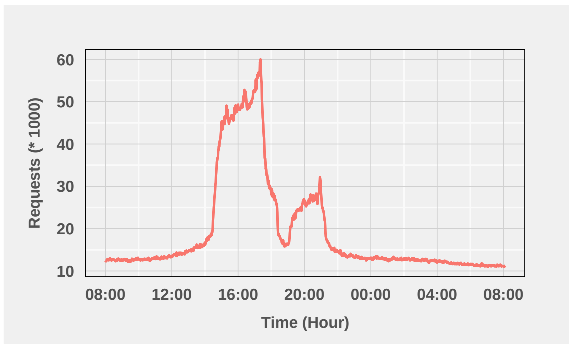
- FIFA World Cup: This consist of all the HTTP requests targeted to the website of *1998 FIFA World Cup* from 30th April 1998 to 26th July 1998 and is available in [22]. This workload is shown in Figure 1.1b. The FIFA trace is one of the mostly used workloads for testing the elastic methods, e.g. [33, 122, 123, 124]. We have, however, derived a one day trace (starting from 08 AM, 03rd July to 08 AM, 04th July) as an example of the *Large variation* and *Big spike* pattern. The derived trace is shown in Figure 5.2a.

- WITS traces: The WITS (Waikato Internet Traffic Storage) [203] project has made available a large range of different sets of Internet traces, which are freely accessible for research purposes. Amongst these traces, we have derived the following three scenarios, i.e. *Quickly varying*, *Dual-phase* and *Tri-phase*. The derived workloads are shown in Figures 5.2b, 5.2c and 5.2d respectively. Some of these traces are also used in Gandhi et al. [200].

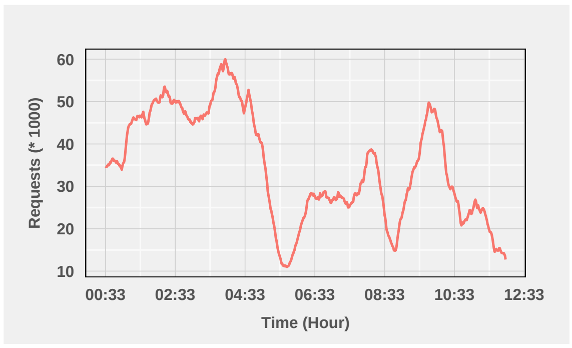
- Synthetic: We have also generated one trace synthetically to represent the *On-off* pattern. The generated trace is shown in Figure 5.2g. Mao and

Humphrey [201] used a similar workload pattern as well. The synthetic generation of this workload trace consists of 6 hours long on and off pattern, where the number of arrivals per minute slightly varied in each pattern. The variation in case the of off pattern time is between 10,000 rpm to 12,000 rpm, whereas in the case of on pattern, it was 41,000 rpm to 44,000 rpm.

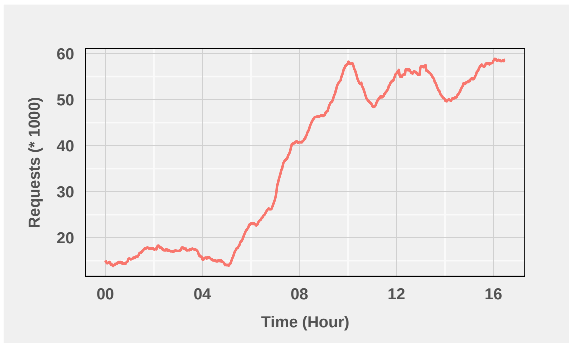
All the workloads mentioned above are vertically scaled (up or down) to a maximum of 60,000 rpm. The number of arrivals on per minute basis is obtained from the count of actual arrivals except for the synthetically generated one. The arrival time of each request in a minute is randomly generated for the creation of *Workload files* as mentioned in Section 5.1. Moreover, the service time of each job request is randomly generated between 100 milliseconds to 500 milliseconds to incorporate the stochastic behaviour of the incoming arrivals whilst considering different kinds of requests, e.g. read-only, or read/write, etc.



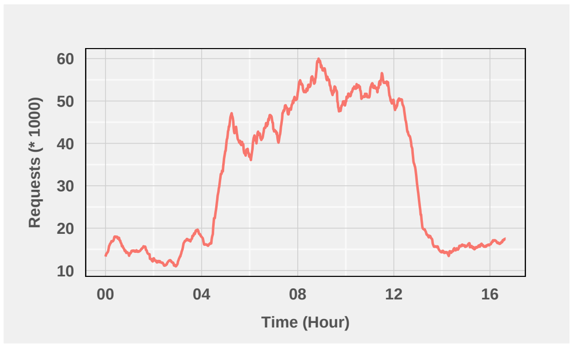
(a) Large variations/Big spike (FIFA Worldcup)



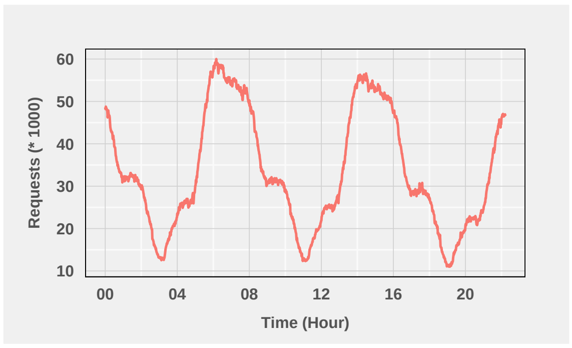
(b) Quickly varying (WITS_QV)



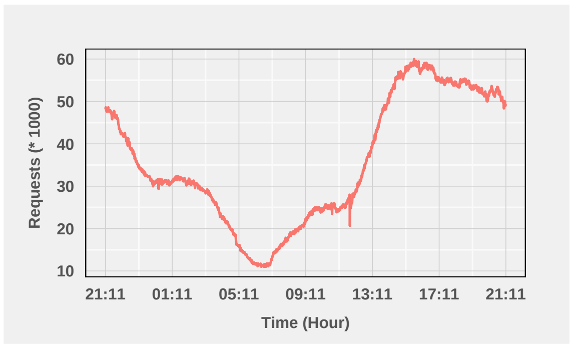
(c) Dual phase (WITS_DP)



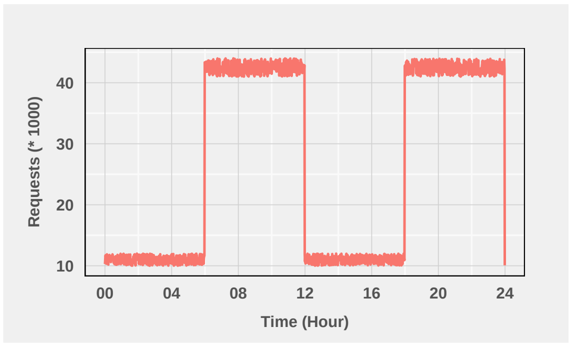
(d) Tri phase (WITS_TP)



(e) Cyclic (Wikipedia)



(f) Slowly varying (Wikipedia_2)



(g) On-off (Synthetic)

Figure 5.2: Various workloads used for experimentation

5.3 BENCHMARK APPROACHES/SCENARIOS

5.3.1 *Fixed gain Feedback Controller*

We have used *Fixed gain* controller as one of the benchmark methods. The general overview of feedback control as an auto-scaling method is described in Section 2.4.2, whereas Fixed gain controller is introduced in Section 3.2.1.1. The key reason of using *Fixed gain* controller as a benchmark approach is that our proposed method is an extension, where we have used multiple *Fixed gain* controllers simultaneously.

We consider the individual controllers termed *Lazy*, *Moderate* and *Aggressive* separately as individual elastic controller. We aim to demonstrate the effect of using the same controllers independently versus using them collectively as in the proposed framework. More generally, we aim to demonstrate the effects of using uniform elastic policy of different natures versus the effects of combining them with an intelligent switching mechanism.

The nature of the individual controllers, i.e. *Lazy*, *Moderate* and *Aggressive*, in general are similar to those used in related elastic methodologies reviewed in Section 3.3.1.1. The individual controllers are implemented following the proportional threshold approach in [31], where the *Reference input* is considered as a range rather than a scaler value, e.g. in our case it is 55%. This approach avoids the unnecessary oscillations by restricting the controller not to take a decision if the measured output is within a certain range. In our implementation, however, we consider a range $\pm 10\%$ of *Reference input*, which is the same as the range of *Normal* linguistic term of *Control error* fuzzy variable used in our proposed switching mechanism.

5.3.2 *RightScale: A Rule-based Approach*

This research also uses *RightScale* [27] as another benchmark method. The *RightScale* is a 3rd party commercially available auto-scaling approach, which is based on the principles of the *Rule-based* method described in Section 2.4.1. In the *RightScale* method, each individual *VM* engages in a voting process, where every VM decides whether a scaling decision is required or not. The decision by individual VMs is based on the set of elasticity rules.

The implementation of *RightScale* includes the setting of decision threshold value for the voting process, which determines whether the scaling decision will be carried out or not. For this purpose, we have used the value 51% representing that if just more than half of the VMs are in favour of the decision, the action will be performed. Otherwise, it will be ignored. Another important aspect of *RightScale* implementation includes the determination of system metric to be used for setting up the rules. For this purpose, we use *CPU utilisation* as a system metric based on its usage as the *Reference input* in the proposed method. The elasticity rules used for the implementation are as following:
For scale up

```
    if CPU Utilisation $> thr_{up}$ then
    $n = n + s_a$ and
    do nothing for t seconds
```

For scale down

```
    if CPU Utilisation $< thr_{down}$ then
    $n = n - s_r$ and
    do nothing for t seconds
```

The value use for thr_{up} is 55%, i.e. the desired *Reference input* of our proposed method as we already know, the performance degrades when *CPU utilisation* becomes higher than 55%. The value for thr_{down} obtained by trying different possible values such as (20%, 30% and 40%) and then selected, the value that produces the best result regarding the evaluation criteria (explain in next section). Another important configuration required is the settings of values for s_a and s_r. For this purpose, we use the following four different settings: (1) $s_a = s_r = 2$, (2) $s_a = 2, s_a = 1$, (3) $s_a = 4, s_r = 2$ and (4) $s_a = 10\%, s_r = 5\%$. Lastly, the t in both of the above rules specifies the cooldown period, which remains same for all the methods, i.e. for the proposed method as well as for the benchmark methods.

5.4 EVALUATION CRITERIA

The key objective of implementing cloud elasticity is to improve the utilisation of computational resources whilst maintaining the desired performance of the system and reducing its operational cost. This statement provides us with the fundamental criteria to assess the quality of an auto-scaling mechanism, which is *Performance* and *Cost*. This section briefly introduces these parameters in the context of this thesis to set the criteria for evaluation purpose.

1. SLO Violations: Section 2.3.2 provides an overview of performance-related concepts including the introduction of different aspects like *Availability*, *Throughput* and *Response time*. The measurements of these aspects represent the performance of the underlying system. Among these aspects, we consider *Response time* as a criterion to measure the performance of the elastic method. The requirements regarding desired performance objective in cloud computing is defined through SLO specification as explained in Section 2.3.2. In this thesis, we consider that each job request of the workload must be completed in the pre-defined desired time, i.e. $\leqslant 0.6$ seconds. Thus an SLO violation is considered, if the desired Response time for a job request has not been achieved.

2. Cost: The *Cost* refers to the operational cost of the rented VMs. These VMs are used to execute the workload and each VM is associated with a cost per time unit. The total running time of all VMs is recorded for the entire experiment.

It includes the time when a VM starts; to the time it finishes execution, either as a result of a *Scale-down* action or when the experiment finishes. The total time is calculated in minutes and an immediate start and stop of the VMs are considered to avoid complexities related to the implementation. The total running time of all the VMs is further converted to hours in the final calculation. A rate of 0.013$ per hour is applied to calculate the final cost based on the Amazon pricing [204] for the VM instances of "t2.micro" type.

Apart from the above mentioned main criteria of evaluation, we have also considered the number of unsuccessful requests in the entire experiments. The unsuccessful requests refer to the job requests that were unable to complete their execution in a maximum time unit. In such a case, we withdraw those job requests and discard them. For the evaluation purposes, we take 2 seconds as the maximum time unit.

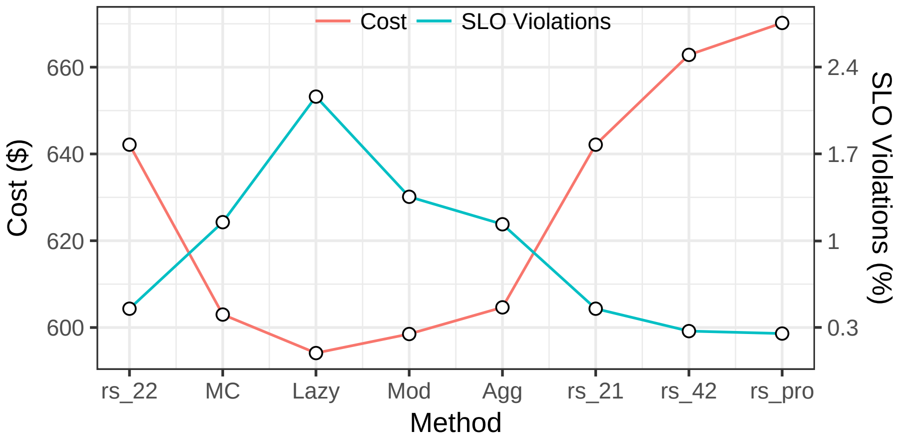
5.5 COMPUTATIONAL RESULTS AND ANALYSIS

The computational results obtained from the experiments carried for evaluation is presented in this section. These results are categorised based on the type of the workload pattern used. The following subsections discuss the results of each workload scenario, whereas subsection 5.5.2 provides a summary of the overall results.

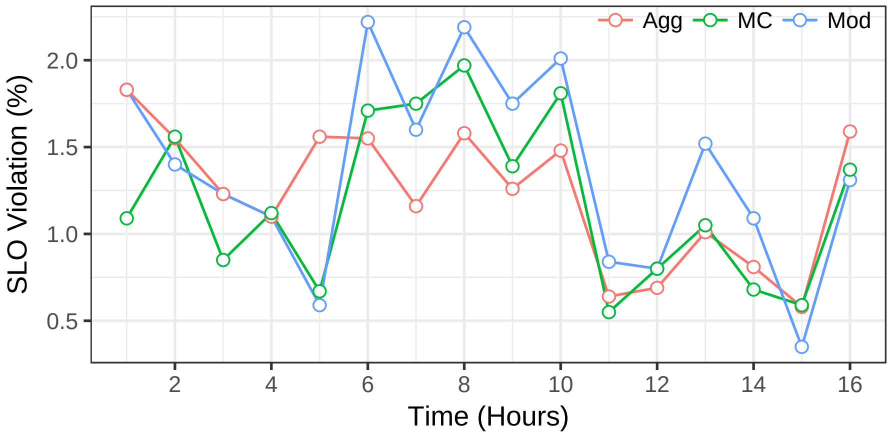
5.5.1 *Individualistic Scenario Analysis*

5.5.1.1 *Dual-phase*

Figure 5.3a presents an aggregated view of the *Cost* versus *Performance* aspect of the overall experiment for all methods, i.e. proposed and benchmark methods. In this figure, rs_{22}, rs_{21}, rs_{42} and rs_{pro} represent the four different settings of the *RightScale* method explained in Section 5.3.2. Whereas *Lazy*, *Mod* and *Agg*

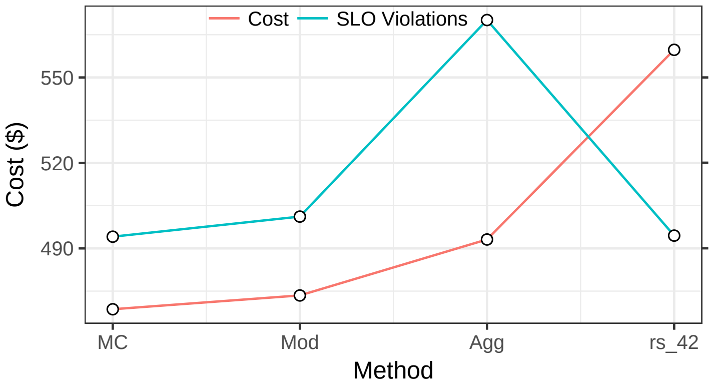


(a) Aggregated view of cost vs performance

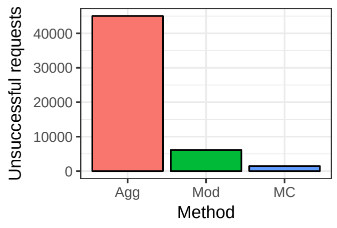


(b) Timeseries view of performance (SLO)

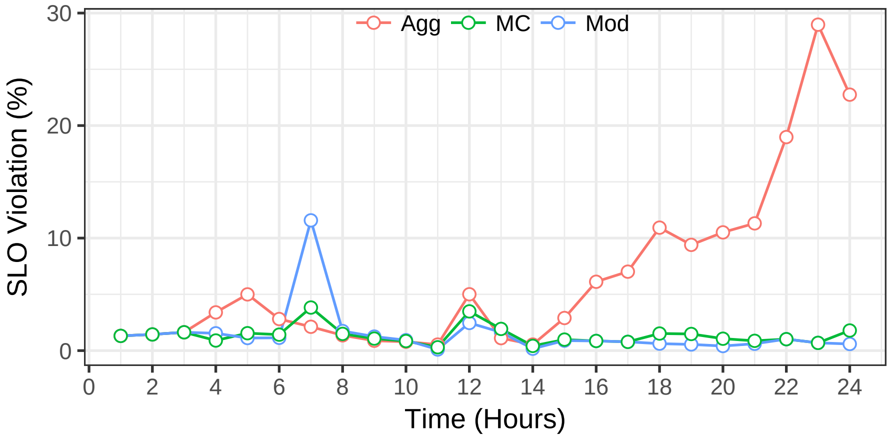
Figure 5.3: Results of *Dual-phase* scenario. The *RightScale* methods results in lowest number of SLO violations, however at a higher cost. Amongst the rest, *MC* comparatively performs better with lowest number of SLO violations, i.e. 1.14% in comparison of 1.28% and 1.33% of *Agg* and *Mod* respectively.



(a) Aggregated view of cost vs performance



(b) Unsuccesful requests



(c) Timeseries view of performance (SLO)

Figure 5.4: Results of *Large variation* scenario. The proposed *MC* approach (a) results in lowest number of SLO violations, (b) maintain less than 5% SLO violations in the entire time period in contrast to others where more than 10% are observed in different hours, and (c) results in lowest number of unsuccessful requests.

refers to the benchmark methods explained in Section 5.3.1 and *MC* represents the proposed *Hard switching* method.

It is evident from Figure 5.3a that rs_{22}, rs_{21}, rs_{42} and rs_{pro} produce better performance results, i.e. lowest number of SLO violations in comparison to all other methods, however at a higher cost, i.e. more than 640$ in contrast to ≈ 600$ of other methods. Amongst other methods, the *MC* comparatively performs better, where the exact measurements are 1.14% for *MC*, 1.28% for *Agg* and 1.33% for *Mod*. Regarding the unsuccessful requests, there was no case observed by any method.

Figure 5.3b shows the percentage number of SLO violations on a per hour basis obtained using *Agg*, *Mod* and *MC* methods. This plot demonstrates the following three aspects when observing in accordance to the *Dual-phase* workload (as shown in Figure 5.2c). (1) The first 5 hours of the plot indicates that *MC* results in lesser number of SLO violations than other methods. During these hours, due to low arrivals (almost stable), there was not much change required and therefore, *MC* was taking decisions using *Lazy* controller when required. Whereas, the decision, using *Mod* and *Agg* results in big changes, hence influencing the performance. (2) There is a continuous rise in the workload from 5^{th} to 10^{th} hours and it is, therefore, the *Agg* approach performs better in this period due to the *Aggressive* policy and *Mod* results poor performance comparatively. The *MC*, however, behaves as the average of both *Mod* and *Agg* due to its adaptive behaviour. (3) In the last set of hours, the *Agg* and *MC* performs almost similarly due to the fact that the incoming workload is high, hence the decision taken by the *Agg* does not result in a big change in the system and is appropriate with respect to that point in time.

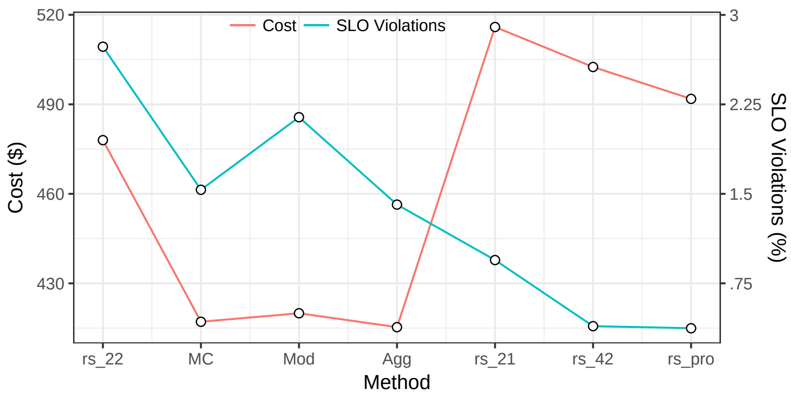
5.5.1.2 *Large variations*

The aggregated view of *Cost* versus *Performance* results is shown in Figure 5.4a. This figure does not show results of some of the methods because they result in large number of *SLO* violations (i.e. $> 5\%$) and therefore they are not included. It is evident from figure 5.4a that the *MC* and rs_{42} produce less number of SLO violations, i.e. 1.42% and 1.43% respectively than other methods. However, the rs_{42} method obtained better performance level at a higher cost, i.e. 560$ in comparison with 468$ of *MC* method. It is also evident from this figure that *MC* has achieved a better performance with slightly lower cost in comparison to other methods. Figure 5.4b presents the number of unsuccessful requests for the three approaches. It is evident from this figure that the number of unsuccessful requests in the case of *MC* approach is much lower than that of *Agg* and *Mod*.

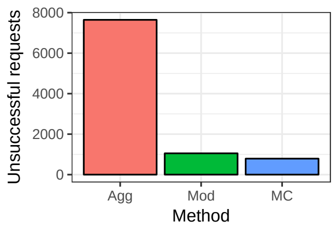
Figure 5.4c shows the per hour percentage number of SLO violations obtained, using *Agg*, *Mod* and *MC* methods. The following conclusions can be derived using this figure when analysing it in accordance with the workload scenario of *Large variations* shown in Figure 5.2a. (1) The *Agg* approach results in poor performance in the part before 7th hour of the workload and after 14th hour. The key reason for this poor performance is that the arrival rate in those hours was low and due to *Aggressive* intensity of scaling action, the system oscillates. (2) The *Mod* approach results in a bad performance, when there is a sharp increase in the workload at 7th hour. (3) The *MC* comparatively results in better performance across the entire duration due to its flexibility of adapting the scaling action based on the current system behaviour.

5.5.1.3 *Quickly varying*

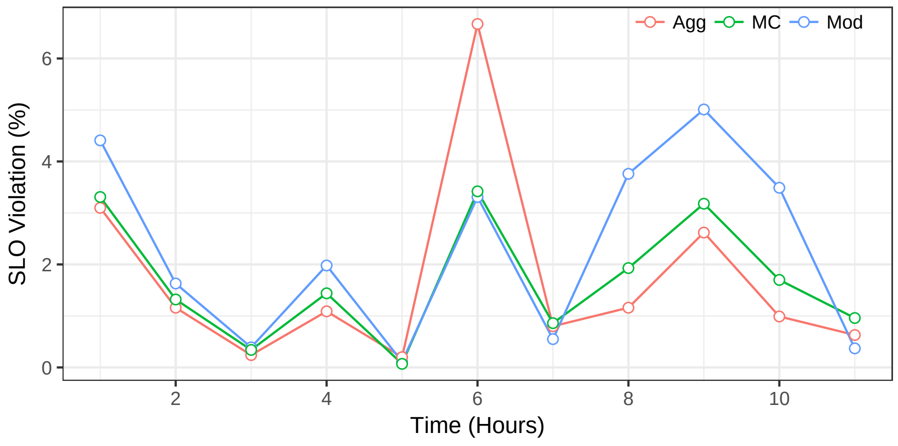
Analogous to the previous scenarios, it is evident from the aggregated results shown in Figure 5.5a that the *Rightscale* methods including rs_{21}, rs_{42} and rs_{pro} result in lower number of SLO violations but at a higher cost compared to other methods. Amongst the rest, the *Agg* method has comparatively achieved better performance than *MC* and *Mod*, where the precise values of SLO violations are 1.40%, 1.53% and 2.14% respectively. Regarding the cost, all three methods spend approximately the same amount, i.e. $\approx$ 417$. However, considering the number of unsuccessful requests shown in Figure 5.5b, the *MC* results in the lowest number,



(a) Aggregated view of cost vs performance

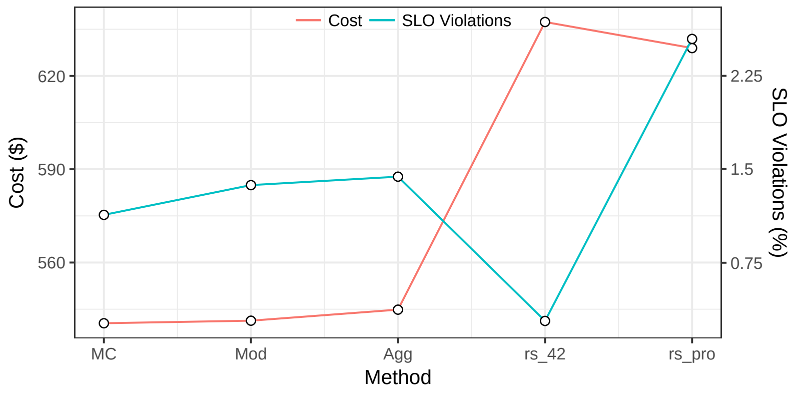


(b) Unsuccesful requests

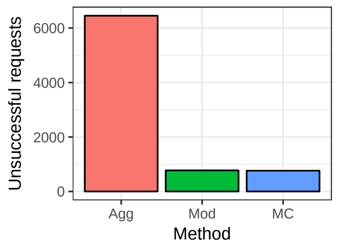


(c) Timeseries view of performance (SLO)

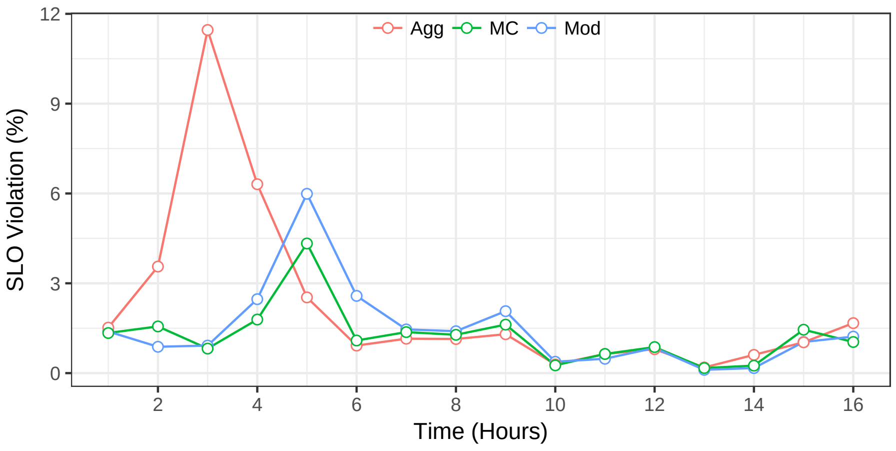
Figure 5.5: Results of *Quickly varying* scenario. The *RightScale* methods results in lowest number of SLO Violations, however at a higher cost. Amongst the rest, the *Agg* approach is overall better, however poor performance, i.e. $\approx 7\%$ of SLO violations was observed in the 6th hour. In contrast, the *MC* method maintains less than 4% of SLO violations in the entire time period.



(a) Aggregated view of cost vs performance



(b) Unsuccesful requests



(c) Timeseries view of performance (SLO)

Figure 5.6: Results of *Tri-phase* scenario. The *RightScale* method (*rs_42*) results in comparatively less number of SLO violations, however, at a higher cost. Amongst others, the *MC* has obtained overall better performance, lowest number of unsuccessful requests and maintained lowest number of SLO violations at the entire time period.

i.e. 790 to 7646 of *Agg*.

It is evident from the aggregated results in this scenario that the *Agg* approach results in slightly better performance than the proposed *MC* approach. However, analysing the time series view of the percentage number of SLO violations on a per hour (shown in Figure 5.5c) demonstrates that the *Agg* approach results in more than 6% SLO violations in the 7th hour. Whereas, the *MC* remains lower than 3.3% at all the time despite a quick variation in the workload. In contrast, the *Mod* can also be seen comparatively larger in the later hours, i.e. from 8th to 10th hour, when there is a sharp change in the workload (see Figure 5.2b) at the beginning of the 8th hour.

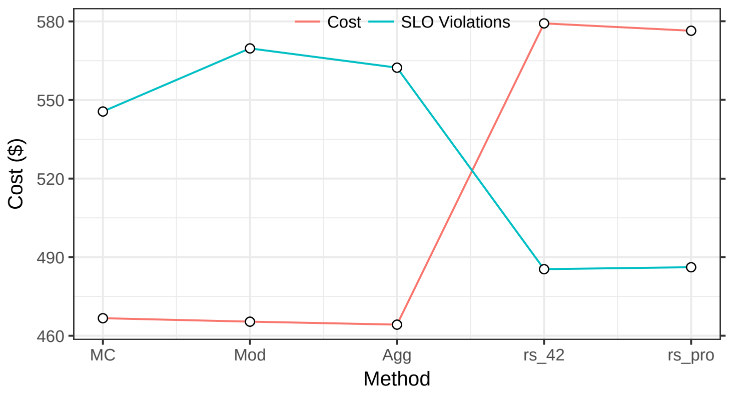
5.5.1.4 *Tri-phase*

The aggregated results for this scenario are shown in Figure 5.6a, which demonstrate that the *MC* outperforms other methods in obtaining the lower number of SLO violations with almost same cost. However the rs_{42} results in better performance but at a higher cost, therefore, we do not consider that as a better result. On the other hand, *MC* also results in a lower number of unsuccessful requests in comparison to that of *Agg*, but similar to the *Mod* method.

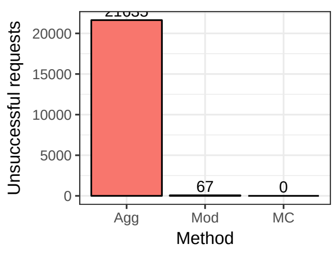
The time series analysis of the percentage number of SLO violations on a per hour basis is shown in Figure 5.6c. This plot indicates that the *MC* has maintained an overall lower number of SLO violations in comparison with other approaches. Specifically, it is very high in the case of *Agg* from 2nd to 4th hour when the incoming workload is low, and slightly more from 4th to 7th hour in the case of *Mod* when the incoming arrival rate was high.

5.5.1.5 *Cyclic*

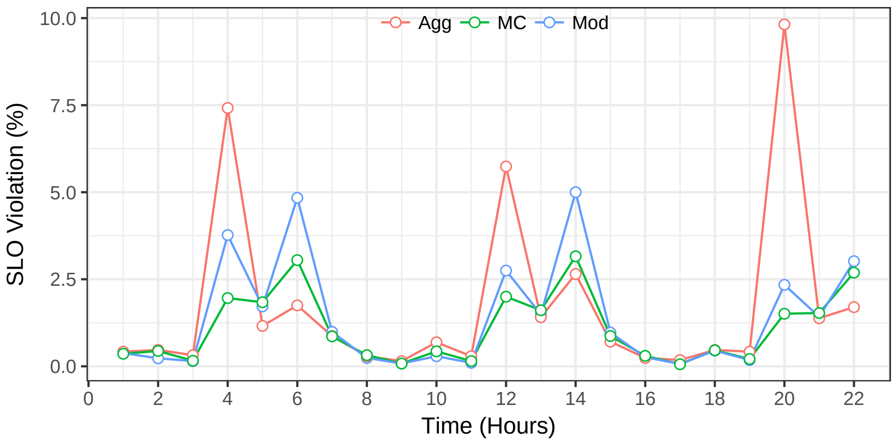
The aggregated results for this scenario are shown in Figure 5.7a. These results indicate that the methods *rs_42* and *rs_pro* have comparatively achieved better performance than all other methods but at a much higher cost. Amongst other,



(a) Aggregated view of cost vs performance

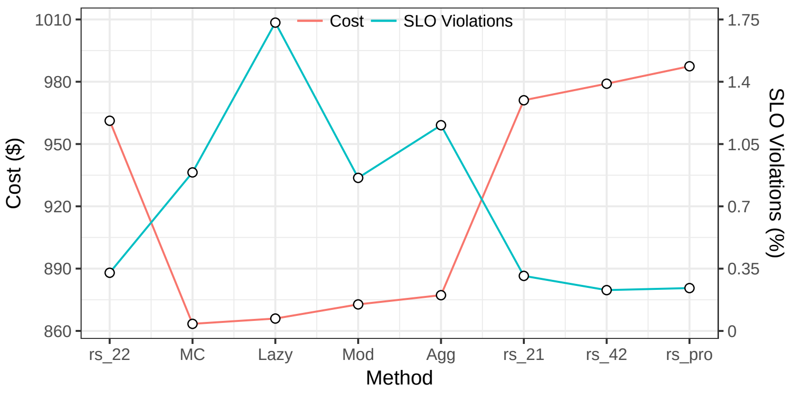


(b) Unsuccesful requests

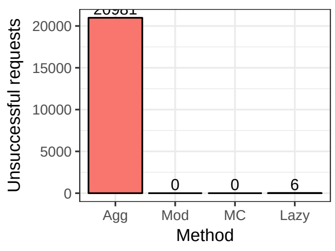


(c) Timeseries view of performance (SLO)

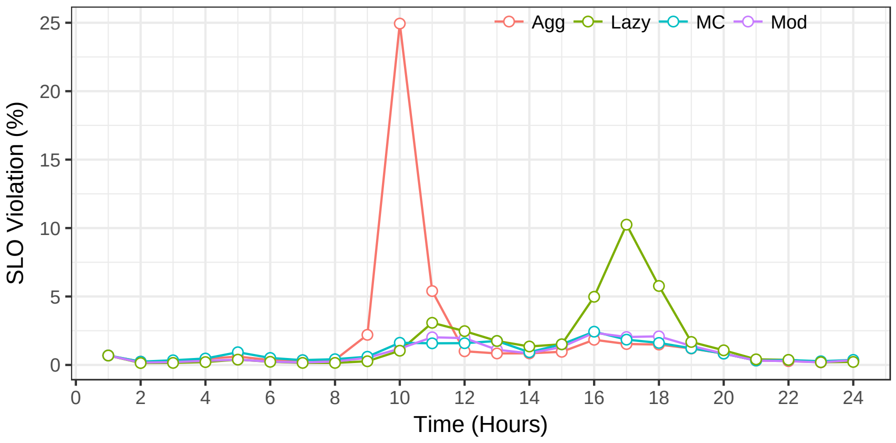
Figure 5.7: Results of *Cyclic* scenario. The *RightScale* methods (*rs_42* and *rs_pro*) has achieved an overall better performance, however at a higher cost. Amongst others, *MC* has achieved 1.05% of SLO violations in comparison with 1.37% of *Mod* and 1.28% of *Agg*. The number of SLO violations on per hour basis remains lower than 3% in case of *MC*, whereas it rise up to 5% in case of *Mod* and 10% using *Agg* approach.



(a) Aggreated view of cost vs performance



(b) Unsuccesful requests



(c) Timeseries view of performance (SLO)

Figure 5.8: Results of *Slowly varying* scenario. The *MC* and *RightScale* methods has achieved an overall SLO violations of less than 0.35%. However, the cost spending in the case of *Rightscale* is larger than that of *MC*. The number of SLO violations observed in all other cases are higher than 0.70%.

the *MC* has performed better and results in the lowest number of SLO violations, i.e. 1.05% in comparison with 1.37% of *Mod* and 1.28% of *Agg*. Regarding cost, all the three methods are same. However, the unsuccessful requests shown in Figure 5.7b, there was none in the case of *MC*, very few (i.e. 67) in the case of *Mod* and more than 20000 in the case of *Agg*.

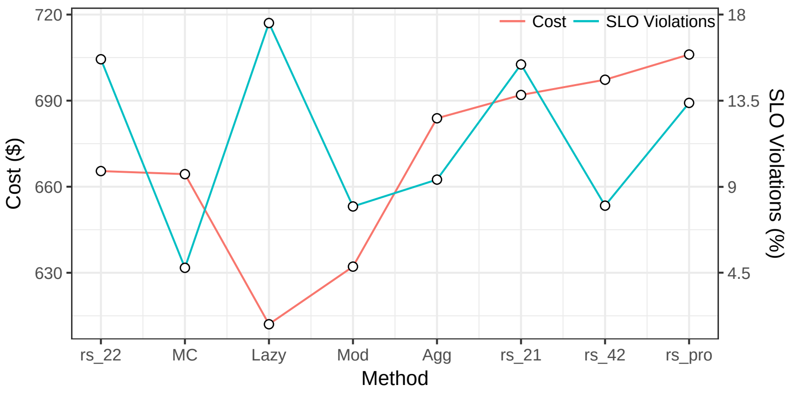
Figure 5.7c shows the time series view of the percentage number of SLO violations on a per hour basis. Analysing this graph per the *Cyclic* workload scenario (see Figure 5.2e) inform the following: (1) In the case of *Agg*, the performance suffers mostly when the arrival rate remains low, i.e. in the 4th, 12th and 20th hours. (2) In the case of *Mod*, the performance degraded when the workload arrival stays low or high, i.e. in the hours 4th, 6th 14th . (3) Whereas in the case of *MC*, the number of SLO violations remains comparatively lower, in almost the entire duration of the experiment.

5.5.1.6 *Slowly varying*

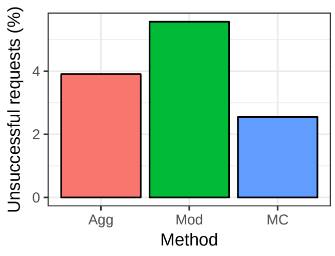
It is evident from the aggregated results presented in Figure 5.8a that the various settings of the *Rightscale* method result in better performance but at a higher cost. Amongst others, the *MC* and *Mod* result in almost similar results, i.e. 0.89% and 0.86% regarding the SLO violations. However, *Mod* is slightly expensive, i.e. 872$ to 863$ of *MC*. Whereas the performances obtained using *Lazy* and *Agg* are poorer in comparison with that of *MC* and *Mod*. The time series view of percentage number of SLO violations (as shown in Figure 5.8c) tell us the following: (1) the performance suffers largely in the case of *Agg* in the 10th hour when the arrival rate is low (see Figure 5.2f); (2) the performance degrades in the case of *Lazy* at the 17th hour when the arrival rate is high. Lastly, there are no unsuccessful requests recorded except in the case of *Agg* as shown in Figure 5.8b.

5.5.1.7 *On-off*

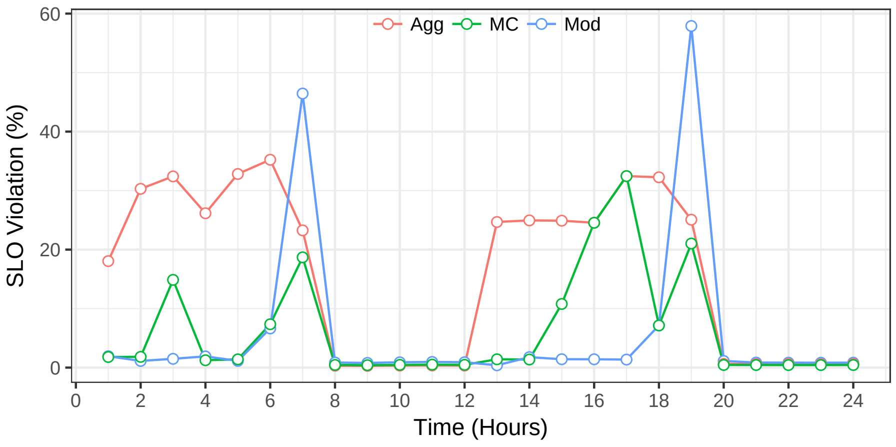
The aggregated results for this scenario are shown in Figure 5.9a. It is evident from this figure that the performance obtained using *MC* is comparatively much



(a) Aggreated view of cost vs performance



(b) Unsuccesful requests



(c) Timeseries view of performance (SLO)

Figure 5.9: Results of *On-off* scenario. The least number of SLO violations observed in this scenario is using *MC*, which is recorded as 4.76%. In all other cases, the number of SLO violations are more than 7.98%. Similarly, the number of unsuccessful requests in the case of *MC* is much lower, i.e. 2.55% to that of *Mod* method, which is recorded as 5.57%.

better than all other approaches. The second best approach is the *Mod*, where the percentage number of SLO violations are 7.98% to 4.76% of *MC*. However, in the case of *MC*, the *Cost* is slightly more than that of *Mod*. Similarly the number of unsuccessful requests (shown in Figure 5.9b) in the case of *MC* is much lower, i.e. 2.55% to that of *Mod*, i.e. 5.57%.

Figure 5.9c displays the time series view of the percentage number of SLO violations on a per hour basis. It is evident from this figure that the performance obtained using *Agg* method suffers largely in the first few hours, i.e. when the arrival rate of the workload remains low (see Figure 5.2g) and similarly in duration from 12th to 20th hours. Whereas in the case of *Mod*, the performance largely suffers at a time, when the workload changes from low to high, i.e. in 7th and 19th hours. The *MC* method, in contrast, performs much better in the hour of swift change of workload from low to high, i.e. in 7th and 19th hours. However, focusing on the 3rd hour and hours from 15th to 18th, the performance also degrades using *MC* approach. This behaviour has not been noticed in the previous scenarios. The reason of this performance degradation is the oscillation caused using *MC* method at those times. Apart from these specific hours, the *MC* achieves better performance than all other approaches. The oscillation aspects are further discussed in Chapter 6.

5.5.2 *Summary and Discussion*

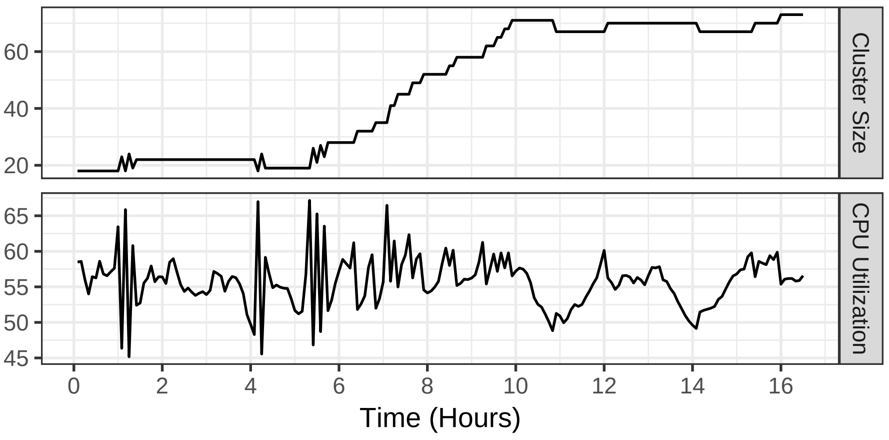
Sections 5.5.1.1 to 5.5.1.7 explain and analyse the obtained results for each workload scenarios employed for the evaluation. In the same realm, this section summarises the findings and briefly discusses each of the applied methods in light of the obtained computational results.

1. Rightscale: It is observed from the obtained results that the different settings of the *Rightscale* method comparatively produce better performance, with a higher cost though. This indicates the over-provisioning phenomenon

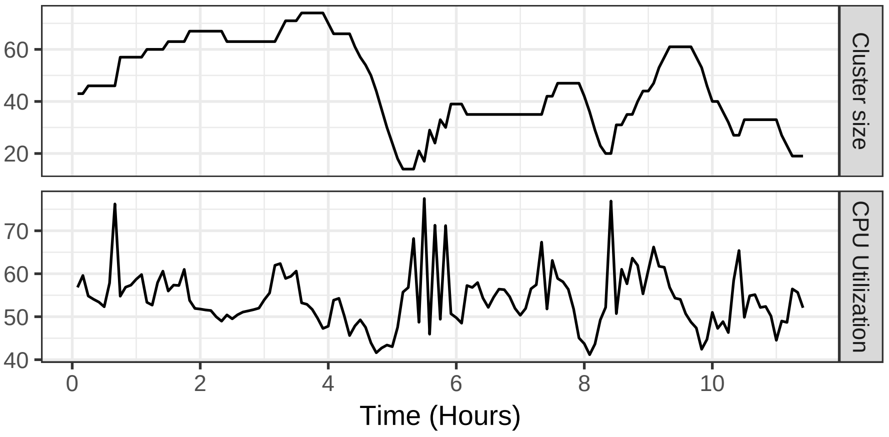
described in Section 2.2.3, where more than required resources are allocated to avoid performance degradation. The better performance with a higher cost phenomena occurred in workload scenarios, where transitions in workloads are comparatively smooth, e.g. in the case of *Dual-phase*, *Cyclic*, and *Slowly varying*. In other scenarios where sharp changes occurred such as *Large variations* and *On-off*, the performance is comparatively poorer than other approaches, despite being expensive. A key reason behind is the underlying static scaling behaviour, where a scaling action is performed using a uniform quantity.

2. Aggressive: It is observed that the aggregated results of performance obtained using the *Aggressive* approach in the case of *Dual-phase* and *Quickly varying* scenarios are comparatively better than *MC*. However, the time series analysis of those scenarios indicates that the performance of the system is poor in certain hours specifically when the arrival rate of the workloads were low. The key reason for this behaviour is the inappropriate scaling intensity that causes a bigger change in some cases, e.g. observe the *Cluster size* and *CPU utilisation* relationship in Figure 5.10a for the first two hours & 5th hour in the case of *Dual-phase* scenario and 6th hour in the case of *Quickly varying* scenario. The worst situation arises in the case of *Large variations*, where the system resources oscillate (see Figure 5.10c) when the arrival rate of the workload was low. This indicates that using a uniform *Aggressive* method at the entire time is not a good choice and could lead the system to an unstable state.

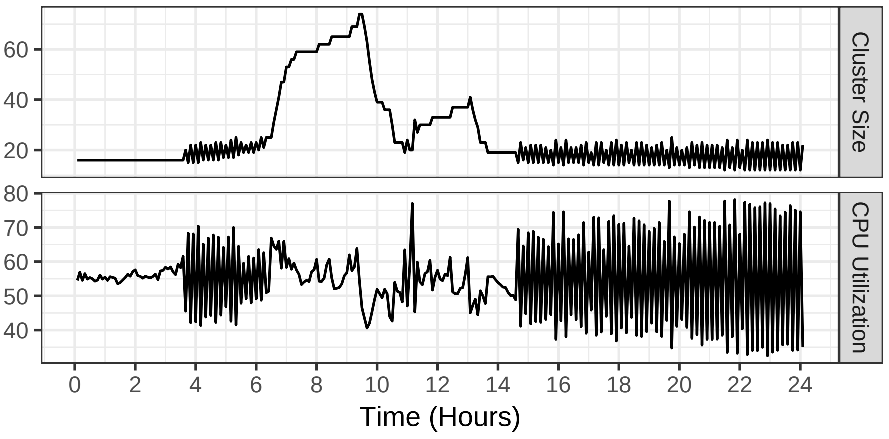
3. Moderate: The performance of the *Moderate* policy works well in the following two scenarios. First, when the incoming workload remains stable in a particular region, e.g. the segment after the 8th hour in the case of *Large variation* (see Figure 5.4c). Secondly, when the arrival rate changes slowly, e.g. in the case of *Slowly varying* (see the results in Figure 5.8c). However, the *Moderate* method performs poorly in comparison to the *Agg* and *MC*, when there are sharp changes in the incoming workloads, e.g. the segment



(a) Dual-phase



(b) Quickly varying



(c) Large variations

Figure 5.10: *Cluster size* versus *CPU Utilisation* using *Aggressive* method. Bumpy transitions are observed in *DualPhase* and *Quickly varying* scenario in the first six hours time period, whereas oscillation episodes are observed in *Large variation* scenario.

after the 7th hour in the case of *Quickly varying*, the 7th and 19th hours in the case of *On-off* etc.

4. MC: The *MC*, in contrast to the above mentioned methods works well in scenarios like *Large variations*, *Tri-phase*, *Cyclic* and *On-off*. In the case of *Dual-phase* and *Quickly varying* scenarios, the aggregate results were slightly poorer than that of the *Aggressive* method. However, considering the time series analysis, the *MC* approach maintains better performance during the entire time. Whereas, in the case of *Slowly varying*, the *MC* and *Moderate* policy have achieved similar performance.

The above discussion indicates that using a uniform fixed policy is unable to cope with changing workload conditions. In contrast, the proposed *Hard switching* consists of the collection of the same policies with an additional switching mechanism result in an improved system performance without an increase in the operational cost.

6 BIOLOGICALLY-INSPIRED SOFT SWITCHING IMPROVEMENTS

This chapter introduces the proposed biologically-inspired *Soft switching* approach to address the cloud elasticity problem. Section 6.1 describes the motivation behind the proposition of the soft switching method. Section 6.2 briefly introduces the related concepts including action selection, BG and how these are related to the cloud resource provisioning. Section 6.3 examines the proposed biologically inspired soft switching approach in detail. Finally, the experiments and the computational results obtained using *Soft switching* method in comparison to the *Hard switching* approach are analysed in Section 6.4.

6.1 MOTIVATION

The computational results obtained in the previous chapter demonstrate that the proposed *Hard switching* based approach has the potential to achieve better system performance in comparison to the benchmark methods. However, such methodologies are more often criticised for their associated unwanted behaviour, termed as bumpy transition, that could lead the system to an oscillatory state [52, 53, 54]. The occurrences of bumpy transitions may be due to an inappropriate switching or some larger changes in the system state. The oscillation (explained in Section 2.2.3) regarding the cloud resource provisioning refers to the scenarios, where a target system acquires and releases computational resources periodically. Such events are undesirable and thus need to be avoided.

The effects of a scaling action reflect in the output of the system, which in our case is the measured *CPU utilisation*. Analysing the time series view of the measured *CPU utilisation* provides insight regarding the presence of bumpy transitions and oscillatory behaviour. The previous chapter (in Section 5.5.2) mentions that the oscillatory behaviour occurred in certain scenarios in the case of *Aggressive* method. Similarly, Figure 6.1 shows the time series view of the measured *CPU utilisation* recorded using the proposed *Hard switching* approach for *Dual-phase*, *Tri-phase* and *On-off* scenarios.

Focusing on the highlighted parts of Figure 6.1 indicate the presence of bumpy

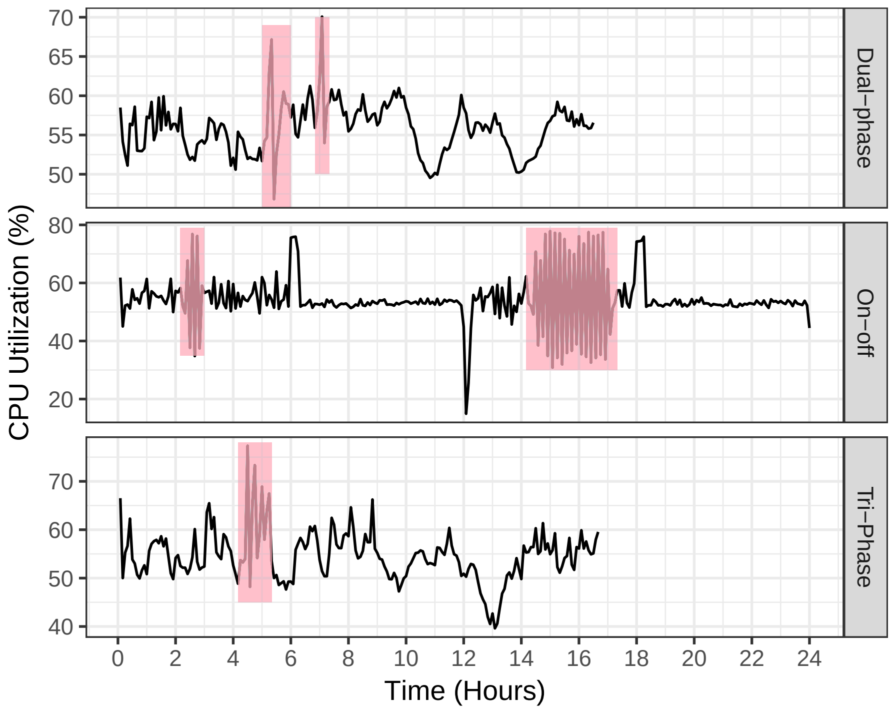


Figure 6.1: Time series view of measured *CPU utilisation* obtained using proposed *Hard switching* approach. The highlighted parts in the plots demonstrate the episodes of bumpy transitions that could lead the system to an oscillatory state.

transitions in the case of a *Dual-phase* and *Tri-phase*. However, in the case of *On-off*, system resources oscillate at different time intervals. This *On-off* scenario is the worst amongst all the tested scenarios in the previous chapters. Such oscillatory behaviour is unwanted and must be avoided. Considering such scenarios, this research aims to reduce the likelihood of the occurrences of bumpy transitions and oscillations mentioned above. More specifically, this chapter focuses on the phenomenon that how a real biological system (animal for example) selects the next action to be carried out. We aim to explore the capabilities of biologically (cognitive) inspired action selection process for seeking the possibility of more smoother (bumpless) transitions, hence improving the stability perspective of the proposed elastic method.

6.2 ACTION SELECTION, BG AND ELASTIC CONTROLLER

The process of determining the next action refers to the action selection problem, which has been the focus of research in many fields [205, 206]. Formally, an action selection is the process of deciding what to do next from a set of available actions by an agent, based on some knowledge of the internal state and some provided sensory information of the environmental context to best achieve its desired goal [7]. Over a period of time, researchers have learnt that in animal's brain, the problem of action selection is handled through the use of a central switching mechanism [207, 208]. This mechanism is implemented by a group of subcortical nuclei collectively refers to as BG [207, 208]. A brief description of the functional anatomy of BG is as following [7].

The key components (nuclie) of a BG include *Straitum*, *Subthalamic nucleus (STN)*, *Globus pallidus (GPe)* and *Substantia nigra (SNr)*. The key function of this group of inter-connected nuclie is to activate the desired actions and to suppress the undesirable competing actions. Figure 6.2 (borrowed from [7]) depicts the essential circuitry of BG. The *Straitum* is the main input nuclei of a *BG* that receives action request in form of neural signals (or channels) from related functional sub modules of brain. The sensory and associated motivational inputs in larger range are received by the *Straitum neurons* at the *STN*. The activity level of the *Straitum neurons* represents the salience of an action. The *SNr* and *GPe* are the key output components of the *BG*, which are tonically active and are responsible to direct the continuous flow of inhibition in brain. This inhibition becomes a source for the generation of movement in brain.

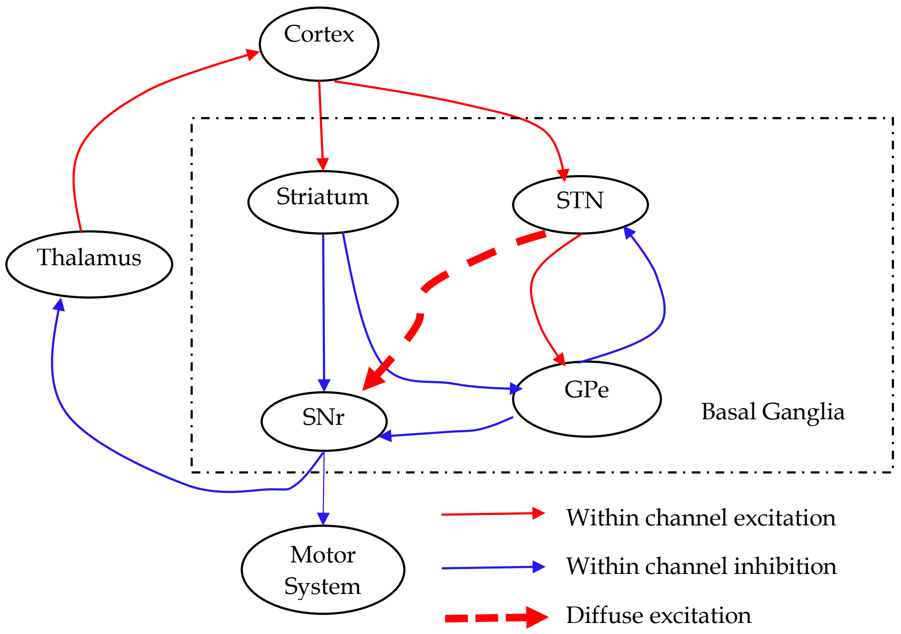


Figure 6.2: BG functional circuitry [7]

The process of BG as an action selection mechanism is as follows. The input channels depicts the competing choices (or actions). Providing the sensory and associated motivational information/inputs, the BG are expected to decide amongst the possible choices and activate the winner channels by dis-inhibiting the corresponding motor circuits [205]. For a detailed functional anatomy of BG, see [7] for details.

The elasticity controller works as an autonomous agent to make scaling decision by utilising the various kinds of available information such as (1) application status, e.g. current performance level, (2) environmental information, e.g. workload arrival rate, (3) internal state of the system, e.g. resource utilisation level. Based on the autonomous behaviour of an elastic controller and the general nature of action selection problem, we have formulated the selection of controllers in the case of proposed multi-controller approach as an action selection problem. Therefore, we treat each of the controllers as an action and exploit the possibility to incorporate the BG behaviour into the proposed framework. The key objective of this research is to reduce the likelihood of bumpy transitions and oscillatory behaviour of the *Hard switching* approach.

6.3 BIOLOGICALLY-INSPIRED SOFT SWITCHING FRAMEWORK

The *Hard switching* approach described in Chapter 4 has the potential to improve system performance in comparison to the benchmark methods. However, the possibility of leading the system to an oscillatory state is undesirable. The oscillation of resources may have deteriorating effects on the system performance as well as on the operational cost. It is therefore desirable that the proposed framework should result in smoother transitions to avoid any oscillation. *Soft switching*, on the other hand, is an alternative technique used to avoid such unwanted behaviour. Such a mechanism in contrast to *Hard switching* has the possibility to select multiple actions rather than one best choice. Following are the key benefits of such an approach: (1) avoidance of singularity and sensitivity problems, (2) improvement of robustness and stability aspects and (3) elimination of chattering issues [209].

This research incorporates the BG based mechanism as an action selection method to implement the *Soft switching* behaviour. We integrate a well established BG based computational model of Gurney et al. [57, 58]. The key advantages of this computational model include its biological plausibility and computational efficiency [210]. Our inspiration of utilising this model comes from the research work carried out in the field of autonomous vehicle control (AVC), where a similar approach has been used for the motion control of autonomous vehicle [210] and the cruise control system [206].

Figure 6.3 shows the architectural diagram of the proposed BG based soft switching approach. Comparing this diagram with the *Hard switching* approach (shown in Figure 4.1), the following three differences are highlighted: (1) the integration of the BG component, (2) the output of the FIS component and (3) the final output

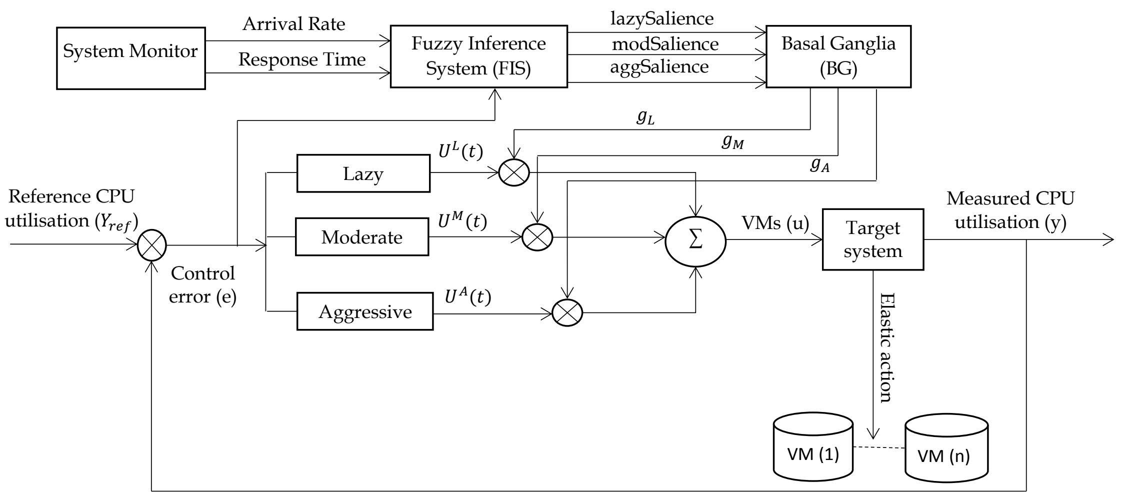


Figure 6.3: Architectural diagram of the proposed biologically-inspired soft switching cloud elasticity framework

of the control system. The details of each of these differences are provided in the following subsections:

6.3.1 *The BG Component*

This component integrates the BG based computational model, which builds on the functional anatomy of BG briefly described in Section 6.2. Some examples of such models include [57, 58, 211, 212, 205, 213]. Amongst these models, we utilised the computational model proposed in [57, 58]. However, any model can be used as our aim is not to identify the best action selection or biologically-inspired computational model rather to demonstrate the effectiveness of such an approach in the context of the cloud elasticity.

Focusing on Gurney et al. [57, 58] computational model, the brain subsystems send excitatory signals that represent the behavioural expressions to the BG. Each behavioural expression defines an action in BG and its strength is determined by the salience that represents the activity level of its neural representation. These actions are mediated through the release of inhibitory signals. Thus in each iteration, the functional model accepts a set of salience signals and produces a set of selected and unselected signals. The functional model can be run in one of three modes, i.e. *Hard, Soft* or *Gate* mode. In the case of the *Hard* mode, a maximum of one action can be selected. Alternatively, multiple actions can be selected in the case of a *Soft* and *Gate* modes. However, in a *Soft* mode, the selected actions are returned as an output, whereas in the case of *Gate*, the model returns the proportion of each selected action. For a detailed description of the functional model refers to [57, 58].

The BG component, shown in Figure 6.3, accepts three inputs namely *lazySalience*, *modSalience* and *aggSalience*. These inputs represent the strength of selection for each controller (depicting as action). The values for these salience signals are computed by the FIS (details provided in the next section). The output of the BG component depends on the running mode of the computational model (further explained in Section 6.3.3).

6.3.2 *The Modified FIS*

The BG based computational model requires salience signals as inputs. Thus the first issue to be dealt with is the generation of salience signals. The method to generate the salience signals can make use of system's internal state, various performance metrics or available sensory information [210]. Therefore, we have extended the FIS, used as a switching mechanism in Chapter 4, to generate the inputs (salience signals) for the BG component of the framework. The inputs of the modified FIS remains the same, i.e. *Workload, ResponseTime* and *ControlError*. However, the output is changed from one, i.e. *Controller* to three *lazySalience, modSalience* and *aggSalience*. Each of these outputs represents the salience strengths for the selection of each of the three controllers. The details of the changes carried out are as following:

1. Membership functions: The inputs of the modified FIS do not change, and therefore the corresponding membership functions of the input fuzzy variables remain the same. However, the output is changed, therefore, the *Controller* membership function is replaced with three new membership functions, i.e. one for each newly introduced output. Similar to the *Controller* membership function, we have used the basic triangular type for all the outputs. The membership function for each salience signal variable is of the form shown in Figure 6.4.

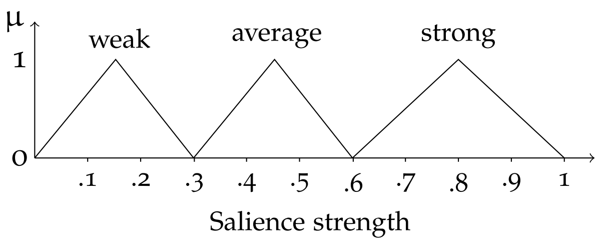


Figure 6.4: Membership functions for each salience variable, i.e. Lazy, Moderate and Aggressive

2. Fuzzy rules: The fuzzy rules are responsible to generate the salience inputs. The fuzzy rules described in Chapter 4 are revised accordingly. Figure 6.5 depicts an example of the modified rule. The inputs of the rules are the same as in the case of *Hard switching* rules presented earlier in Figure 4.8. However, the output can be formed using the linguistic terms (*weak, average* and *strong*) for each salience.

6.3.3 *Derivation of Final Output*

As mentioned earlier, the adopted functional model support three modes including *Hard*, *Soft* and *Gate* mode, hence the final decision, i.e. the number of VMs depends on the running mode of this framework. In this research we have focused on *Soft* and *Gate* modes as they support the soft switching behaviour by allowing the possibility of selecting more than one controller. The final decision, whilst considering these two cases can be derived using the output signals returned by

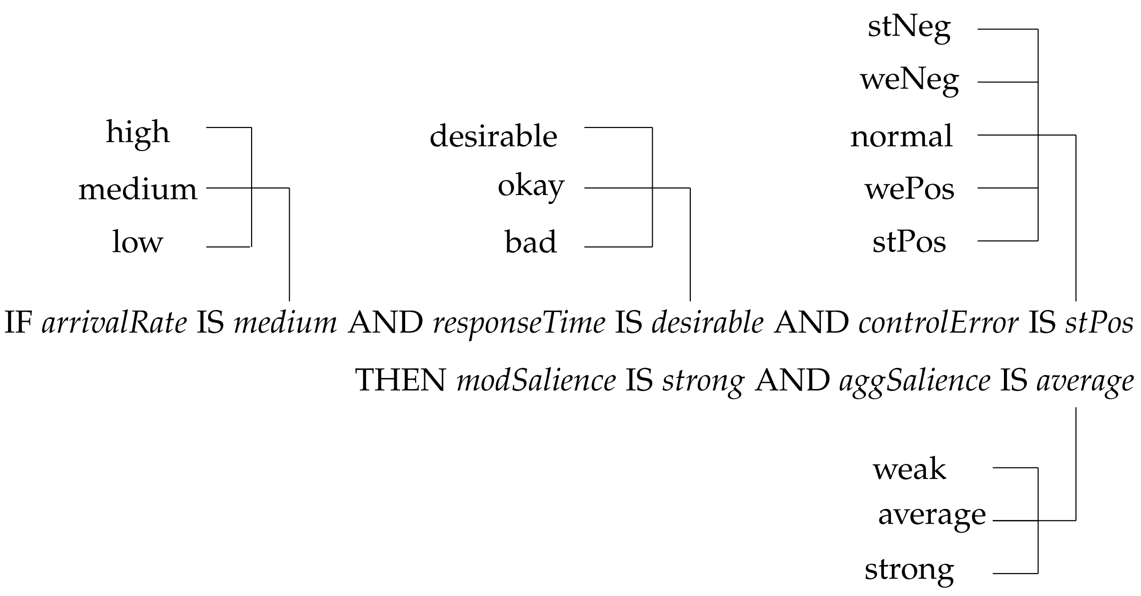

stNeg
weNeg
high
desirable
normal
medium
okay
wePos
low
bad
stPos
IF arrivalRate IS medium AND responseTime IS desirable AND controlError IS stPos
THEN modSalience IS strong AND aggSalience IS average
weak
average
strong

Figure 6.5: Example of a soft switching rule

the BG component and the outputs of the individual controllers. The following equation represents this derivation.

$$u_t = \frac{(u^L(t) * g_L) + (u^M(t) * g_M) + (u^A(t) * g_A)}{g} \tag{6.1}$$

The u_t in the above equation represent the final decision. $u^L(t)$, $u^M(t)$ and $u^A(t)$ represents the output of individual controllers, i.e. *Lazy*, *Moderate* and *Aggressive* respectively. These outputs are computed as per the equations described in Section 4.4.4. Whereas, g_L, g_M and g_A are the output signals returned by the BG component (shown in Figure 6.3). The values of these signals in the case of *Soft* mode are either 1 or 0. The value 1 represents the selection of that particular controller, whereas the value 0 indicates no selection and in such a case, the output of that controller shall not be considered in the final output. On the other hand, in the case of *Gate* mode, the value of these output signals represent the proportion (between 0 to 1). Lastly, the denominator g represents the number of those output signals with a value higher than zero. However, it is not always the case that more than one controller to be selected.

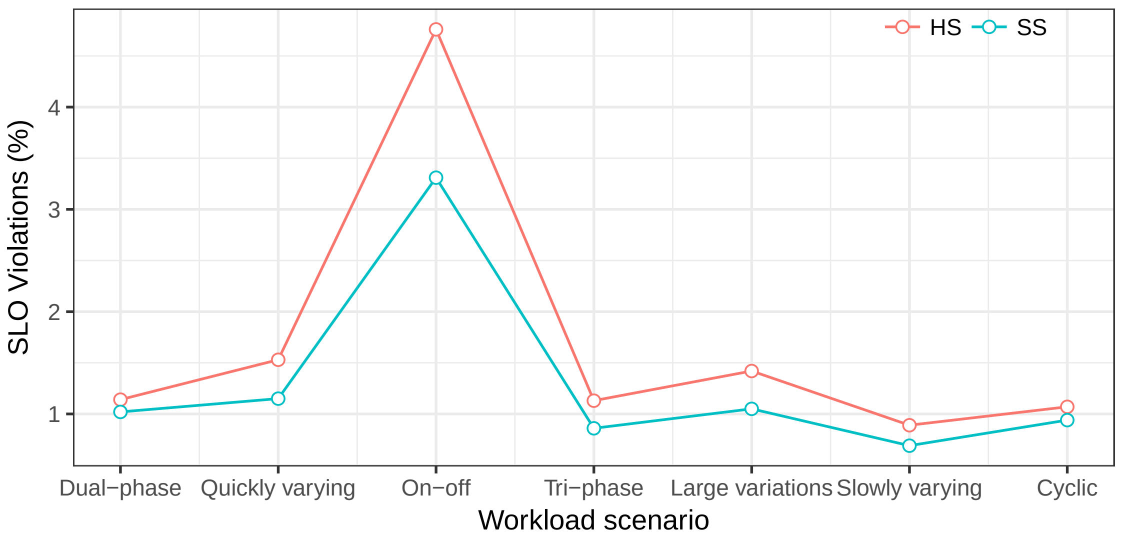
6.4 EXPERIMENTATION AND EVALUATION

The BG based proposed method is implemented as one of the elastic controllers into our experimental framework (explained in Section 5.1). We have used the same experimental settings and scenarios (i.e. gains for controllers and workloads) as in Chapter 5 to evaluate the *Soft switching* approach. It is already concluded in Chapter 5 (Section 5.5.2) that *Hard switching* achieved better results compared to the benchmark methods. Therefore, in this chapter, we only compare the computational results obtained using *Soft switching* to that of *Hard switching*. Hence, we aim to demonstrate the effectiveness of BG based *Soft switching* approach to that of *Hard switching* approach. Thus, we present and discuss the obtained computational results in the following two aspects:

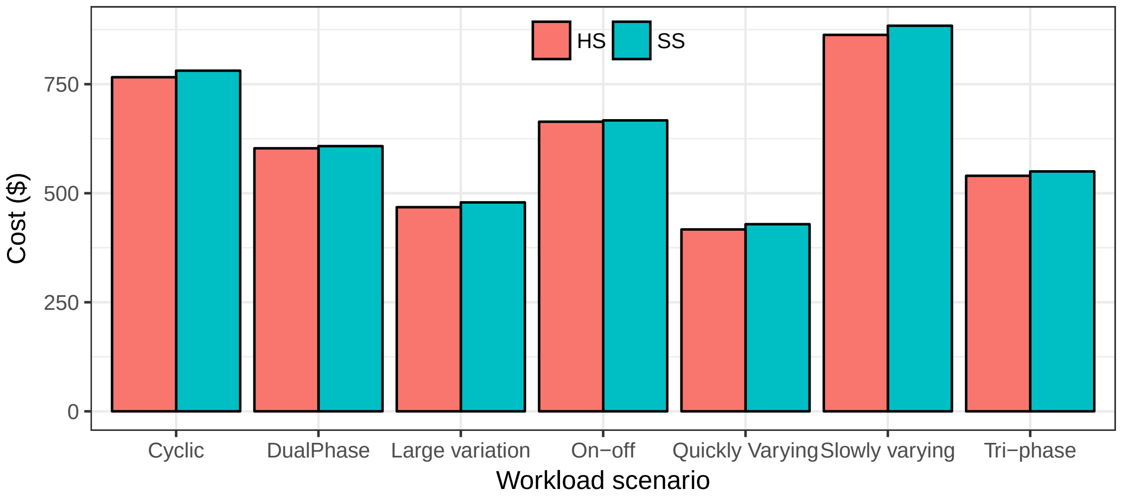
6.4.1 *Performance*

Figure 6.6 shows the aggregated view of the results obtained using both the proposed approaches, i.e. *Soft switching* and *Hard switching*. These approaches are represented as *SS* and *HS* respectively in the reported results. It is evident from Figure 6.6a, which compares the obtained performance that the *SS* approach has obtained a lower number of SLO violations in all of the employed workload scenarios to that of *HS*. The comparison of the cost perspective are shown in Figure 6.6b, where the spending using the *SS* approach is almost similar to that of *HS*. This demonstrates that the *SS* approach results better performance compared to that of *HS* without increasing the operational cost of the system.

Figure 6.7 shows the time series view of the percentage number of SLO violations on a per hour basis for each workload scenarios. This diagram provides an insight into the performance of both the approaches on an hourly basis. Each plot of this diagram represents the result for each of the employed workload scenario. The analysis of these plots provide the following insights: (1) The per-



(a) Performance



(b) Cost

Figure 6.6: Aggregated view of results. In all employed scenarios, the proposed *Soft switching* approach results in comparatively better performance with almost same cost than that of *Hard switching* approach.

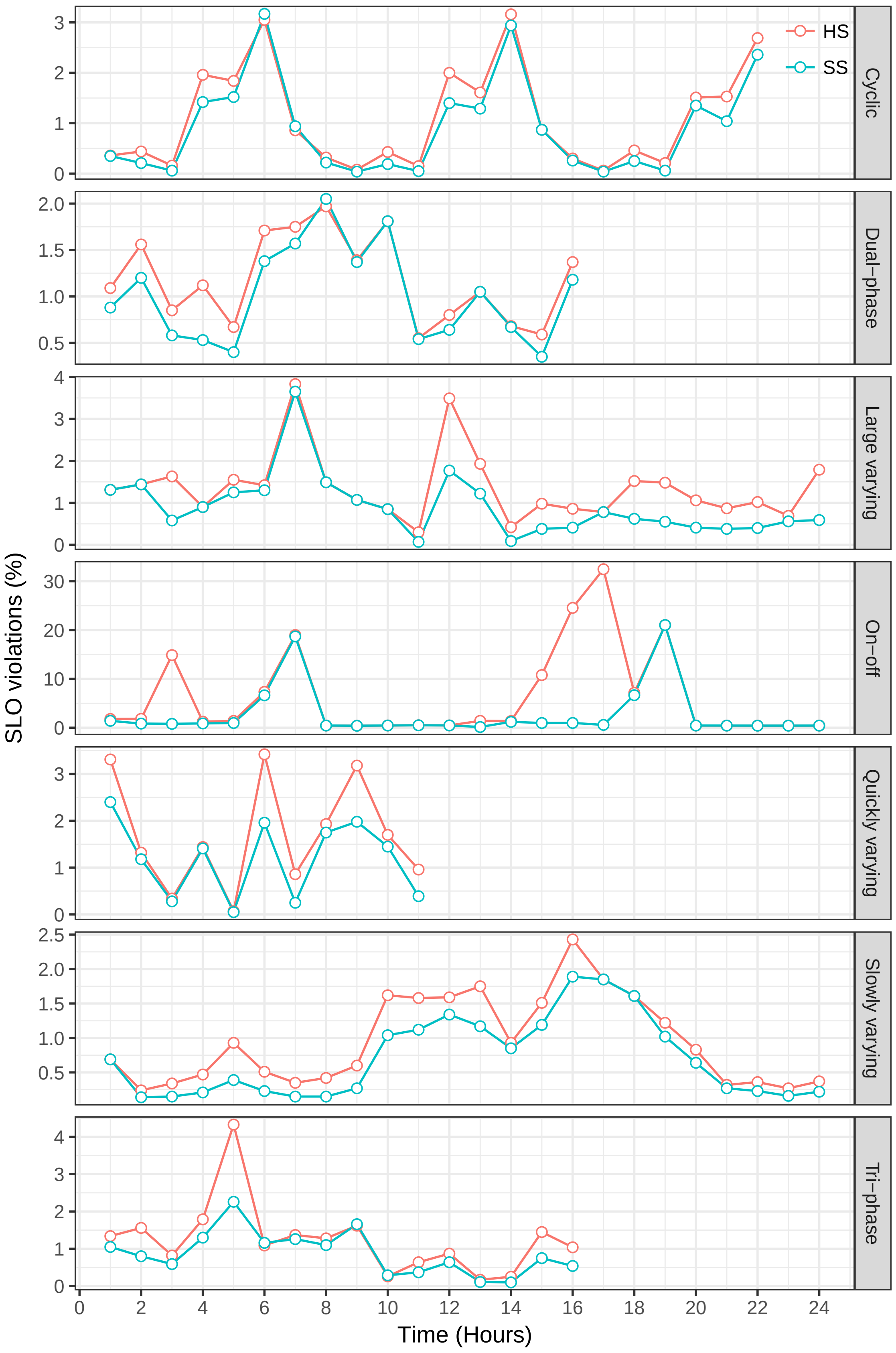


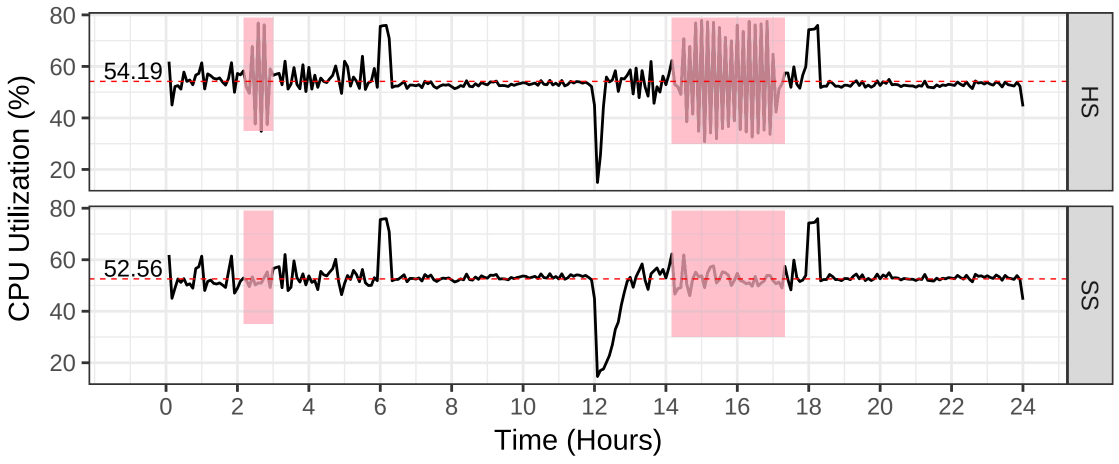
Figure 6.7: Time series view of performance. In all employed scenarios, the number of SLO violations on per hour basis using *Soft switching* remains lower or same than that of *Hard switching* approach.

formance obtained using *SS* approach in almost each hour for all the workload scenarios is either similar to that of *HS* or comparatively better. This indicates a higher potential to maintain better performance during the entire period of the experiment. (2) The *SS* and *HS* approach behave almost similarly in scenarios, when there are sharp increases in workload, e.g. the 6th hour in the case of *Cyclic*, the 7th hour in the case of *Large variations* and the 7th & 19th hours in the case of *On-off* scenarios. This indicates that the *SS* approach has figured out a sharp increase in the workload and behaves similar to *HS* to avoid the degradation of the system performance. (3) The *SS* approach performs comparatively better when the arrival rate of the workload remains low, e.g. initial 5 hours period in the case of *Dual-phase* and the hours from 15th to 18th in the case of *On-off*. This indicates that at the time of low workload, the decision of *HS* affects the performance more due to its best controller selection strategy in comparison to that of *SS* approach.

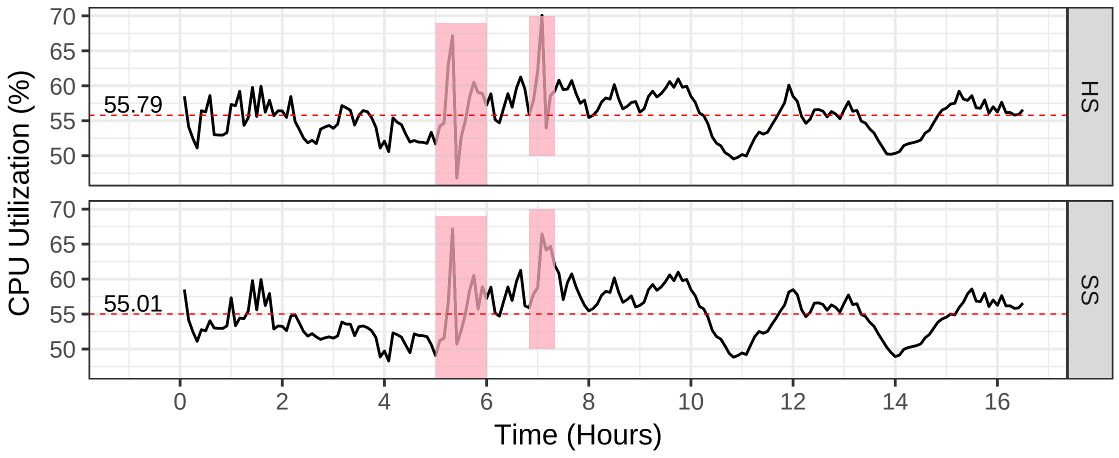
6.4.2 *Oscillatory Behaviour*

The results presented in the previous section demonstrate the effectiveness of the *Soft switching* approach regarding the improvement of the overall performance. However, it is mainly adopted to improve the possibility to reduce the likelihood of bumpy transitions and oscillation in comparison to the *Hard switching* approach. This section discusses this possibility by comparing the measurements of system's output, i.e. measured *CPU utilisation* obtained using each approach, i.e. *HS* and *SS*.

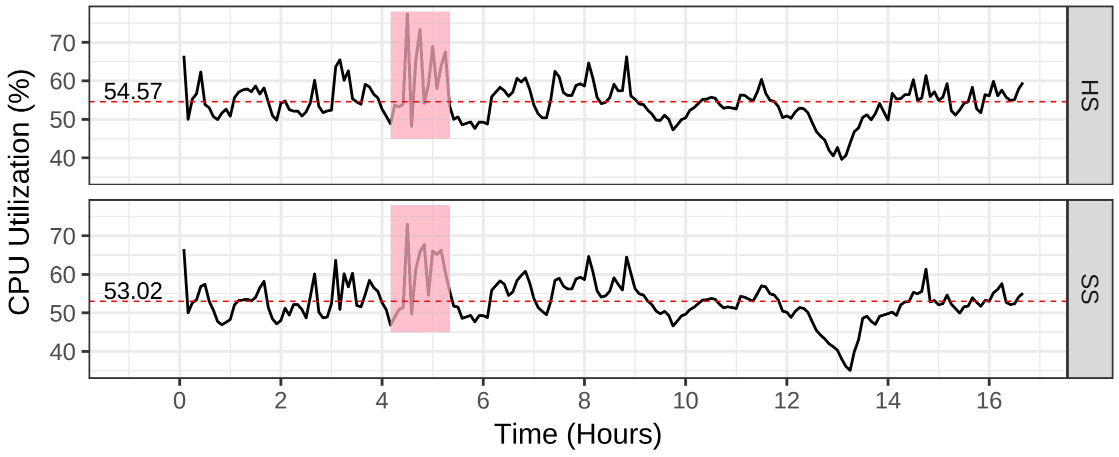
Figure 6.1 has shown, the time series view of measured *CPU utilisation* obtained using *HS* approach for the following three workload scenarios, i.e. *Dual-phase*, *Tri-phase* and *On-off* respectively. Amongst these scenarios, the presence of an oscillatory behaviour can be clearly seen in the case of *On-off*, whereas minor bumpy transitions are visible in the case of *Dual-phase* and *Tri-phase*. These key spots are highlighted in the figure. Similarly, we have recorded the measured



(a) On-off



(b) Dual-phase



(c) Tri-phase

Figure 6.8: CPU utilisation (HS vs SS). Each pair of plots are obtained using *SS* and *HS* methods. The oscillations did not occur in the *On-off* scenario using *SS*, where the intensity of bumpy transitions in other cases are also reduced.

CPU utilisation using the *SS* approach, which is shown in comparison to that of *HS* approach in Figure 6.8.

The *CPU utilisation* of the *Cluster* is recorded on a per minute basis and an average of 5 minutes is considered for decision-making and final reporting. Thus, each measurement of these plots represents the average *CPU utilisation* of 5 minutes duration. The analysis of these plots provide the following insights:

1. Focusing on the highlighted parts of *HS* plot (shown in Figure 6.8b) for *On-off* scenario clearly hint at the presence of oscillations at two occasions, i.e. in the 3rd hours and in the 15th to 16th hours. On the other hand, using *SS* approach, no such oscillations can be seen in the corresponding *SS* plot, which demonstrates clear improvements.

2. Figure 6.8b shows the time series view of *CPU utilisation* for the *Dual-phase* scenario. It is evident from this figure that there is no oscillation using both the approaches. However, the highlighted part in the case of *HS* shows some bumpy transitions, i.e. in the 6th and 8th hours. Whereas in the case of the *SS* approach, the intensity of these bumpy transitions is reduced as is evident by visual inspection of both plots of Figure 6.8b. Moreover, the variance of *CPU utilisation* measurements of three hours, i.e. from 6th to 8th is calculated for both cases. These calculations are recorded as 12.84 and 15.24 for the *SS* and *HS* respectively. This demonstrates that the *SS* results in fewer variations compared to that of *HS* in those three hours. Similar results (shown in Figure 6.8c) are also observed in the case of *Tri-phase* scenario, where the variation in the case of *SS* from 4th to 6th is fewer than that of *HS*.

3. The red dashed line in each plot of Figure 6.8 represents the mean *CPU utilisation* obtained using the respective methods in each corresponding scenario. In all of the given three scenarios, the mean obtained using *SS* approach is comparatively less than that of *HS*, e.g. in the case of an *On-off*, the means are 52.56 and 54.19 recorded using *SS* and *HS* respectively. This

demonstrates that the *SS* approach is comparatively better and maintains the *CPU utilisation* below 55% more often than the *HS* approach.

In light of the above discussions, we can claim that the BG based *Soft switching* approach has higher potential to reduce the number of SLO violations. Therefore, we obtain comparatively better system performance. Moreover, compared with the *HS* approach, it has demonstrated the possibility of reducing the likelihood of bumpy transitions and oscillatory behaviour. The intuitive explanation for this improvement is the integration of controllers (shown in Equation (6.1)) in a biologically-inspired fashion augmented with the BG process, which facilitates the natural selection of actions that result in less 'bumping' at the switching time [214]. Moreover, the computational model of [57, 58] in particular is successfully validated to avoid oscillation [205].

7

OPTIMISATION OF FUZZY MEMBERSHIP FUNCTIONS

This chapter formulates the construction of the fuzzy membership functions, introduced in Chapter 4, as a multi-objective optimisation problem to explore the near optimal parameter settings for their design. Section 7.1 describes the motivation and aims of this research. Section 7.2 formally introduces the problem, the parameters considered, their ranges and constraints. Section 7.3 provides details of the employed GA in the context of the underlying problem, whereas Section 7.4 explains the problem formulation using the Taguchi approach. Finally, the experiments and the computational results obtained using both employed techniques are discussed in Section 7.5.

7.1 INTRODUCTION

The switching mechanism, implemented as a FIS, of our proposed framework discussed in Section 4.5 is responsible for the selection of an elastic controller to make scaling decisions. This implies that the overall performance of the framework is primarily dependent on the accuracy of the FIS and ultimately the corresponding design of its components, i.e. membership functions and fuzzy rules. The construction of membership functions is the primary step in the design of a fuzzy control system, which in our case consists of the definition of three input parameters and one output parameter (see Section 4.5.2 for more details). Amongst these four parameters, the *Workload* is based on the knowledge of domain experts initially introduced in [16]; the design of the *Controller* follows the guidelines in [54]; *Control error* is obtained using trial and error method; and *Response time* reflects the desired performance objective of a service provider.

The adopted design of the parameters mentioned above indicates that the following assumptions were made at the time of switching mechanism implementation: (1) The design ranges for the linguistic terms of *Workload* parameter adopted from Jamshidi et al. [16] is correct, near optimal and suitable to be applied in any application scenario. (2) The ranges for the linguistic terms of *Control error* parameter, obtained using the trial and error method, are correct and fits in any application scenario. (3) Each range of the output parameter (*Controller*) represents a discrete decision and therefore no overlapping was considered. This research explores the possibility to find near optimal parameter settings for the design of the membership functions without considering the aforementioned assumptions. The objective of this research work is twofold. Firstly, to introduce methods to derive suitable values for the switching mechanism parameters, i.e. *Workload*, *Control error* and *Controller*. Secondly, to explore the possibility to improve the obtained computational results by fine tuning the used parameters.

To address the above mentioned problem, this chapter employs two different approaches including the commonly used evolutionary approach and an alternative, less known approach called the Taguchi method. The evolutionary approaches based on the idea of natural biological evolution, where the survival of the fittest can be assured through natural processes such as randomly created population followed by reproduction and mutation [215]. They are best known for their ability to identify near optimal parameter settings from a large search space even in the absence of a precise description of the underlying problem [216]. The use of such techniques has successfully proven their suitability and has the potential to solve optimisation problems from a wide range of domains including their extended use in fuzzy systems as well. For example, the tuning membership functions for the regression problems [217, 218], inference engine [219, 220], simultaneous learning of knowledge base and rule base [221] etc. Alternatively, the Taguchi method is not a well-known generic optimisation technique. However, it provides a systematic and economical method to obtain parameter settings with relatively fewer trials [222]. The key reason behind the employment of Taguchi approach is the consideration of scenarios where a larger exploration of search space (as commonly required by GAs) is not feasible.

7.2 DEFINITION OF PROBLEM AND PARAMETERS

The role of a membership function in a fuzzy system is to represent the information contained in the fuzzy sets to helps in converting to/from crisp and fuzzy values. The membership function defines the degree of a crisp value in the range of 0 to 1 in accordance with a given linguistic variable. The membership functions are of different types, e.g. Triangular, Trapezoidal, Gaussian and Sigmoidal, etc. The choice of their selection is application dependent though [223]. In our existing settings, we have only used triangular and trapezoidal as shown in Figure 4.7. Each membership function, irrespective of the corresponding type, consist of the following three key ingredients: *Support, Boundary and Core (also known as Prototype)* [224]. Figure 7.1 represents these ingredients diagrammatically, whilst considering triangular and trapezoidal as the types of membership functions, whereas, their brief descriptions are as following [224]:

- Support: The X refers to the universe of a given fuzzy variable. Thus for a given fuzzy set A, *support* refers to that region of the universe X, where all elements $x \in X$ are characterised by $\mu_A(x) > 0$.
- Core/Prototype: For a given fuzzy set A, *core* refers to that region of the universe X, where all elements $x \in X$ are characterised by complete membership of set A, i.e $\mu_A(x) = 1$. The *prototype* is characterised by the same definition as *core* but with an exception that there is only one such element where $\mu_A(x) = 1$.
- Boundary: The *boundary* refers to that region of the universe X, where all elements $x \in X$ are characterised by $0 < \mu_A(x) < 1$.

In light of the above definitions, we are interested to find out the near optimal values of the above ingredients for each membership function shown in Figure

7.2. The figure is marked with unique labels in the format of α_n, θ_n and σ_n to represent a different point over the universe of a fuzzy variable. Each point

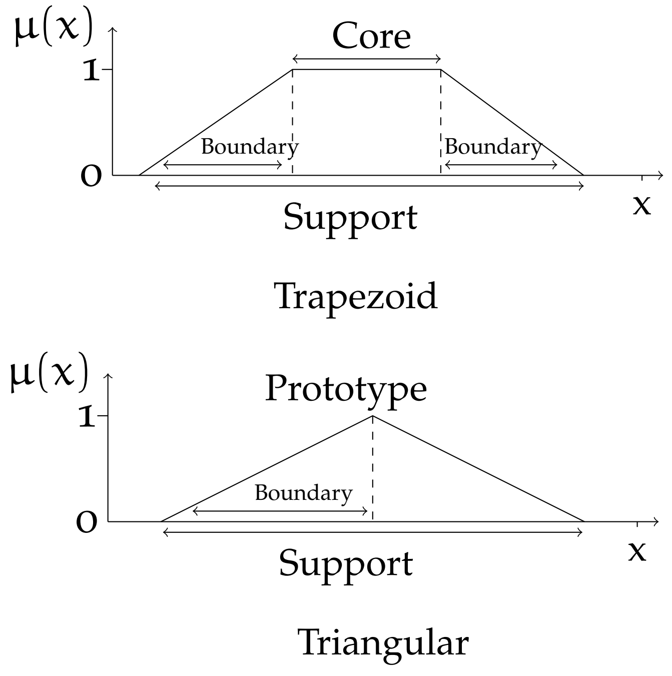


Figure 7.1: Ingredients of membership functions

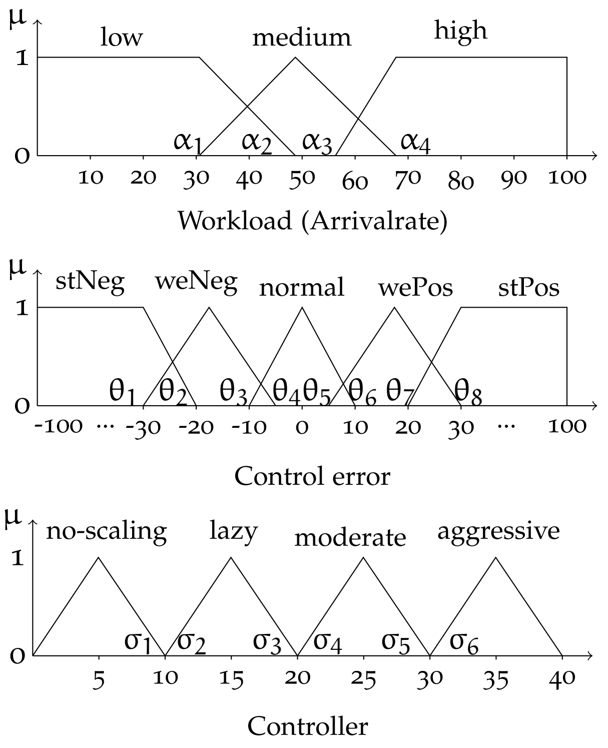


Figure 7.2: Membership functions

represents an independent parameter, and the collection of these parameters in a specific order defines the ingredients of a membership function, e.g. the length from α_1 to α_4 identifies the *support* region for *medium* membership function of the *Workload* variable. Similarly, the *core* and *boundaries* are also calculated from these parameters automatically. For example, the length from α_1 to α_2 defines the *boundary* for *low* and *medium* membership functions, whereas the length from 0 to the value of α_1 identifies the *core* for *low* membership function. In total, 18 parameters are shown in Figure 7.2. All these parameters are assigned with a range of values as shown in Table 7.1. These ranges are not fixed and can be reduced or increased whilst considering a particular application scenario.

Fuzzy Variable	Linguistic Terms							
	Start range — End range							
Control error	θ_1	θ_2	θ_3	θ_4	θ_5	θ_6	θ_7	θ_8
	-45 – -30	-35 – -20	-20 – -5	-15 – 0	0 – 15	5 – 20	20 – 35	30 – 45
Controller	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6		
	10 – 15	5 – 10	20 – 25	15 – 20	30 – 35	25 – 30		
Workload	α_1	α_2	α_3	α_4				
	15 – 35	30 – 50	50 – 70	60 – 80				

Table 7.1: The design range of each parameter

The employed methods, i.e. genetic algorithm and the Taguchi approach (explained in Section 7.3 and 7.4 respectively) operates to derive the near optimal combinations of the above parameters. However, during this process, each combination of the parameters shall satisfy the following constraints:

1. $\alpha_1 \leqslant \alpha_2$, $\alpha_3 \leqslant \alpha_4$ and
2. $\theta_1 \leqslant \theta_2$, $\theta_3 \leqslant \theta_4$, $\theta_5 \leqslant \theta_6$, $\theta_7 \leqslant \theta_8$ and
3. $\sigma_2 \leqslant \sigma_1$, $\sigma_4 \leqslant \sigma_3$ and $\sigma_6 \leqslant \sigma_5$.

These constraints ensure that every developing fuzzy partition should either be adjacent or overlapped with a neighbouring partition, hence confirming that every point x over the universe of a fuzzy variable must exist in at-least one of the *support* regions. Therefore, a generated solution during the evolution process will only be valid if it satisfies the constraints mentioned above. Invalid solutions are not considered for evaluation and will be replaced with a valid solution.

7.3 THE DESIGN OF MEMBERSHIP FUNCTIONS USING GENETIC ALGORITHM

The Genetic Algorithms (GAs) [215] are search mechanisms that are based on the idea of genetics and natural selection. They are iterative procedures and work on the population of individuals (also called chromosomes or solutions). The algorithm usually starts with a randomly generated population of solutions and aims to evolve towards better solutions by applying different genetic operations including natural selection, recombination and mutation. Such algorithms are considered very useful for problems that demand efficient search in the largely available problem space [225]. Moreover, GAs have the ability to identify near optimal parameter settings from a large search space even in the absence of a precise description of the underlying problem [216]. The following subsections explain the various details in the context of the underlying problem.

7.3.1 *Chromosome Encoding and Fitness Criteria*

The first stage of employing a GA for a problem is to represent the chromosome in such a way that it is suitable to be modified by the genetic operations. We have used the binary encoding and represent each parameter using a fixed (L) number of bits, which are concatenated to form the binary string of fixed size (66) bits as following:

$$\underbrace{\overbrace{b_1..b_L}^{\alpha_1}\,....\,\overbrace{b_1..b_L}^{\alpha_4}}_{Workload(16)}\;\underbrace{\overbrace{b_1..b_L}^{\theta_1}\,....\,\overbrace{b_1..b_L}^{\theta_8}}_{Controlerror(32)}\;\underbrace{\overbrace{b_1..b_L}^{\sigma_1}\,....\,\overbrace{b_1..b_L}^{\sigma_6}}_{Controller(18)}$$

The *Workload*, *Control error* and *Controller* are the fuzzy variables. The length of each parameter in the binary string is relative to the fuzzy variable. Thus, during the decoding process, the binary string is decomposed into three parts, whereas each parameter is decoded to get its corresponding value using the following equation adopted from [224],

$$C_i = C_{min_i} + \frac{b}{2^L - 1}(C_{max_i} - C_{min_i}) \tag{7.1}$$

C_i in the above equation represents a specific parameter i, e.g. α_1. C_{min_i} and C_{max_i} represent the minimum and maximum range for parameter i, b is the corresponding decimal value of the binary bits and L represents the number of bits length for parameter i.

After chromosome representation, the next step is to define the fitness function that quantifies the quality of each solution in the evolution process. Each solution represents different parameter settings of the underlying membership functions. We are interested to find out how a particular parameter settings contribute to the overall performance of an elastic policy. Thus, for the assessment of an elastic policy to quantify the quality of a solution, we consider the two-dimensional criteria described in Section 5.4. The two objectives of this evaluation criterion include the *Number of SLO violations* and *Cost*. Thus, whilst considering these objectives, the lower the values of *SLO violations* and *Cost*, the better the quality of a solution will be.

7.3.2 *Employing Multi-Objective GA with Adaptive Attributes*

The standard nondominated sorting algorithm-II (NSGA-II) [226] is utilised to tackle the underlying problem. The NSGA-II is a generic and commonly used standard algorithm to solve multi-objective optimisation problems. The NSGA-II starts as any other evolutionary approach with a population of competing solutions. It sorts and ranks every solution of the population regarding nondomination level followed by applying the genetic operators (selection, crossover and mutation) to create the offspring populations. The NSGA-II takes the union of parent and offspring populations followed by partitioning of the union in fronts. After this step, the NSGA-II applies a mechanism called crowding distance to enhance the spread and diversity of individuals followed by the step of elitism, where the population for the next generation is selected from the nondominated individuals thus improving the convergence [227]. In addition to the traditional NSGA-II, we have also utilised the following additional methods:

1. Adaptive population size: The benefits of employing adaptive population size approach are evident in the optimisation literature, e.g. increase in speed [228], escape in stagnation and diversity [229], etc. Algorithm 1 presents the pseudocode of the employed genetic algorithm in addition to the adaptive population size method. The lines from 6 to 12 in this

Algorithm 1 The pseudocode of adopted genetic algorithm with adaptive population size method [228]

```
1: Generate and initialise population
2: Evaluate every individual of the population
3: while not stopping criteria do
4:     Selection, Recombine and Mutate
5:     Evaluate every individual of the offspring
6:     if best fitness improved then
7:         Grow population size by rate $\triangle_1$
8:     else if no improvement for long period then
9:         Grow population size by rate $\triangle_2$
10:    else
11:        Shrink population size by rate $\triangledown$
12:    end if
13:    Evaluate new individuals
14: end while
```

algorithm represent the method of adaptive population size proposed by Eiben et al. [228]. This method suggests that the population size increases in the following two scenarios: (1) When fitness of the offspring population improves. This will make the algorithm biased towards further exploration of the population; (2) When fitness of the offspring population does not improve for a longer period of time. This helps the algorithm to get out of stagnation. Alternatively, the population size reduces, if both of the above conditions are not fulfilled. The growth ($\triangle_1$ and $\triangle_2$) and shrink rate ($\triangledown$) can be adjusted as per the needs. We use constant proportional rates of 10% for $\triangle_1$, 15% for $\triangle_2$ and 5 % for $\triangledown$. Regarding the bi-objective nature of the problem, the condition of *best fitness improved* holds true, if the offspring individual improves in either of the objectives.

2. Adaptive crossover probability: We have also employed the approach in [230] that computes the probabilities of genetic operations (crossover and mutation) on runtime per the fitness values of the current population. This technique helps maintain population diversity and strengthens convergence as well. According to such an approach, the probabilities increase when the GA ceases in local optimum and decrease when the individuals are well dispersed in the solution space. The actual values are derived using the following equation at runtime [230]:

$$p_c = \begin{cases} k_1(f_{max} - f')/(f_{max} - \bar{f}), & f' \geqslant \bar{f} \\ k_2, & f' < \bar{f} \end{cases} \tag{7.2}$$

f_{max} in the above equation represents the best fitness of the population, $\bar{f}$ the average fitness, and f' the best fitness among the parents that have to be recombined during crossover operation. The values of k_1 and k_2 are between 0 and 1. We use the value 1 for both k_1 and k_2 as per the recommendations in [230]. These values ensure that all those solutions must go through crossover operation, if their fitness is below or equal to the average fitness. The probability of crossover decreases for solutions where their fitness values are better than the mean fitness, and tends to zero as the fitness value reaches to the value of best fitness solution.

7.4 THE DESIGN OF MEMBERSHIP FUNCTIONS USING TAGUCHI METHOD

The Taguchi method provides a systematic and economical method to obtain parameter settings with relatively fewer trials by exploiting the use of orthogonal array [222]. Taguchi method was initially introduced for the quality improvement of manufacturing goods [231] and is more commonly used for the optimisation problems in research fields like machine design and electrical power systems [232, 233, 234]. Although Taguchi method has not received much attention as a generic approach for optimisation problems, there exist some proposals that demonstrate the use of Taguchi approach for optimising problems. Some examples of its use in various research disciplines include control theory [235, 236], Parameter tuning of optimisation algorithms [237, 238, 239], Task scheduling in cloud environment [240] and Fuzzy systems [241, 242], etc. Taguchi method has the inherent advantage of exploring the search space with significantly fewer trials. Therefore we employ its use to find parameter settings of the fuzzy membership functions considering scenarios where larger exploration (as commonly required by evolutionary approaches) are not possible or computationally expensive. This following subsections provides the basics and the various details of how we employed the Taguchi approach for the underlying problem.

7.4.1 *Preliminaries of Orthogonal Array*

The Orthogonal Array (OA) is a special set of Latin squares [243] used for experimental design in Taguchi method. The OA can be expressed as OA(N,k,s,t), where N represents the number of rows, k the number of columns, s the number of possible values of set S for columns and t the strength of OA. The columns of an OA are referred to as factors, whereas the values of columns are called as levels. Each factor represents an independent variable of an experiment, where levels are referred to all possible discrete values of a factor. Any matrix that consists of N rows and k columns with values obtained from a set S is referred to as an OA, if it satisfies the property of mutually balance. This means that, every sub-array of *N* rows and *t* columns must contains all possible combinations of values from set *S* [244].

In order to understand the mutual balance property with respect to strength *t*, consider an example of *OA(9,4,3,2)* array provided in Table 7.2. In this example, any column pair, e.g. *A* and *B* contains all combinations of levels for factors *A* and *B*, which can be seen from the table as (1,1), (1,2), (1,3), (2,1), (2,2), (2,3), (3,1), (3,2) and (3,3). Similarly, the property persist for other pairs of columns as well, such as (A and C), (A and D), (B and C), (B and D) and (C and D). Thus, the nine trials covers all possible combinations of levels for each pair of columns equally. The assignment of factors to columns are independent and can be assigned arbitrary to any column. The traditional factorial design approach in comparison will

Exp #	A	B	C	D
1	1	1	1	1
2	1	2	2	2
3	1	3	3	3
4	2	1	2	3
5	2	2	3	1
6	2	3	1	2
7	3	1	3	2
8	3	2	1	3
9	3	3	2	1

Table 7.2: An example of Orthogonal array represented as $L_9(3^4)$

require 81 (3^4) trials for the same example. Thus OA has the advantage to find the near optimal settings with smaller number of experiments because of it's mutual balance property that guarantee the full exploration of all possible combinations.

7.4.2 *Definition of Factors, Levels and Orthogonal Array*

A factor represents an independent variable (or parameter) of an experiment, whereas levels are referred to all possible discrete values of a factor. In the context of the underlying problem, each parameter of the membership functions earlier explained in Section 7.2 is considered as a separate factor. Thus all 18 parameters become factors here. Regarding the number of levels per factor, to the best of our knowledge, there is no standard principle available. The general guidelines are, the higher numbers of levels increases the chances of obtaining better parameter settings [244]. However, increasing the number of levels increase the size of number of trials as well that indicates a trade-off between the number of levels and the size of orthogonal array. Thus, a suitable orthogonal array is required to maintains the best balance by using the largest possible levels with fewer trails [242].

The formal construction of an orthogonal array for a particular problem depends on the total Degree of Freedom (DOF) that informs the minimum number of trials needed to analyse the selected factors [245]. The DOF of a factor is the number of levels for that particular factor minus one and the total DOF is the sum of all factors' DOF plus one. The 1 DOF is associated with the mean irrespective of the number of factors to be considered for the underlying problem [245]. Thus the minimum number of trials in an orthogonal array must be at-least equal to the total DOF. The following equation depicts this constraint,

$$N \geqslant 1 + \sum_{i=1}^{k} (A_i - 1) \tag{7.3}$$

The k in above equation represents the number of factors, where A_i indicates the number of levels for i^{th} factor. The total factors are 18 and the number of levels for each factor can be obtain by solving Equation 7.4 that is itself derived from Equation 7.3 [242].

$$A \leqslant \frac{N-1}{18} + 1 \tag{7.4}$$

where N is usually the square of a number as following

$$N = m^2, m \in \{2, 3, 4, 5.....n\} \tag{7.5}$$

Solving equation 7.4 produce 17 that represents the smallest integer value for m that satisfies the constraint $17 \leqslant \frac{N-1}{18} + 1$. Thus selecting 17 as the number of levels, we divide the design range (as shown in Table 7.1) of each respective parameter into 17 equally spaced levels. The snapshot of corresponding factors and levels relationship is shown in Table 7.3.

	L1	L2	L3	L4 — L14	L15	L16	L17
α_1	15	16.25	17.50	——	32.50	33.75	35
:	::	::	::	——	::	::	::
α_4	60	61.25	62.50	——	77.50	78.75	80
θ_1	-45.0	-44.06	-43.12	——	-31.88	-30.94	-30.00
:	::	::	::	——	::	::	::
θ_8	30.0	30.94	31.88	——	43.12	44.06	45.00
σ_1	10.0	10.31	10.62	——	14.38	14.69	15.00
:	::	::	::	——	::	::	::
σ_6	25.0	25.31	25.62	——	29.38	29.69	30.00

Table 7.3: Design factors and levels relationship

Solving equation 7.3 results in 289 as the total DOF by considering 18 factors in total. Thus, any valid orthogonal array with $N \geqslant 289$ is appropriate for the underlying problem. The next step is to identify the suitable orthogonal array and we fortunately found the OA $L_{289}(17^{18})$ from the freely available repository [246]. Thus forming ($R = 289$) as the total number of experiments (or trials or individuals), where each trial represents a different parameter settings and it shall obey the constraints mentioned in Section 7.2. If any trial does not satisfy the constraints then that particular parameter setting is not valid and will be discarded. Thus, we first checked all the trials of $L_{289}(17^{18})$ OA and found that only *115* of them are valid as per the constraints. Hence we only left with a search space of *115* trails. A snapshot is provided in Table 7.4.

7.5 COMPUTATIONAL RESULTS

One of the advantages of using the employed Taguchi method is that it does not require the implementation of any additional algorithm or the usage of any sophisticated software. On the other hand, for GA experiments, we have utilised NSGA-II algorithm by integrating a well-known Java based optimisation library called jMetal [247] into our existing experimental framework (explained in Section 5.1).

To demonstrate the effectiveness of both employed methods, we perform the following two kind of experiments: (1) We carried out an exploratory experiment using both methods considering a small scale training workload scenario. In the case of GA, we used the extended experimental framework (i.e. proposed elasticity framework and the employed GA algorithm) to find out the Pareto front of solutions that are best in either of the two considered objectives, i.e. SLO violations and Cost. Whereas, in the case of Taguchi approach we have evaluated all the trials, i.e. *115* using the proposed framework and records the output of both objectives, i.e. SLO violations and Cost for each trial. (2) We select the best individuals regarding each objective, for both methods (i.e. from the Pareto front in the case of GA and from the results of all trials in the case of Taguchi method). We perform real experiments, using our proposed framework considering the four selected individuals and some large scale test workload scenarios, to analyse the effectiveness of both employed techniques.

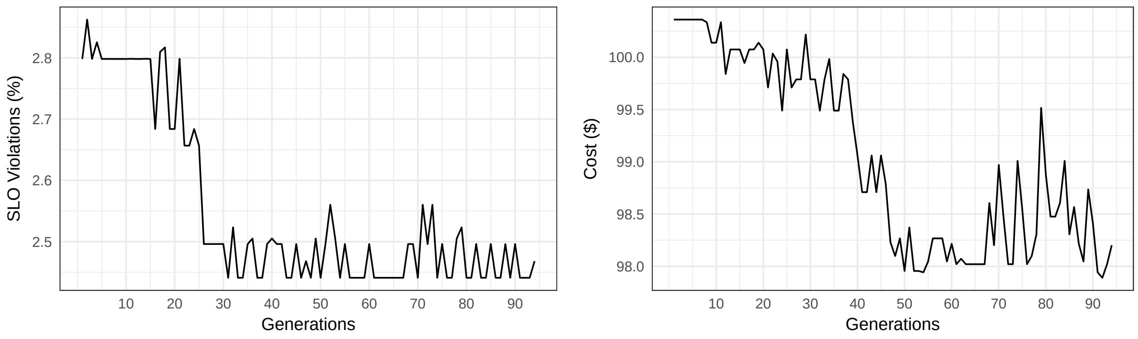
To conduct both experiments, we have used the same experimental settings, i.e. gains for controllers and fuzzy rules, which we have used for the experiments performed in Chapter 5. It is important to mention that we are only interested to find out parameter settings for the utilised membership functions and not for the

Table 7.4: Snapshot of valid trials $L_{115}(17^{18})$

Exp #	α_1	α_2	α_3	α_4	θ_1	θ_2	θ_3	θ_4	θ_5	θ_6	θ_7	θ_7	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6
1	15	30	51.25	77.5	-40.31	-27.5	-14.38	-12.19	5.62	12.5	24.69	44.06	11.88	7.81	20.94	15.94	33.44	29.69
2	15	32.5	61.25	76.25	-45	-35	-15.31	-13.12	7.5	10.62	21.88	37.5	13.75	7.19	22.19	19.69	35	26.56
:	::	::	::	::	::	::	::	::	::	::	::	::	::	::	::	::	::	::
30	20	41.25	51.25	60	-43.12	-35	-10.62	-2.81	8.44	9.69	27.5	43.12	15	5.31	21.88	18.75	32.5	29.06
31	20	45	50	70	-41.25	-21.88	-20	-11.25	10.31	17.19	24.69	41.25	13.75	6.25	24.06	16.88	32.19	25.94
:	::	::	::	::	::	::	::	::	::	::	::	::	::	::	::	::	::	::
60	25	50	55	77.5	-33.75	-22.81	-13.44	-0.94	10.31	14.38	27.5	32.81	12.5	7.19	21.56	17.19	32.81	25.62
61	26.25	31.25	60	80	-42.19	-31.25	-13.44	-2.81	15	20	24.69	36.56	13.44	9.06	23.12	15.62	35	30
:	::	::	::	::	::	::	::	::	::	::	::	::	::	::	::	::	::	::
90	30	42.5	51.25	75	-41.25	-28.44	-18.12	-6.56	1.88	19.06	22.81	32.81	13.44	5.94	22.5	19.38	34.06	26.56
91	30	43.75	62.5	73.75	-37.5	-29.38	-14.38	-8.44	7.5	7.81	28.44	39.38	12.5	9.06	23.44	18.75	30.94	25.94
:	::	::	::	::	::	::	::	::	::	::	::	::	::	::	::	::	::	::
114	35	46.25	55	62.5	-41.25	-25.62	-16.25	-13.12	0	9.69	32.19	34.69	10.31	9.06	22.81	20	33.75	29.69
115	35	50	66.25	78.75	-45	-33.12	-18.12	-9.38	12.19	18.12	24.69	30	10.94	6.56	21.25	18.75	30	26.25

gains of the employed controllers or fuzzy rules. We have used the four hours trace of *Wikipedia* workload as the training workload scenario for the experiments.

In the case of Taguchi approach we have evaluated all the trials, i.e. *115* using the proposed framework and records the output of both objectives, i.e. SLO violations and Cost for each trial. Whereas, in the case of GA, the experiment consists of a full evaluation, i.e. the execution of GA for the given test scenario. In each iteration, the individuals represent the different parameter settings of the membership functions (shown in Figure 7.2) that have to be evaluated regarding the number of *SLO violations* and *cost* by the proposed elasticity framework. Figure 7.3 display the summary of best solution value regarding each objective across all generations. This demonstrate that each passing generation explore better solution regarding *SLO violation* objective until *Generation no:32*, whereas, in the case of *Cost* objective, the improvement can be seen until *Generation no:50*.



(a) SLO Violations (b) Cost)

Figure 7.3: GA performance summary per generation

The Pareto front obtained is shown in Figure 7.4, which displays all those individuals that are best regarding either of the two objective amongst the solutions evaluated during the GA evolution process. The selection of an individual to be used for actual execution is a design decision and is based on the preference regarding the objectives. However, for the sake of analysis, we select the following two individuals: (1) *SLO wise best* - the solution with the lowest number of *SLO*

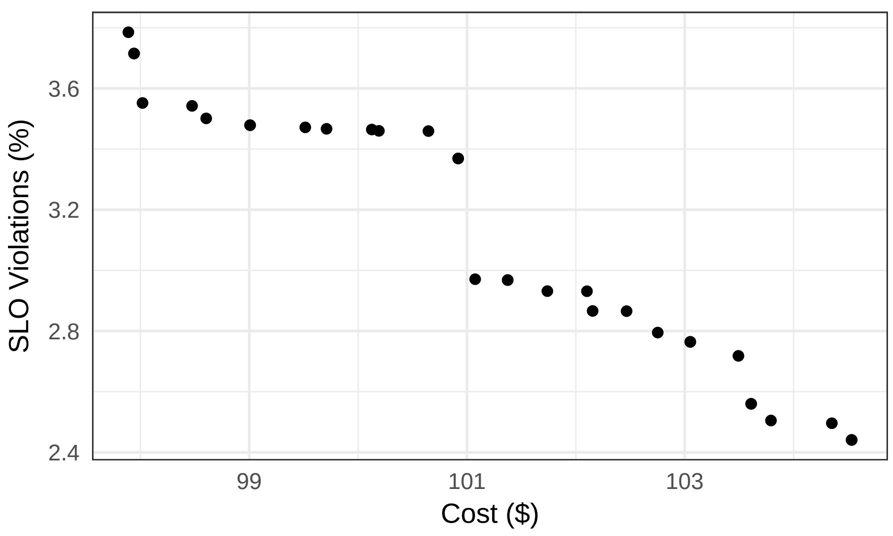


Figure 7.4: The Pareto front of *Wikipedia* workload scenario

violations. (2) *Cost wise best* - the solution with the lowest operational *cost*. Similarly, from the Taguchi results of *115* trials, we select two trials where each trial is best regarding each objective.

Using the four selected solutions, we have carried out experiments for *Cyclic* and *Slowly varying* workload scenarios (shown in Figures 5.2e and 5.2f respectively) that both utilise the *Wikipedia* trace of 24 hours duration. The key aim of these experiments is to evaluate the performance of the parameter settings of the four selected individuals to that of the parameter settings used in Chapter 5 experiments. The results obtained from Chapter 5 experiments are shown in Table 7.5, whereas the results obtained from the selected four experiments are summarised in Tables 7.6 and 7.7. Following the approach used in [225, 248], we calculate the Relative Percentage Deviation (RPD) using Equations 7.6 and 7.7, to comparatively assess the quality of results obtained using existing and new parameter settings.

$$\mathrm{RPD}_{\mathrm{slo}} = \left(\frac{\mathrm{New}_{\mathrm{slo}} - \mathrm{Existing}_{\mathrm{slo}}}{\mathrm{Existing}_{\mathrm{slo}}}\right) * 100 \tag{7.6}$$

$$\mathrm{RPD}_{\mathrm{cost}} = \left(\frac{\mathrm{New}_{\mathrm{cost}} - \mathrm{Existing}_{\mathrm{cost}}}{\mathrm{Existing}_{\mathrm{cost}}}\right) * 100 \tag{7.7}$$

The RPD value of each objective specifies the percentile change in comparison with the existing results. A positive RPD value shows the percentage degradation of the quality in comparison with the existing results, whereas a negative RPD value represents the percentage improvement in the quality. This implies that the lower the value of RPD, the better the quality of results in comparison to the existing results.

Workload Scenario	Cost ($)	SLO Violations (%)
Cyclic	766.70	1.075
Slowly varying	863.40	0.89

Table 7.5: Results obtained using existing membership functions (Figure 4.7)

Scenario	GA				Taguchi			
	SLO	RPD_{slo}	Cost	RPD_{cost}	SLO	RPD_{slo}	Cost	RPDcost
Cyclic	0.98	-9.26	799.17	4.24	1.1	1.85	784	2.26
Slowly varying	0.76	-14.61	897.61	3.96	0.81	-8.99	881.32	2.08

Table 7.6: Results obtained using the parameter settings of GA and Taguchi approaches considering *SLO wise best* individuals

Scenario	GA				Taguchi			
	Cost	RPD_{cost}	SLO	RPD_{slo}	Cost	RPD_{cost}	SLO	RPDslo
Cyclic	753	-1.79	1.33	23.15	755.84	-1.42	1.42	31.48
Slowly varying	842.37	-2.44	1.24	39.33	840.85	-2.61	1.35	51.69

Table 7.7: Results obtained using the parameter settings of GA and Taguchi approaches considering *Cost wise best* individuals

Table 7.6 presents the results of both workload scenarios obtained using the parameter settings of *SLO wise best* individuals. Focusing on the highlighted columns in the GA part, it is evident that the results in both workload scenarios are improved, i.e. the number of *SLO violations* are dropped by 9.26% and 14.61%

for *Cyclic* and *Slowly varying* scenarios respectively. However, the results obtained using Taguchi approach indicate an improvement of 9% in the case of *Slowly varying* scenario, whereas no improvements are observed in the case of *Cyclic* scenario.

Table 7.7 presents the results of both workload scenarios obtained using the parameter settings of *Cost wise best* individuals. Focusing on the highlighted columns in this table demonstrate minor improvement regarding *Cost* objective using both approaches, i.e. GA and Taguchi. However, this minor improvement is achieved at the expense of performance degradation. Therefore, careful attention is required to prioritise the objective when selecting the individual (parameter settings) from the results of Taguchi trials or Pareto front.

Comparing the results of both employed approaches considering *SLO wise best* scenario indicate that the GA method has higher potential to search better parameter settings than Taguchi. However, the results of *Cost wise best* scenario do not demonstrate a clear difference. An intuitive explanation for this is the consideration of small scale training set that only consist of 4 hours of data. Increasing the size of the training data set could provide more insights about the employed techniques. However, considering larger training data set also increase the computation required for the GA evolution process. This is the key reason of employment two different techniques to address the same problem. We recommend the use of GA, when possible. Otherwise, the employed Taguchi approach is an economical method to set-up (or explore better parameter settings) in the absence of experts knowledge.

Part III

CONCLUSIONS

8 CONCLUSIONS

This chapter concludes the thesis and discusses the potential future work. It provides a summary of the research undertaken, reviews the contributions made, and discusses how they were addressed. Finally, limitations of the research presented here and possible future research directions are discussed.

8.1 DISCUSSION

The proposed elasticity methods are developed using control-theoretical based multiple controllers and a fuzzy control system. This synergy, on the one hand, enables us to address the inherent uncertainty related issues of a cloud environment using the fuzzy control system. On the other hand, the systematic design of model-based feedback controllers helps in strengthening the reliability of the system. In this thesis, we chose to address the cloud elasticity from a SPs perspective. The key motivations behind this choice are that the cloud based applications are subject to varying workload conditions, and the CPs lack control and visibility regarding application performance aspects that make it difficult to perform efficient scaling decisions [249]. In contrast, the SPs have full control and visibility of cloud resources using monitoring and management APIs provided by CPs, as well as an up-to-date knowledge of an application status using custom or 3rd party tools. The proposed methods are hybrid in nature, and consider an application level metric (Response time) as well as a system level metric (CPU utilisation). Additionally, we consider the *Arrival rate*, which represents the incoming workload intensity level into the decision making process. The consideration of these three parameters empower the proposed methodology to make an informed scaling decision, as opposed to the majority of the existing related approaches that either rely on application level [171, 16] or system level metrics [167, 33, 110].

The existing *Rule-based* solutions in general are prevalent due to their intuitive, simplistic and, more importantly, commercial availability factors [4]. Such approaches [26, 27, 28, 29, 30] are easy to design and well understood by the system designers and administrators alike. However, such methods lack a formal systematic design process as they are designed based on previous experiences or apply a trial and error approach [4, 95]. Moreover, they are criticised for the difficulty in setting-up various thresholds of the rules and their inability to cope with the changing environment behaviour [16, 45]. This is evident from the configurations and results of the *RightScale* approach discussed in Chapter 5.

The feedback control solutions [31, 114, 32, 33, 110, 34, 113] follow the fixed gain design principle of control theory. Such fixed gain methodologies in general work well for systems that are subject to stable or slowly varying workload conditions [95]. However, due to the lack of adaptive behaviour at runtime, the performance suffers in scenarios where the operating conditions change quickly or when the environmental conditions and configuration spaces are too wide to be explored effectively [44]. The lack of adaptivity issue has been addressed by incorporating a kind of runtime adaptation mechanism using online learning algorithms such as the use of linear regression [122], optimisation [131], Kalman filter [135] and reinforcement learning [134]. In general, such adaptive control methodologies have the ability to modify themselves to the changing behaviour in the system environment that make them suitable for systems with changing workload conditions. However, they are also criticised for the additional computational cost caused due to the online learning [45], their associated risk of reducing the quality assurance of the resulted system, and the impossibility of deriving a convergence or stability proof [44]. Moreover, they are unable to cope with sudden changes in the workloads.

Al-Shishtawy and Vlassov [159] addressed the elasticity problem using a two-level approach, where they utilised a combination of an MPC based feedforward control solution and a PI based feedback control method. Using such an approach, the feed-forward method follows a predictive approach that takes scaling decisions for a longer time in advance; whereas the feedback method is responsible for making gradual changes in a reactive style. Such two-step hybrid control solutions are effective; however, currently our focus is on the efficiency of elastic solution implemented at the 2$^{\text{nd}}$ level that follows a reactive strategy. Al-Shishtawy and Vlassov [159] utilised a fixed gain PI feedback controller that suffers from various issues discussed earlier in this section, whereas the approach adopted in this thesis uses multiple fixed gain controllers. Wang et al., Kjaer et al. [160, 161, 162] followed a similar approach, i.e. the combination of feed-forward and feedback. However, they focused on vertical elasticity (see Section 3.3.3.1 for more details).

The following proposals have also adopted a similar approach as employed in this thesis. For example, Grimaldi et al. [167] used a PID gain scheduling. Their gain scheduler is an optimal controller that derives the gains using an optimisation based tuning procedure. The key issues of such an approach are similar to that of an adaptive methods discussed earlier in this section. Saikrishna et al., Qin and Wang, and Taneli et al. [168, 169, 170] followed a Linear Parameter Varying (LPV) approach. CPU utilisation is considered as the single scheduling parameter by Saikrishna et al. [168], whereas Qin and Wang, and Taneli et al [169, 170] rely on arrival rate and service rate. Patikirikorala et al. [171] followed a MMST based control solution. Their method use two different operating regions and consist of two different fixed gain controllers with an if-else switching that is based on *Response time* only. Saikrishna et al. [172], in contrast, used ten distinct operating regions and *Arrival rate* as a switching signal.

Jamshidi et al. [16, 35] highlighted the uncertainty related issues and the idea of qualitative elasticity rules using a fuzzy control system to address the issues of *Rule-based* approach. The inputs to their method consist of *Arrival rate* and *Response time*, whereas the output is the number of VMs to be added or removed.

Their approach facilitates a dynamic response based on the aforementioned two parameters by making a scaling decision with different intensity level, and consequently it helps avoid the static scaling issue of the *Rule-based* approaches. However, the output (number of VMs) are a pre-defined range of constant integers, and it is not clear how these numbers are set-up. Therefore, it creates similar problems to that of the *Rule-based* approach, i.e. difficulty in setting-up threshold values of rules and lack of a well-founded design approach. On the other hand, machine learning based control solutions that utilise either reinforcement learning [21, 39] or neural networks [180, 250] provide high levels of flexibility and adaptivity. However, such flexibility and adaptivity come at the cost of long training delays, poor scalability, slower convergence rate, and the impossibility of deriving stability proof [44, 45, 79, 251].

It is concluded from the above discussion that different elastic controllers due to their underlying implementation techniques have different pros and cons, hence there is no best solution and the choice of selecting suitable approaches depends on the requirements [44]. The research work carried out in this thesis advocates the idea of a fixed-adaptive approach (also referred to as hybrid by Gambi et al. [44]) in contrast to either completely fixed or fully adaptive methods. The proposed elastic methodologies are implemented using the combination of the model-based control-theoretical approach and the knowledge based fuzzy control system. This combination, in comparison with the existing fixed-adaptive methods [167, 168, 169, 170, 171, 172], addresses the uncertainty related issues and enables us to provide qualitative elasticity rules as well.

8.2 THESIS SUMMARY

This thesis investigates the horizontal elasticity problem from the CPs perspective and proposes biologically-inspired auto-scaling solutions. The proposed elastic methods follow a *Reactive* triggering approach, target *Web applications*, and aim to maintain the desired performance level whilst reducing operational cost. The proposed methods are implemented using a *Control theoretical feedback technique* and a *Fuzzy control system*. The motivation behind this research work, thesis statement and the research contributions is presented in the first chapter. The basic concepts and terminologies related to the research work undertaken are introduced in Chapter 2 to provide the foundations for the research domain. This includes a high-level overview of cloud computing explaining its historical and technical perspective. This is followed by a detailed technical introduction of cloud elasticity including a broad overview, its working mechanism, its various characteristics and the commonly known issues associated to it.

Chapter 3 presents a state-of-the-art review including methods that are developed using control-theoretical techniques, as the proposed elastic methods follow the same techniques. whilst conducting the state-of-the-art review, it is observed that there is no specific review paper that has particularly focused on control-theoretical approaches. Such a review paper could help the interested researchers to understand the control-theoretical point of view and facilitate comparison of the closely related approaches. Thus, for the sake of completeness, Chapter 3 also provides a brief overview of all review papers that are related to this thesis. Furthermore, a novel taxonomy is introduced which consists of characteristics from the control-theoretical point of view (i.e. as an implementation technique) as well as from a cloud elasticity perspective (i.e. as an application domain). The state-of-the-art review is conducted in light of the proposed taxonomy where existing elastic solutions are clustered, based on the following types of controllers: *Basic*, *Optimal*, *Advanced* and *Intelligent*. Furthermore, various open issues and challenges are also highlighted and discussed.

Chapter 4 starts with a discussion based on some of the identified issues and challenges in Chapter 3 to formulate the requirements for the new elastic control solution. The proposed control solution (referred to it as *Hard switching*) consists of the simultaneous use of multiple controllers, where each controller is particularly designed to achieve better performance in a different situation. The selection of a suitable controller is realised at runtime using an additional fuzzy control solution. This thesis adopts three different controllers termed as *Lazy*, *Moderate* and *Aggressive* to demonstrate the effectiveness of the proposed control solution. Each controller is responsible for computing the scaling decision at a different levels of intensity as their name specifies. Chapter 4 discusses the design of individual components, including the feedback control and the fuzzy system based switching mechanism. The feedback control part consists of the identification of various control elements such as system input, output and reference input. The fuzzy system based switching mechanism considers three different inputs including *Arrival rate*, *Response time* and *Control error*. The output of the fuzzy control system is one of the multiple employed controllers mentioned above.

In Chapter 5, we discuss the evaluation of the proposed *Hard switching* methodology in comparison with related benchmark methods. The experimental environment is developed using Java programming language and it further integrates a well-known cloud simulation framework called *CloudSim* [55], a fuzzy logic library called JFuzzylogic [196], and an optimisation library called jMetal [247]. The benchmark methods employed for comparison purposes include the commercially available *Rule-based* methodology called *RightScale* [27] and the three employed fixed gain controllers. Furthermore, we adopt a bi-objective criterion consisting of the number of *SLO violations* and *Cost*. Moreover, a set of different types of commonly used workload patterns including *Dual-phase*, *Tri-phase*, *Cyclic*, *Slowly varying*, *Quickly varying*, *Large variation* and *On-off* are identified and adopted.

The analysis of the obtained computational results demonstrates that the *Hard switching* method produces better results for each employed workload scenario in comparison to other benchmark methods. However, in general such methods are also criticised for an associated unwanted behaviour, termed as bumpy transitions, that could lead the system to an oscillatory state. The results obtained using the *Hard switching* method demonstrate the presence of oscillation only in the case of an *On-off* scenario, and few occurrences of bumpy transitions in the case of *Dual-phase* and *Tri-phase*. To avoid such occurrences, a *Soft switching* approach is introduced in Chapter 6. Such an approach allows the possibility of selecting more than one controller for the decision making. The development of the proposed *Soft switching* approach is inspired by the central switching mechanism of action selection in animal brains implemented by a group of subcortical nuclei collectively referred to as BG [207, 208].

The proposed *Soft switching* approach depicts each controller as a separate action and extends the existing *Hard switching* framework by incorporating a computational model [57, 58] of action selection that is based on the functional anatomy of BG. This model accepts three salience input signals and produces output signals. Each input signal represents the selection strength of each employed controller. These signals are generated using the modified form of the fuzzy control system that is used as a switching mechanism in the case of the *Hard switching* approach. The final output of the *Soft switching* method is computed using the individual output of the controllers and the output signals returned by the BG computational model. Similar to the *Hard switching* method, the *Soft switching* approach is evaluated by conducting experiments using all the adopted workload patterns. The obtained computational results are compared with the results of the *Hard switching* approach. It is observed that the *Soft switching* approach produces a lower number of *SLO violations* in each workload scenario compared to *Hard switching*. Moreover, in the case of *Soft switching*, there are no oscillations observed for the *On-off* scenario and the intensity of bumpy transitions in *Dual-phase* and *Tri-phase* scenarios are also reduced.

In chapter 7, we formulate the construction of the fuzzy membership functions as a multi-objective optimisation problem to explore the near optimal parameter settings of their design. This problem is addressed using two different techniques including the commonly used GA and an alternative, less known approach called the *Taguchi* method. The employment of a GA approach is motivated due to its ability to identify near optimal parameter settings from a large search space even in the absence of a precise description of the problem. In contrast, the Taguchi method provides a systematic and economical method. The Taguchi approach can be used in scenarios where larger exploration (as required by GA) is not feasible. The obtained computational results demonstrate that GA is comparatively more effective and is a preferable choice to be adopted, when possible.

8.3 SUMMARY OF CONTRIBUTIONS

This thesis explores the cloud resource management issue with a focus on the dynamic resource provisioning (elasticity) feature of cloud computing. The thesis investigates the current state-of-the-art elastic methodologies, highlights the open issues and proposes new elastic methods to contribute towards resolving the identified issues. The summary of the key contributions of this thesis is as follows:

1. **A control-theoretical review of cloud elasticity:** Chapter 3 proposes a novel taxonomy to carry out the state-of-the-art review of existing control-theoretical based auto-scaling solutions. This taxonomy integrates attributes from control-theoretical (i.e. as an implementation technique) as well as from cloud elasticity (i.e. as an application domain) perspectives. The existing auto-scaling solutions are reviewed and grouped, based on the various types of control methodologies. Such a classification helps to understand, analyse and compare the closely related approaches. Moreover, various open issues and challenges are highlighted and discussed. The content of Chapter 3 has been submitted to the peer review journal ***Cluster Computing - Springer*** and currently is in the review stage. This review paper aims to fill the gap in the existing cloud elasticity literature where the existing reviews focused mostly on elasticity based classification.

2. **Biologically-inspired hard switching approach:** Chapter 4 discusses the proposal of a novel biologically-inspired elastic method focusing on the problem of horizontal cloud elasticity from the SPs perspective. The pro-

posed control solution (referred to it as *Hard switching*) consists of the simultaneous use of an array of controllers, whereas the selection of a suitable controller is realised at runtime using an additional fuzzy control solution. The simultaneous use of multiple controllers helps to achieve the adaptive behaviour using switching among controllers, whereas the use of a fuzzy control system helps to address the inherent uncertainty related issues of the cloud environment. The preliminary results obtained using *Hard switching* are accepted for publication in the peer review journal named ***International Journal of High Performance Computing and Networking*** and is currently in printing stage.

3. **Biologically-inspired soft switching approach:** Chapter 6 discusses the proposal of a biologically-inspired soft switching mechanism that extends our initially proposed *Hard switching* approach. Such an approach allows the possibility of selecting more than one suitable controller for the final decision making. The soft switching approach improves the possibility of bumpless transition and to avoid oscillatory behaviour. The preliminary results obtained using this approach are published in the peer review journal ***Cognitive Computation - Springer*** [252].

4. **Design of fuzzy membership functions using GA:** Chapter 7 discusses the design of the employed membership functions used for the switching mechanism of our proposed *Hard switching* framework, as a parameter settings optimisation problem. This chapter further proposes the use of a multi-objective GA algorithm to obtain near optimal parameter settings. This helps to design the employed membership functions in the absence of experts knowledge and without considering any assumptions. The preliminary results obtained using this research work is published in the conference proceedings of ***31st Annual ACM Symposium on Applied Computing (SAC 2016)*** [253].

5. **Design of fuzzy membership functions using Taguchi approach:** Chapter 7 also discusses the design of the employed membership functions using an

alternative, less known approach called the Taguchi method. Using such an approach, the design problem of membership functions is addressed with the use of an orthogonal array. The computational results obtained using this research work has been submitted to a peer review journal named ***Computing - Springer*** and is currently in the review process.

6. **Performance evaluation:** Chapter 5 and 6 present the computational results obtained using the proposed *Hard switching* and *Soft switching* methodologies in comparison with the state-of-the-art approaches. The experiments for evaluation purposes are conducted utilizing real HTTP traces that followed certain workload patterns. The results are compared following an evaluation criteria consisting of attributes like performance, cost, and oscillatory behaviour.

8.4 LIMITATIONS AND FUTURE WORK

The computational results presented in this research work demonstrated the effectiveness of the proposed methodologies. However, it has some limitations that are important to be highlighted, as they are out of the scope of our existing research. These limitations signify the directions for potential future research that will pave the way to extend the proposed methodologies. These are highlighted as following:

1. The cloud computing model provides a pool of unlimited computational resources in the form of VM offerings with the VMs themselves of different computational specifications. In the research work presented in this thesis, we only consider VMs with the same computational specifications (commonly referred to as homogeneous servers). Renting homogeneous servers is not always pragmatic. However, considering heterogeneous servers for horizontal elasticity creates challenges for building efficient and accurate performance models because of their different computational capabilities.

Thus a further investigation is required to find the efficiency of our proposed methodology whilst considering heterogeneous servers.

2. The elastic controller is responsible for taking a scaling decision. However, its execution is not spontaneous. The elastic controller interacts with the service of a CP to initiate its execution. This process may take several minutes before the new VMs are set-up and ready to accept the load of the running application. Thus, the delay between the decision and the actual execution could cause some problems for the underlying application. Such delays and possible problems are not considered in this thesis. Moreover, situations like failure of a VM at runtime are also not considered. However, the cloud environment is subject to such uncertainties, and the considerations of these aspects are considered as a potential future work.

3. The proposed control methodologies are evaluated on a small scale, i.e. using various workloads spanning several hours or even days in some cases and a cluster size of up to 80 VMs. However, the suitability of the control methodologies at a larger scale whilst considering realistic enterprise level web applications, e.g. Animoto which saw cluster size swing from 50 to 4000 VMs [19], should be further evaluated.

4. The proposed auto-scaling solution is evaluated in a controlled setting using a simulation environment. However, to demonstrate its effectiveness and to be able to use it in a practical setting, the implementation and evaluation of the proposed methods need further investigation in a real cloud environment.

5. The proposed control system is evaluated against the benchmark methods using evaluation criteria consisting of performance and operational costs. However, a detailed theoretical convergence and stability analysis to formally evaluate the proposed approach against other state-of-the-art approaches are not performed. Therefore, the proposed methods require

further investigation from a formal evaluation perspective that focuses on theoretical convergence and stability analysis.

6. The proposed control system in this thesis follows a reactive triggering approach. However, such an approach is criticised for the delay caused between the time of reaction to a change, and the time taken to complete the reconfiguration process. The hybrid approach, in contrast, combines both the predictive and reactive approaches to one elastic method. A few such elastic methods are discussed earlier in Section 3.3.3.1, where a feedforward and feedback approach are working together as a hybrid approach. We argue that such an approach is more effective in comparison to either reactive or proactive approaches. Therefore, extending the proposed control solution to make it a hybrid methodology of the aforementioned type is one way to advance forward.

BIBLIOGRAPHY

[1] S. Johnston, "Cloud Computing," 2011. [Online]. Available: http://tinyurl.com/yc9tgeye

[2] M. Armbrust, I. Stoica, M. Zaharia, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, and A. Rabkin, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, p. 50, 2010.

[3] Amazon, "Web Application Hosting," 2017. [Online]. Available: https://tinyurl.com/aws-webHosting

[4] H. Ghanbari, B. Simmons, M. Litoiu, and G. Iszlai, "Exploring alternative approaches to implement an elasticity policy," in *Cloud Computing (CLOUD), 2011 IEEE International Conference on*. IEEE, 2011, pp. 716–723.

[5] X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, P. Padala, and K. Shin, "What does control theory bring to systems research?" *ACM SIGOPS Operating Systems Review*, vol. 43, no. 1, pp. 62–69, 2009.

[6] A. Filieri, M. Maggio, K. Angelopoulos, N. D'Ippolito, I. Gerostathopoulos, A. B. Hempel, H. Hoffmann, P. Jamshidi, E. Kalyvianaki, C. Klein, F. Krikava, S. Misailovic, A. V. Papadopoulos, S. Ray, A. M. Sharifloo, S. Shevtsov, M. Ujma, and T. Vogel, "Software Engineering Meets Control Theory," in *Proceedings - 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2015*, 2015, pp. 71–82.

[7] M. T. Prescott, "Action Selection," 2008. [Online]. Available: http://www.scholarpedia.org/article/Action{_}selection

[8] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for Cloud computing," *Future Generation Computer Systems*, vol. 28, no. 5, pp. 755–768, 2012.

[9] Z.-H. Zhan, X.-F. Liu, Y.-J. Gong, J. Zhang, H. S.-H. Chung, and Y. Li, "Cloud computing resource scheduling and a survey of its evolutionary approaches," *ACM Computing Surveys (CSUR)*, vol. 47, no. 4, p. 63, 2015.

[10] D. Fernández-Cerero, A. Fernández-Montes, L. G. Abril, J. A. Ortega, and J. A. Álvarez, "Fear Assessment: Why data center servers should be turned off." in *CCIA*, 2014, pp. 253–256.

[11] X. Zhu, L. T. Yang, H. Chen, J. Wang, S. Yin, and X. Liu, "Real-time tasks oriented energy-aware scheduling in virtualized clouds," *IEEE Transactions on Cloud Computing*, vol. 2, no. 2, pp. 168–180, 2014.

[12] L. Yu, T. Jiang, Y. Cao, and Q. Qi, "Carbon-aware energy cost minimization for distributed internet data centers in smart microgrids," *IEEE Internet of Things Journal*, vol. 1, no. 3, pp. 255–264, 2014.

[13] L. A. Barroso, J. Clidaras, and U. Hölzle, "The datacenter as a computer: An introduction to the design of warehouse-scale machines," *Synthesis lectures on computer architecture*, vol. 8, no. 3, pp. 1–154, 2013.

[14] J. Garside, "Amazon's record $21bn Christmas sales push shares to new high," jan 2013. [Online]. Available: http://tinyurl.com/qjo7g6v

[15] Theguardian, "China's Alibaba records 'singles day' sales of $8bn in 10 hours," 2015. [Online]. Available: http://tinyurl.com/ch-AliBhaba

[16] P. Jamshidi, A. Ahmad, and C. Pahl, "Autonomic resource provisioning for cloud-based software," in *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. ACM, 2014, pp. 95–104.

[17] G. Urdaneta, G. Pierre, and M. van Steen, "Wikipedia workload analysis for decentralized hosting," *Computer Networks*, vol. 53, pp. 1830–1845, 2009.

[18] M. Kihl, E. Elmroth, J. Tordsson, K.-E. Årzén, and A. Robertsson, "The challenge of cloud control," in *8th International Workshop on Feedback Computing*, 2013.

[19] J. Jiang, J. Lu, G. Zhang, and G. Long, "Optimal cloud resource auto-scaling for web applications," in *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*. IEEE, 2013, pp. 58–65.

[20] P. Costa, T. Zahn, A. Rowstron, G. O'Shea, and S. Schubert, "Why should we integrate services, servers, and networking in a data center?" in *Proceedings of the 1st ACM workshop on Research on enterprise networking*. ACM, 2009, pp. 111–118.

[21] J. Liu, Y. Zhang, Y. Zhou, D. Zhang, and H. Liu, "Aggressive resource provisioning for ensuring QoS in virtualized environments," *IEEE Transactions on Cloud Computing*, vol. 02, no. 03, pp. 119–131, 2014.

[22] I. T. Archive, "Worldcup 1998 Web trace," 2015. [Online]. Available: http://ita.ee.lbl.gov/html/contrib/WorldCup.html

[23] V. Almeida, M. Arlitt, and J. Rolia, "Analyzing a web-based system's performance measures at multiple time scales," *SIGMETRICS Perform. Eval. Rev.*, vol. 30, no. 2, pp. 3–9, 2002.

[24] M. Arlitt and T. Jin, "A workload characterization study of the 1998 world cup web site," *IEEE network*, vol. 14, no. 3, pp. 30–37, 2000.

[25] N. R. Herbst, S. Kounev, and R. Reussner, "Elasticity in cloud computing: what it is , and what it is not," in *10th International Conference on Autonomic Computing*, 2013, pp. 23–27.

[26] Amazon, "Amazon auto scaling," 2015. [Online]. Available: https://aws.amazon.com/

[27] Rightscale, "Set up autoscaling using alert escalations," 2015. [Online]. Available: http://tinyurl.com/rightScale-autoscaling

[28] R. Han, L. Guo, M. M. Ghanem, and Y. Guo, "Lightweight resource scaling for cloud applications," in *Cluster, Cloud and Grid Computing (CCGrid), 2012 12th IEEE/ACM International Symposium on*. IEEE, 2012, pp. 644–651.

[29] P. Koperek and W. Funika, "Dynamic business metrics-driven resource provisioning in cloud environments," *Parallel Processing and Applied Mathematics*, pp. 171–180, 2012.

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

[31] H. C. Lim, S. Babu, J. S. Chase, and S. S. Parekh, "Automated control in cloud computing: challenges and opportunities," in *Proceedings of the 1st workshop on Automated control for datacenters and clouds.* ACM, 2009, pp. 13–18.

[32] A. Arman, A. Al-Shishtawy, and V. Vlassov, "Elasticity controller for Cloud-based key-value stores," *Proceedings of the International Conference on Parallel and Distributed Systems - ICPADS*, pp. 268–275, 2012.

[33] I. Gergin, B. Simmons, and M. Litoiu, "A decentralized autonomic architecture for performance control in the cloud," in *Proceedings - 2014 IEEE International Conference on Cloud Engineering, IC2E 2014*, 2014, pp. 574–579.

[34] A. Ashraf, B. Byholm, and I. Porres, "CRAMP: Cost-efficient Resource Allocation for Multiple web applications with Proactive scaling," in *CloudCom 2012 - Proceedings: 2012 4th IEEE International Conference on Cloud Computing Technology and Science*, 2012, pp. 581–586.

[35] P. Jamshidi, A. Sharifloo, C. Pahl, H. Arabnejad, A. Metzger, and G. Estrada, "Fuzzy self-learning controllers for elasticity management in dynamic cloud architectures," in *Proceedings - 2016 12th International ACM SIGSOFT Conference on Quality of Software Architectures, QoSA 2016*, 2016, pp. 70–79.

[36] Q. Zhang, M. F. Zhani, S. Zhang, Q. Zhu, R. Boutaba, and J. L. Hellerstein, "Dynamic Energy-Aware Capacity Provisioning for Cloud Computing En-

vironments," *Proceedings of the 9th international conference on Autonomic computing*, pp. 145–154, 2012.

[37] Q. Zhang, Zhani, R. Boutaba, and J. L. Hellerstein, "Dynamic heterogeneity-aware resource provisioning in the cloud," *IEEE Transactions on Cloud Computing*, vol. 2, no. 1, pp. 14–28, 2015.

[38] H. Ghanbari, B. Simmons, M. Litoiu, C. Barna, and G. Iszlai, "Optimal autoscaling in a IaaS cloud," *Proceedings of the 9th international conference on Autonomic computing - ICAC '12*, vol. 2, no. i, p. 173, 2012.

[39] C.-Z. Xu, J. Rao, and X. Bu, "URL: A unified reinforcement learning approach for autonomic cloud management," *Journal of Parallel and Distributed Computing*, vol. 72, no. 2, pp. 95–105, 2012.

[40] A. Ali-Eldin, J. Tordsson, and E. Elmroth, "An adaptive hybrid elasticity controller for cloud infrastructures," in *Network Operations and Management Symposium (NOMS), 2012 IEEE*. IEEE, 2012, pp. 204–212.

[41] R. Ranjan, L. Wang, A. Y. Zomaya, D. Georgakopoulos, X.-H. Sun, and G. Wang, "Recent advances in autonomic provisioning of big data applications on clouds," *Cloud Computing, IEEE Transactions on*, vol. 3, no. 2, pp. 101–104, 2015.

[42] S. Singh and I. Chana, "QoS-aware autonomic resource management in cloud computing: a systematic review," *ACM Computing Surveys (CSUR)*, vol. 48, no. 3, p. 42, 2015.

[43] D. Gmach, J. Rolia, L. Cherkasova, and A. Kemper, "Resource pool management: Reactive versus proactive or let's be friends," *Computer Networks*, vol. 53, no. 17, pp. 2905–2922, 2009.

[44] A. Gambi, G. Toffetti, and M. Pezzè, "Assurance of self-adaptive controllers for the cloud," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7740 LNCS, 2013, pp. 311–339.

[45] 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.

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

[47] S. Farokhi, P. Jamshidi, I. Brandic, and E. Elmroth, "Self-adaptation Challenges for Cloud-based Applications: A Control Theoretic Perspective," in *10th International Workshop on Feedback Computing*, Seattle, USA, 2015.

[48] K. S. Narendra, O. A. Driollet, M. Feiler, and K. George, "Adaptive control using multiple models, switching and tuning," *International Journal of Adaptive Control and Signal Processing*, vol. 17, no. 2, pp. 87–102, 2003.

[49] S. Farokhi, P. Jamshidi, E. B. Lakew, I. Brandic, and E. Elmroth, "A hybrid cloud controller for vertical memory elasticity: A control-theoretic approach," *Future Generation Computer Systems*, vol. 65, pp. 57–72, 2016.

[50] Q. Lu, X. Xu, L. Zhu, L. Bass, Z. Li, S. Sakr, P. L. Bannerman, and A. Liu, "Incorporating uncertainty into in-cloud application deployment decisions for availability," in *Cloud Computing (CLOUD), 2013 IEEE Sixth International Conference on*. IEEE, 2013, pp. 454–461.

[51] L. A. Zadeh, "Fuzzy logic= computing with words," *Fuzzy Systems, IEEE Transactions on*, vol. 4, no. 2, pp. 103–111, 1996.

[52] G. Qin, Z. Duan, G. Wen, Y. Yan, and Z. Jiang, "An Improved Anti-Windup Bumpless Transfer Structures Design for Controllers Switching," *Asian Journal of Control*, vol. 16, no. 4, pp. 1245–1251, 2014.

[53] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury, *Feedback control of computing systems*. John Wiley & Sons, 2004.

[54] R. Abdullah, A. Hussain, K. Warwick, and A. Zayed, "Autonomous intelligent cruise control using a novel multiple-controller framework incorporating fuzzy-logic-based switching and tuning," *Neurocomputing*, vol. 71, no. 13, pp. 2727–2741, 2008.

[55] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, 2011.

[56] A. Hussain, R. Abdullah, E. Yang, and K. Gurney, "An intelligent multiple-controller framework for the integrated control of autonomous vehicles," in *Advances in Brain Inspired Cognitive Systems*. Springer, 2012, pp. 92–101.

[57] K. Gurney, T. J. Prescott, and P. Redgrave, "A computational model of action selection in the basal ganglia. I. A new functional anatomy." *Biological cybernetics*, vol. 84, no. 6, pp. 401–410, 2001.

[58] ——, "A computational model of action selection in the basal ganglia. II. Analysis and simulation of behaviour." *Biological cybernetics*, vol. 84, no. 6, pp. 411–423, 2001.

[59] I. Foster, Y. Zhao, I. Raicu, and S. Lu, "Cloud computing and grid computing 360-degree compared," in *Grid Computing Environments Workshop, 2008. GCE'08*. Ieee, 2008, pp. 1–10.

[60] P. H. Beckman, "Building the TeraGrid," *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, vol. 363, no. 1833, pp. 1715–1728, 2005.

[61] R. Cossu, M. Petitdidier, J. Linford, V. Badoux, L. Fusco, B. Gotab, L. Hluchy, G. Lecca, F. Murgia, C. Plevier, and Others, "A roadmap for a dedicated Earth Science Grid platform," *Earth Science Informatics*, vol. 3, no. 3, pp. 135–148, 2010.

[62] N. S. Foundation, "Open Science Grid," 2013. [Online]. Available: https://www.opensciencegrid.org/

[63] B. Jeff, "Amazon EC2 beta," 2006. [Online]. Available: https://aws.amazon.com/blogs/aws/amazon{_}ec2{_}beta/

[64] L. Qian, Z. Luo, Y. Du, and L. Guo, "Cloud computing: An overview," *Cloud computing*, pp. 626–631, 2009.

[65] S.-C. Media and Inc, "Twenty-One Experts Define Cloud Computing." [Online]. Available: http://tinyurl.com/21-cloudDef

[66] P. Mell and T. Grance, "The NIST definition of cloud computing," *NIST Special Publication*, vol. 145, p. 7, 2011. [Online]. Available: http://www.mendeley.com/research/the-nist-definition-about-cloud-computing/

[67] Amazon, "Amazon Cloud," 2016. [Online]. Available: https://aws.amazon.com/ec2/

[68] J.-M. Kang, H. Bannazadeh, and A. Leon-Garcia, "SAVI testbed: Control and management of converged virtual ICT resources," in *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*. IEEE, 2013, pp. 664–667.

[69] S. Wardley, E. Goyer, and N. Barcet, "Ubuntu enterprise cloud architecture," *Technical white paper*, 2009.

[70] B. Zwattendorfer, K. Stranacher, A. Tauber, and P. Reichstädter, "Cloud computing in e-government across europe," in *International Conference on Electronic Government and the Information Systems Perspective*. Springer, 2013, pp. 181–195.

[71] Equinix, "Efficient, Flexible, and Cost-Effective Storage Solutions," 2017. [Online]. Available: http://www.equinix.com/partners/net-app/

[72] D. Owens, "Securing elasticity in the cloud," *Communications of the ACM*, vol. 53, no. 6, pp. 46–51, 2010.

[73] J. O. Kephart and W. E. Walsh, "An artificial intelligence perspective on autonomic computing policies," in *Policies for Distributed Systems and Networks, 2004. POLICY 2004. Proceedings. Fifth IEEE International Workshop on*. IEEE, 2004, pp. 3–12.

[74] E. Ferreira, C. ·. Flávio, R. De, C. Sousa, P. Antonio, L. Rego, ·. Danielo, G. Gomes, J. Neuman De Souza, E. F. Coutinho, F. R. De, P. A. L. Rego, D. G. Gomes, and J. N. De Souza, "Elasticity in cloud computing: a survey," *Ann. Telecommun*, vol. 70, pp. 289–309, 2015.

[75] A. Naskos, A. Gounaris, and S. Sioutas, "Cloud Elasticity: A Survey," in *Algorithmic Aspects of Cloud Computing*. Springer, 2016, pp. 151–167.

[76] F. Fargo, C. Tunc, Y. Al-Nashif, A. Akoglu, and S. Hariri, "Autonomic workload and resources management of cloud computing services," in *Cloud and Autonomic Computing (ICCAC), 2014 International Conference on*. IEEE, 2014, pp. 101–110.

[77] A. Gandhi, Y. Chen, D. Gmach, M. Arlitt, and M. Marwah, "Minimizing data center sla violations and power consumption via hybrid resource provisioning," in *Green Computing Conference and Workshops (IGCC), 2011 International*. IEEE, 2011, pp. 1–8.

[78] A. Najjar, X. Serpaggi, C. Gravier, and O. Boissier, "Survey of elasticity management solutions in cloud computing," in *Continued Rise of the Cloud*. Springer, 2014, pp. 235–263.

[79] C. Qu, R. N. Calheiros, and R. Buyya, "Auto-scaling Web Applications in Clouds: A Taxonomy and Survey," *arXiv preprint arXiv:1609.09224*, 2016.

[80] P. Patel, A. H. Ranabahu, and A. P. Sheth, "Service level agreement in cloud computing," 2009.

[81] M. Alhamad, T. Dillon, and E. Chang, "Conceptual SLA framework for cloud computing," in *Digital Ecosystems and Technologies (DEST), 2010 4th IEEE International Conference on*. IEEE, 2010, pp. 606–610.

[82] M. C. Calzarossa, M. L. Della Vedova, L. Massari, D. Petcu, M. I. M. Tabash, and D. Tessera, "Workloads in the Clouds," in *Principles of Performance and Reliability Modeling and Evaluation*. Springer, 2016, pp. 525–550.

[83] B. Schroeder, A. Wierman, and M. Harchol-Balter, "Open versus closed: a cautionary tale," *Proceedings of the 3rd conference on Networked Systems Design & Implementation (NSDI'06)*, vol. 1, 2006.

[84] A. Ali-Eldin, J. Tordsson, E. Elmroth, and M. Kihl, "Workload classification for efficient auto-scaling of cloud resources," *Department of Computer Science, Umea University, Umea, Sweden, Tech.Rep*, 2013.

[85] T. F. Abdelzaher, K. G. Shin, and N. Bhatti, "Performance guarantees for web server end-systems: A control-theoretical approach," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 13, no. 1, pp. 80–96, 2002.

[86] N. Gandhi, D. M. Tilbury, Y. Diao, J. Hellerstein, and S. Parekh, "Mimo control of an apache web server: Modeling and controller design," in *American Control Conference, 2002. Proceedings of the 2002*, vol. 6. IEEE, 2002, pp. 4922–4927.

[87] S. Parekh, K. Rose, Y. Diao, V. Chang, J. Hellerstein, S. Lightstone, and M. Huras, "Throttling utilities in the IBM DB2 universal database server," in *American Control Conference, 2004. Proceedings of the 2004*, vol. 3. IEEE, 2004, pp. 1986–1991.

[88] M. Karlsson, C. Karamanolis, and X. Zhu, "Triage: Performance differentiation for storage systems using adaptive control," *ACM Transactions on Storage (TOS)*, vol. 1, no. 4, pp. 457–480, 2005.

[89] L. A. Zadeh, "Fuzzy sets," *Information and control*, vol. 8, no. 3, pp. 338–353, 1965.

[90] B. Jennings and R. Stadler, "Resource management in clouds: Survey and research challenges," *Journal of Network and Systems Management*, vol. 23, no. 3, pp. 567–619, 2015.

[91] S. Mustafa, B. Nazir, A. Hayat, S. A. Madani, and Others, "Resource management in cloud computing: Taxonomy, prospects, and challenges," *Computers & Electrical Engineering*, vol. 47, pp. 186–203, 2015.

[92] S. S. Manvi and G. Krishna Shyam, "Resource management for Infrastructure as a Service (IaaS) in cloud computing: A survey," pp. 424–440, 2014.

[93] S. Singh and I. Chana, "Resource provisioning and scheduling in clouds: QoS perspective," pp. 1–35, 2016.

[94] C. a. Yfoulis and a. Gounaris, "Honoring SLAs on cloud computing services: A control perspective," *Control Conference (ECC), 2009 European*, pp. 184–189, 2009.

[95] T. Patikirikorala and A. Colman, "Feedback controllers in the cloud," in *Proceedings of APSEC*, 2010.

[96] T. Patikirikorala, A. Colman, J. Han, and L. Wang, "A systematic survey on the design of self-adaptive software systems using control engineering approaches," in *Proceedings of the 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. IEEE Press, 2012, pp. 33–42.

[97] S. Costache, D. Dib, N. Parlavantzas, and C. Morin, "Resource Management in Cloud Platform as a Service Systems: Analysis and Opportunities," *Journal of Systems and Software*, 2017.

[98] F. D. Muñoz-Escoí and J. M. Bernabéu-Aubán, "A survey on elasticity management in paas systems," *Computing*, vol. 99, no. 7, pp. 617–656, 2017.

[99] G. Galante and L. C. E. De Bona, "A survey on cloud computing elasticity," in *Proceedings - 2012 IEEE/ACM 5th International Conference on Utility and Cloud Computing, UCC 2012*, 2012, pp. 263–270.

[100] Y. Shoaib and O. Das, "Performance-oriented Cloud Provisioning: Taxonomy and Survey," *arXiv:1411.5077*, no. November, pp. 1–14, 2014.

[101] Y. Al-Dhuraibi, F. Paraiso, N. Djarallah, and P. Merle, "Elasticity in Cloud Computing: State of the Art and Research Challenges," *IEEE Transactions on Services Computing*, 2017.

[102] A. R. Hummaida, N. W. Paton, and R. Sakellariou, "Adaptation in cloud resource configuration: a survey," *Journal of Cloud Computing*, vol. 5, no. 1, pp. 1–16, 2016.

[103] G. Galante and R. da Rosa Righi, "Exploring Cloud Elasticity in Scientific Applications," in *Cloud Computing*. Springer, 2017, pp. 101–125.

[104] D. Ardagna, C. Ghezzi, and R. Mirandola, "Rethinking the use of models in software architecture," *Quality of Software Architectures. Models and Architectures*, pp. 1–27, 2008.

[105] L. Ljung, "Black-box models from input-output measurements," in *Instrumentation and Measurement Technology Conference, 2001. IMTC 2001. Proceedings of the 18th IEEE*, vol. 1. IEEE, 2001, pp. 138–146.

[106] A. E. Ruano, *Intelligent control systems using computational intelligence techniques*. Iet, 2005, vol. 70.

[107] J. B. Rawlings and D. Q. Mayne, *Model predictive control: Theory and design*. Nob Hill Pub., 2009.

[108] A. Tchernykh, U. Schwiegelsohn, V. Alexandrov, and E.-g. Talbi, "Towards Understanding Uncertainty in Cloud Computing Resource Provisioning," *Procedia Computer Science*, vol. 51, pp. 1772–1781, 2015.

[109] D. Kusic, J. O. Kephart, J. E. Hanson, N. Kandasamy, and G. Jiang, "Power and performance management of virtualized computing environments via lookahead control," *Cluster Computing*, vol. 12, no. 1, pp. 1–15, 2009.

[110] C. Barna, M. Fokaefs, M. Litoiu, M. Shtern, and J. Wigglesworth, "Cloud Adaptation with Control Theory in Industrial Clouds," in *Cloud Engineering*

Workshop (IC2EW), 2016 IEEE International Conference on. IEEE, 2016, pp. 231–238.

[111] M. Berekmeri, D. Serrano, S. Bouchenak, N. Marchand, and B. Robu, "A Control Approach for Performance of Big Data Systems," *{IFAC} Proceedings Volumes*, vol. 47, no. 3, pp. 152–157, 2014.

[112] A. Ashraf, B. Byholm, J. Lehtinen, and I. Porres, "Feedback Control Algorithms to Deploy and Scale Multiple Web Applications per Virtual Machine," in *2012 38th EUROMICRO conference on Software Engineering and Advanced Applications (SEAA)*, 2012, pp. 431–438.

[113] J. Heo, X. Zhu, P. Padala, and Z. Wang, "Memory overbooking and dynamic control of xen virtual machines in consolidated environments," in *2009 IFIP/IEEE International Symposium on Integrated Network Management, IM 2009*, 2009, pp. 630–637.

[114] H. C. Lim, S. Babu, and J. S. Chase, "Automated control for elastic storage," in *Proceedings of the 7th international conference on Autonomic computing*. ACM, 2010, pp. 1–10.

[115] Vmware, "Hyperic HQ," 2016. [Online]. Available: http://www.vmware.com/products/vrealize-hyperic.html

[116] F. Cappello, E. Caron, M. Dayde, F. Desprez, Y. Jegou, P. Primet, E. Jeannot, S. Lanteri, J. Leduc, N. Melab, G. Mornet, R. Namyst, B. Quetier, and O. Richard, "Grid'5000: A large scale and highly reconfigurable Grid experimental testbed," in *Proceedings - IEEE/ACM International Workshop on Grid Computing*, vol. 2005, 2005, pp. 99–106.

[117] J. Chase, L. Grit, D. Irwin, V. Marupadi, P. Shivam, and A. Yumerefendi, "Beyond virtual data centers: Toward an open resource control architecture," in *Selected Papers from the International Conference on the Virtual Computing Initiative (ACM Digital Library)*, 2007.

[118] R. Sumbaly, J. Kreps, L. Gao, A. Feinberg, C. Soman, and S. Shah, "Serving large-scale batch computed data with project voldemort," in *Proceedings of the 10th USENIX conference on File and Storage Technologies*. USENIX Association, 2012, p. 18.

[119] Q. Li, Q. F. Hao, L. M. Xiao, and Z. J. Li, "An integrated approach to automatic management of virtualized resources in cloud environments," *Computer Journal*, vol. 54, no. 6, pp. 905–919, 2011.

[120] M. A. Moulavi, A. Al-Shishtawy, and V. Vlassov, "State-space feedback control for elastic distributed storage in a cloud environment," in *The 8th International Conference on Autonomic and Autonomous Systems (ICAS 2012)*, 2012, pp. 589–596.

[121] P. S. Saikrishna, R. Pasumarthy, and H. A. Kruthika, "Stability Analysis of Cloud Computing Systems under Uncertain Time delays," in *Proc. The 21st International Symposium on Mathematical Theory of Networks and Systems, Groningen, The Netherlands*, 2014.

[122] S. Farokhi, P. Jamshidi, D. Lucanin, and I. Brandic, "Performance-based vertical memory elasticity," in *Proceedings - IEEE International Conference on Autonomic Computing, ICAC 2015*, 2015, pp. 151–152.

[123] S. Spinner, S. Kounev, X. Zhu, L. Lu, M. Uysal, A. Holler, and R. Griffith, "Runtime vertical scaling of virtualized applications via online model estimation," in *2014 IEEE Eighth International Conference on Self-Adaptive and Self-Organizing Systems*. IEEE, 2014, pp. 157–166.

[124] A. Ali-Eldin, M. Kihl, J. Tordsson, and E. Elmroth, "Efficient provisioning of bursty scientific workloads on the cloud using adaptive elasticity control," in *Proceedings of the 3rd workshop on Scientific Cloud Computing Date*. ACM, 2012, pp. 31–40.

[125] P. Padala, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, and K. Salem, "Adaptive control of virtualized resources in utility comput-

ing environments," in *ACM SIGOPS Operating Systems Review*, vol. 41, no. 3. ACM, 2007, pp. 289–302.

[126] Zimbra, "Zimbra Collaboration Serve." [Online]. Available: https://www.zimbra.com/

[127] J. Banks, J. S. CARSON II, L. Barry, and Others, *Discrete-event system simulationfourth edition*. Pearson, 2005.

[128] W. Iqbal, M. N. Dailey, D. Carrera, and P. Janecek, "Adaptive resource provisioning for read intensive multi-tier applications in the cloud," *Future Generation Computer Systems*, vol. 27, no. 6, pp. 871–879, 2011.

[129] T. C. Chieu, A. Mohindra, A. A. Karve, and A. Segal, "Dynamic scaling of web applications in a virtualized cloud computing environment," in *E-Business Engineering, 2009. ICEBE'09. IEEE International Conference on*. IEEE, 2009, pp. 281–286.

[130] X. Zhu, Z. Wang, and S. Singhal, "Utility-driven workload management using nested control design," in *American Control Conference, 2006*. IEEE, 2006, pp. 6—-pp.

[131] P. Padala, K.-Y. Hou, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, and A. Merchant, "Automated Control of Multiple Virtualized Resources," in *EuroSys'09*, 2009, pp. 13–26.

[132] X. Liu, X. Zhu, S. Singhal, and M. Arlitt, "Adaptive entitlement control of resource containers on shared servers," in *2005 9th IFIP/IEEE International Symposium on Integrated Network Management, IM 2005*, vol. 2005, 2005, pp. 163–176.

[133] S.-M. Park and M. Humphrey, "Self-tuning virtual machines for predictable escience," in *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*. IEEE Computer Society, 2009, pp. 356–363.

[134] Q. Zhu and G. Agrawal, "Resource provisioning with budget constraints for adaptive applications in cloud environments," *IEEE Transactions on Services Computing*, vol. 5, no. 4, pp. 497–511, 2012.

[135] E. Kalyvianaki, T. Charalambous, and S. Hand, "Self-adaptive and self-configured cpu resource provisioning for virtualized servers using kalman filters," in *Proceedings of the 6th international conference on Autonomic computing*. ACM, 2009, pp. 117–126.

[136] A. J. Smola and B. Schölkopf, "A tutorial on support vector regression," *Statistics and computing*, vol. 14, no. 3, pp. 199–222, 2004.

[137] F. P. Kelly, "Models for a self–managed Internet," *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, vol. 358, no. 1773, pp. 2335–2348, 2000.

[138] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar, "An integrated experimental environment for distributed systems and networks," *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 255–270, 2002.

[139] S. M. Park and M. Humphrey, "Feedback-controlled resource sharing for predictable escience," in *2008 SC - International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2008*, 2008.

[140] G. Moltó, M. Caballer, E. Romero, and C. de Alfonso, "Elastic memory management of virtualized infrastructures for applications with dynamic memory requirements," *Procedia Computer Science*, vol. 18, pp. 159–168, 2013.

[141] S. Farokhi, E. B. Lakew, C. Klein, I. Brandic, and E. Elmroth, "Coordinating cpu and memory elasticity controllers to meet service response time constraints," in *Cloud and Autonomic Computing (ICCAC), 2015 International Conference on*. IEEE, 2015, pp. 69–80.

[142] S. Cerf, M. Berekmeri, B. Robu, N. Marchand, and S. Bouchenak, "Cost function based event triggered Model Predictive Controllers application to Big Data Cloud services," in *Decision and Control (CDC), 2016 IEEE 55th Conference on*. IEEE, 2016, pp. 1657–1662.

[143] P. Lama, Y. Guo, C. Jiang, and X. Zhou, "Autonomic Performance and Power Control for Co-Located Web Applications in Virtualized Datacenters," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 5, pp. 1289–1302, 2016.

[144] B. Trushkowsky, P. Bodik, A. Fox, M. J. Franklin, M. I. Jordan, and D. A. Patterson, "The SCADS Director: Scaling a Distributed Storage System Under Stringent Performance Requirements," *FAST '11 Proceedings of the 9th USENIX conference on File and Storage Technologies*, pp. 163–176, 2011.

[145] T. Patikirikorala, L. Wang, and A. Colman, "Towards optimal performance and resource management in web systems via model predictive control," in *Australian Control Conference (AUCC), 2011*, 2011, pp. 469–474.

[146] X. Wang and M. Chen, "Cluster-level feedback power control for performance optimization," *2008 IEEE 14th International Symposium on High Performance Computer Architecture*, pp. 101–110, 2008.

[147] L. Wang, J. Xu, H. A. Duran-Limon, and M. Zhao, "QoS-driven Cloud Resource Management Through Fuzzy Model Predictive Control," *IEEE International Conference on Autonomic Computing (ICAC), 2015*, pp. 81–90, 2015.

[148] D. Kusic and N. Kandasamy, "Risk-aware limited lookahead control for dynamic resource provisioning in enterprise computing systems," *Cluster Computing*, vol. 10, no. 4, pp. 395–408, 2007.

[149] J. Bai and S. Abdelwahed, "Efficient algorithms for performance management of computing systems," in *Fourth International Workshop on Feedback*

Control Implementation and Design in Computing Systems and Networks FeBID. IEEE, 2009.

[150] G. E. P. Box, G. M. Jenkins, and G. C. Reinsel, *Time Series Analysis: Forecasting & Control*, 1994.

[151] Z. Ren, X. Xu, J. Wan, W. Shi, and M. Zhou, "Workload characterization on a production hadoop cluster: A case study on taobao," in *Workload Characterization (IISWC), 2012 IEEE International Symposium on*. IEEE, 2012, pp. 3–13.

[152] Google, "Google cluster workload traces." [Online]. Available: http://code.google.com/p/googleclusterdata/

[153] M. F. Arlitt and C. L. Williamson, "Web server workload characterization: The search for invariants," *ACM SIGMETRICS Performance Evaluation Review*, vol. 24, no. 1, pp. 126–137, 1996.

[154] P. Lama and X. Zhou, "Coordinated power and performance guarantee with fuzzy MIMO control in virtualized server clusters," *IEEE Transactions on Computers*, vol. 64, no. 1, pp. 97–111, 2015.

[155] Z. Gong and X. Gu, "Pac: Pattern-driven application consolidation for efficient cloud computing," in *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2010 IEEE International Symposium on*. IEEE, 2010, pp. 24–33.

[156] W. Dawoud, I. Takouna, and C. Meinel, "Elastic VM for cloud resources provisioning optimization," in *Communications in Computer and Information Science*, vol. 190 CCIS, no. PART 1, 2011, pp. 431–445.

[157] P. Xiong, Z. Wang, S. Malkowski, Q. Wang, D. Jayasinghe, and C. Pu, "Economical and robust provisioning of n-tier cloud workloads: A multi-level control approach," in *Distributed Computing Systems (ICDCS), 2011 31st International Conference on*. IEEE, 2011, pp. 571–580.

[158] J. Rao, Y. Wei, J. Gong, and C.-Z. Xu, “QoS guarantees and service differentiation for dynamic cloud applications,” *IEEE Transactions on Network and Service Management*, vol. 10, no. 1, pp. 43–55, 2013.

[159] A. Al-Shishtawy and V. Vlassov, “ElastMan: elasticity manager for elastic Key-Value stores in the cloud,” *Cloud and Autonomic Computing Conference (CAC '13)*, p. 1, 2013.

[160] Z. Wang, X. Liu, A. Zhang, C. Stewart, X. Zhu, T. Kelly, S. Singhal, and Others, “AutoParam: Automated Control of Application-Level Performance in Virtualized Server Environments,” in *In Proceedings of the 2 nd IEEE International Workshop on Feedback Control Implementation in Computing Systems and Networks (FeBid*. Citeseer, 2007.

[161] Z. Wang, Y. Chen, D. Gmach, S. Singhal, B. J. Watson, W. Rivera, X. Zhu, and C. D. Hyser, “AppRAISE: application-level performance management in virtualized server environments,” *IEEE Transactions on Network and Service Management*, vol. 6, no. 4, 2009.

[162] M. A. Kjær, M. Kihl, and A. Robertsson, “Resource allocation and disturbance rejection in web servers using slas and virtualized servers,” *IEEE Transactions on Network and Service Management*, vol. 6, no. 4, 2009.

[163] C. Stewart, T. Kelly, and A. Zhang, “Exploiting nonstationarity for performance prediction,” in *ACM SIGOPS Operating Systems Review*, vol. 41, no. 3. ACM, 2007, pp. 31–44.

[164] T. Kelly, “Detecting performance anomalies in global applications,” in *Proceedings of the 2nd conference on Real, Large Distributed Systems-Volume 2*. USENIX Association, 2005, pp. 42–47.

[165] D. Henriksson, Y. Lu, and T. Abdelzaher, “Improved prediction for web server delay control,” *Proceedings. 16th Euromicro Conference on Real-Time Systems, 2004. ECRTS 2004.*, pp. 61–68, 2004.

[166] X. Liu, J. Heo, L. Sha, and X. Zhu, "Adaptive control of multi-tiered Web applications using queueing predictor," *IEEE Symposium Record on Network Operations and Management Symposium*, pp. 107–114, 2006.

[167] D. Grimaldi, V. Persico, A. Pescapé, A. Salvi, and S. Santini, "A feedback-control approach for resource management in public clouds," in *Global Communications Conference (GLOBECOM), 2015 IEEE*. IEEE, 2015, pp. 1–7.

[168] P. S. Saikrishna, R. Pasumarthy, and N. P. Bhatt, "Identification and Multivariable Gain-Scheduling Control for Cloud Computing Systems," *IEEE Transactions on Control Systems Technology*, 2016.

[169] W. Qin and Q. Wang, "Modeling and control design for performance management of web servers via an LPV approach," *IEEE Transactions on Control Systems Technology*, vol. 15, no. 2, pp. 259–275, 2007.

[170] M. Tanelli, D. Ardagna, and M. Lovera, "Identification of LPV state space models for autonomic web service systems," *IEEE Transactions on Control Systems Technology*, vol. 19, no. 1, pp. 93–103, 2011.

[171] T. Patikirikorala, A. Colman, and J. Han, "4M-Switch: Multi-mode-multi-model supervisory control framework for performance differentiation in virtual machine environments," in *Network and Service Management (CNSM), 2014 10th International Conference on*. IEEE, 2014, pp. 145–153.

[172] P. S. Saikrishna and R. Pasumarthy, "Multi-objective switching controller for cloud computing systems," *Control Engineering Practice*, vol. 57, pp. 72–83, 2016.

[173] IRCache, "Web Caching project," 2001. [Online]. Available: http://tinyurl.com/ircache

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

[175] J. Xu, M. Zhao, J. Fortes, R. Carpenter, and M. Yousif, "Autonomic resource management in virtualized data centers using fuzzy logic-based approaches," *Cluster Computing*, vol. 11, no. 3, pp. 213–227, 2008.

[176] L. Wang, J. Xu, M. Zhao, Y. Tu, and J. A. B. Fortes, "Fuzzy modeling based resource management for virtualized database systems," in *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2011 IEEE 19th International Symposium on*. IEEE, 2011, pp. 32–42.

[177] P. Lama and X. Zhou, "Efficient server provisioning with end-to-end delay guarantee on multi-tier clusters," in *Quality of Service, 2009. IWQoS. 17th International Workshop on*. IEEE, 2009, pp. 1–9.

[178] C. Anglano, M. Canonico, and M. Guazzone, "FC2Q: exploiting fuzzy control in server consolidation for cloud applications with SLA constraints," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 17, pp. 4491–4514, 2015.

[179] ——, "FCMS: A fuzzy controller for CPU and memory consolidation under SLA constraints," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 5, 2017.

[180] P. Lama and X. Zhou, "Autonomic provisioning with self-adaptive neural fuzzy control for percentile-based delay guarantee," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 8, no. 2, p. 9, 2013.

[181] D. Grimaldi, A. Pescape, A. Salvi, V. Persico, and Others, "A Fuzzy Approach based on Heterogeneous Metrics for Scaling Out Public Clouds," *IEEE Transactions on Parallel and Distributed Systems*, 2017.

[182] S. Frey, L. Claudia, C. Reich, and N. Clarke, "Cloud QoS Scaling by Fuzzy Logic," in *IEEE International Conference on Cloud Engineering (IC2E)*, 2014.

[183] D. Minarolli and B. Freisleben, "Virtual machine resource allocation in cloud computing via multi-agent fuzzy control," in *Proceedings - 2013 IEEE*

3rd International Conference on Cloud and Green Computing, CGC 2013 and 2013 IEEE 3rd International Conference on Social Computing and Its Applications, SCA 2013, 2013, pp. 188–194.

[184] F. Chen, D. Lambert, and J. C. Pinheiro, "Incremental quantile estimation for massive tracking," in *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2000, pp. 516–522.

[185] D. T, "t-digest: an algorithm for computing extremely accurate quantiles," 2015. [Online]. Available: https://github.com/tdunning/t-digest

[186] A. V. Papadopoulos, "Design and performance guarantees in cloud computing: Challenges and opportunities," in *10th International Workshop on Feedback Computing*, 2015.

[187] A. V. Papadopoulos, A. Ali-Eldin, K.-E. Årzén, J. Tordsson, and E. Elmroth, "Peas: A performance evaluation framework for auto-scaling strategies in cloud applications," *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)*, vol. 1, no. 4, p. 15, 2016.

[188] T. Abdelzaher, Y. Diao, J. L. Hellerstein, C. Lu, and X. Zhu, "Introduction to control theory and its application to computing systems," in *Performance Modeling and Engineering*. Springer US, 2008, pp. 185–215.

[189] X. Dutreilh, S. Kirgizov, O. Melekhova, J. Malenfant, N. Rivierre, and I. Truck, "Using reinforcement learning for autonomic resource allocation in clouds: towards a fully automated workflow," in *ICAS 2011, The Seventh International Conference on Autonomic and Autonomous Systems*, 2011, pp. 67–74.

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

[191] L. Ljung, *System identification*. Wiley Online Library, 1999.

[192] S. Parekh, N. Gandhi, J. Hellerstein, D. Tilbury, T. Jayram, and J. Bigus, "Using control theory to achieve service level objectives in performance management," *Real-Time Systems*, vol. 23, no. 1-2, pp. 127–141, 2002.

[193] J. C. Helton, "Uncertainty and sensitivity analysis in the presence of stochastic and subjective uncertainty," *Journal of Statistical Computation and Simulation*, vol. 57, no. 1-4, pp. 3–76, 1997.

[194] S. Chaudhuri, "What next?: a half-dozen data management research goals for big data and the cloud," in *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGAI symposium on Principles of Database Systems*. ACM, 2012, pp. 1–4.

[195] K. M. Passino, S. Yurkovich, and M. Reinfrank, *Fuzzy control*. Menlo Park, CA: Addison-wesley, 1998, vol. 42.

[196] P. Cingolani and J. Alcala-Fdez, "jFuzzyLogic: a robust and flexible fuzzy-Logic inference system language implementation." in *FUZZ-IEEE*. Citeseer, 2012, pp. 1–8.

[197] A. Beloglazov and R. Buyya, "Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers," *Concurrency and Computation: Practice and Experience*, vol. 24, no. 13, pp. 1397–1420, 2012.

[198] ——, "Adaptive threshold-based approach for energy-efficient consolidation of virtual machines in cloud data centers." in *MGC@ Middleware*, 2010, p. 4.

[199] D. B. LD and P. V. Krishna, "Honey bee behavior inspired load balancing of tasks in cloud computing environments," *Applied Soft Computing*, vol. 13, no. 5, pp. 2292–2303, 2013.

[200] A. Gandhi, M. Harchol-Balter, R. Raghunathan, and M. A. Kozuch, "Autoscale: Dynamic, robust capacity management for multi-tier data centers," *ACM Transactions on Computer Systems (TOCS)*, vol. 30, no. 4, p. 14, 2012.

[201] M. Mao and M. Humphrey, “Auto-scaling to minimize cost and meet application deadlines in cloud workflows,” in *High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for*. IEEE, 2011, pp. 1–12.

[202] Wikibench, “Wikipedia access traces,” jan 2009. [Online]. Available: http://www.wikibench.eu/?page{_}id=60

[203] WAND, “WITS: Waikato Internet Traffic Storage,” 2017. [Online]. Available: https://wand.net.nz/wits/index.php

[204] Amazon, “Amazon EC2 pricing,” 2015. [Online]. Available: https://aws.amazon.com/ec2/pricing/

[205] B. Girard, N. Tabareau, Q.-C. Pham, A. Berthoz, and J.-J. Slotine, “Where neuroscience and dynamic system theory meet autonomous robotics: a contracting basal ganglia model for action selection,” *Neural Networks*, vol. 21, no. 4, pp. 628–641, 2008.

[206] E. Yang, A. Hussain, and K. Gurney, “A brain-inspired soft switching approach: towards a cognitive cruise control system,” in *WIT Transactions on Engineering Sciences*, 2014.

[207] P. Redgrave, T. J. Prescott, and K. Gurney, “The basal ganglia: A vertebrate solution to the selection problem?” *Neuroscience*, vol. 89, no. 4, pp. 1009–1023, 1999.

[208] T. J. Prescott, P. Redgrave, and K. Gurney, “Layered control architectures in robots and vertebrates,” *Adaptive Behavior*, vol. 7, no. 1, pp. 99–127, 1999.

[209] S. E. Lyshevski, *Control systems theory with engineering applications*. Springer Science & Business Media, 2012.

[210] E. Yang, A. Hussain, and K. Gurney, “A basal ganglia inspired soft switching approach to the motion control of a car-like autonomous vehicle,” in *Advances in Brain Inspired Cognitive Systems*, 2013, vol. 7888, pp. 245–254.

[211] J. Fix, N. Rougier, and F. Alexandre, "A dynamic neural field approach to the covert and overt deployment of spatial attention," *Cognitive Computation*, vol. 3, no. 1, pp. 279–293, 2011.

[212] K. N. Gurney, "Reverse engineering the vertebrate brain: methodological principles for a biologically grounded programme of cognitive modelling," *Cognitive Computation*, vol. 1, no. 1, pp. 29–41, 2009.

[213] A. Mandali, M. Rengaswamy, V. S. Chakravarthy, and A. A. Moustafa, "A spiking basal ganglia model of synchrony, exploration and decision making," *Frontiers in Neuroscience*, vol. 9, p. 191, 2015.

[214] E. Yang, A. Hussain, and K. Gurney, "Neurobiologically-inspired soft switching control of autonomous vehicles," in *Advances in Brain Inspired Cognitive Systems*. Springer, 2012, pp. 82–91.

[215] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, 1989, vol. Addison-We.

[216] M. Fazzolari, R. Alcala, Y. Nojima, H. Ishibuchi, and F. Herrera, "A review of the application of multiobjective evolutionary fuzzy systems: Current status and further directions," *Fuzzy Systems, IEEE Transactions on*, vol. 21, no. 1, pp. 45–65, 2013.

[217] R. Alcalá, M. J. Gacto, F. Herrera, and J. Alcalá-Fdez, "A multi-objective genetic algorithm for tuning and rule selection to obtain accurate and compact linguistic fuzzy rule-based systems," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 15, no. 05, pp. 539–557, 2007.

[218] M. J. Gacto, R. Alcalá, and F. Herrera, "Integration of an index to preserve the semantic interpretability in the multiobjective evolutionary rule selection and tuning of linguistic fuzzy systems," *IEEE Transactions on Fuzzy Systems*, vol. 18, no. 3, pp. 515–531, 2010.

[219] A. A. Márquez, F. A. Márquez, and A. Peregr'in, "Rule Base and Inference System Cooperative Learning of Mamdani Fuzzy Systems with Multiobjective Genetic Algorithms." in *IFSA/EUSFLAT Conf.*, 2009, pp. 1045–1050.

[220] A. A. Márquez, F. A. Márquez, and A. Peregrín, "A multi-objective evolutionary algorithm with an interpretability improvement mechanism for linguistic fuzzy systems with adaptive defuzzification," in *Fuzzy Systems (FUZZ), 2010 IEEE International Conference on.* IEEE, 2010, pp. 1–7.

[221] R. Alcalá, P. Ducange, F. Herrera, B. Lazzerini, and F. Marcelloni, "A multiobjective evolutionary approach to concurrently learn rule and data bases of linguistic fuzzy-rule-based systems," *IEEE Transactions on Fuzzy Systems*, vol. 17, no. 5, pp. 1106–1122, 2009.

[222] R. RanjitK, "Design of experiment using the Taguchi approach," 2001.

[223] C.-C. Lee, "Fuzzy logic in control systems: fuzzy logic controller. II," *IEEE Transactions on systems, man, and cybernetics*, vol. 20, no. 2, pp. 419–435, 1990.

[224] T. J. U. o. N. M. Ross, *Fuzzy logic with engineering applications*, 2010.

[225] J. Li and K. Raymond, "A fuzzy genetic algorithm for driver scheduling," *European Journal Of Operational Research*, vol. 147, no. 2, pp. 334–344, 2003.

[226] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[227] C. a. C. Coello, G. B. Lamont, and D. a. V. Veldhuizen, *Evolutionary Algorithms for Solving Multi-Objective Problems Second Edition*, 2007.

[228] A. E. Eiben, E. Marchiori, and V. A. Valkó, "Evolutionary Algorithms with On-the-Fly Population Size Adjustment," *Parallel Problem Solving from Nature, (PPSN VIII)*, vol. 3242, no. 8, pp. 41–50, 2004.

[229] M. Affenzeller, S. Wagner, and S. Winkler, “Self-adaptive population size adjustment for genetic algorithms,” *Computer Aided Systems Theory*, pp. 820–828, 2007.

[230] M. Srinivas and L. M. Patnaik, “Adaptive Probabilities of Crossover and Mutation in Genetic Algorithms,” *IEEE Transactions on Systems, Man and Cybernetics*, vol. 24, no. 4, pp. 656–667, 1994.

[231] G. Taguchi, “System of experimental design; engineering methods to optimize quality and minimize costs,” New york, Tech. Rep., 1987.

[232] Y.-H. Liu and Y.-F. Luo, “Search for an optimal rapid-charging pattern for Li-ion batteries using the Taguchi approach,” *Industrial Electronics, IEEE Transactions on*, vol. 57, no. 12, pp. 3963–3971, 2010.

[233] A. M. Omekanda, “Robust torque and torque-per-inertia optimization of a switched reluctance motor using the Taguchi methods,” *IEEE Transactions on Industry Applications*, vol. 42, no. 2, pp. 473–478, 2006.

[234] F. D. Zahlay, K. S. Rama Rao, and T. B. Ibrahim, “A new intelligent autoreclosing scheme using artificial neural network and taguchi’s methodology,” *IEEE Transactions on Industry Applications*, vol. 47, no. 1, pp. 306–313, 2011.

[235] S.-Y. Ho, L.-S. Shu, and S.-Y. Ho, “Optimizing fuzzy neural networks for tuning PID controllers using an orthogonal simulated annealing algorithm OSA,” *Fuzzy Systems, IEEE Transactions on*, vol. 14, no. 3, pp. 421–434, 2006.

[236] H. M. Hasanien and S. M. Muyeen, “A Taguchi approach for optimum design of proportional-integral controllers in cascaded control scheme,” *IEEE Transactions on Power Systems*, vol. 28, no. 2, pp. 1636–1644, 2013.

[237] J. T. Tsai, J. H. Chou, and T. K. Liu, “Tuning the structure and parameters of a neural network by using hybrid Taguchi-genetic algorithm,” *IEEE Transactions on Neural Networks*, vol. 17, no. 1, pp. 69–80, 2006.

[238] H. Wang, Q. Geng, and Z. Qiao, "Parameter tuning of particle swarm optimization by using Taguchi method and its application to motor design," in *Information Science and Technology (ICIST), 2014 4th IEEE International Conference on*. IEEE, 2014, pp. 722–726.

[239] N. S. Jaddi, S. Abdullah, and A. R. Hamdan, "Taguchi-based parameter designing of genetic algorithm for artificial neural network training," in *Informatics and Creative Multimedia (ICICM), 2013 International Conference on*. IEEE, 2013, pp. 278–281.

[240] J.-T. Tsai, J.-C. Fang, and J.-H. Chou, "Optimized task scheduling and resource allocation on cloud computing environment using improved differential evolution algorithm," *Computers & Operations Research*, vol. 40, no. 12, pp. 3045–3055, 2013.

[241] H. Vasudevan, N. C. Deshpande, and R. R. Rajguru, "Grey Fuzzy Multiobjective Optimization of Process Parameters for CNC Turning of GFRP/Epoxy Composites," *Procedia Engineering*, vol. 97, pp. 85–94, 2014.

[242] J. Li and R. S. K. Kwan, "A meta-heuristic with orthogonal experiment for the set covering problem," *Journal of Mathematical Modelling and Algorithms*, vol. 3, no. 3, pp. 263–283, 2004.

[243] A. D. Keedwell and J. Dénes, *Latin squares and their applications*. Elsevier, 2015.

[244] W.-C. Weng, F. Yang, and A. Elsherbeni, "Electromagnetics and antenna optimization using Taguchi's method," *Synthesis Lectures on Computational Electromagnetics*, vol. 2, no. 1, pp. 1–94, 2007.

[245] S. P. Madahav, "Quality engineering using robust design," New Jersey, 1989.

[246] N. Sloane, "A Library of Orthogonal Arrays," 2012. [Online]. Available: http://neilsloane.com/oadir/

[247] J. J. Durillo and A. J. Nebro, "JMetal: A Java framework for multi-objective optimization," *Advances in Engineering Software*, vol. 42, no. 10, pp. 760–771, 2011.

[248] J. Li, "Fuzzy Evolutionary Approaches for Bus and Rail Driver Scheduling," Ph.D. dissertation, University of Leeds, 2002.

[249] A. Gandhi, P. Dube, A. Karve, A. Kochut, and L. Zhang, "Adaptive, Model-driven Autoscaling for Cloud Applications." in *ICAC*, vol. 14, 2014, pp. 57–64.

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

[251] A. Gambi, M. Pezze, and G. Toffetti, "Kriging-based self-adaptive cloud controllers," *IEEE Transactions on Services Computing*, vol. 9, no. 3, pp. 368–381, 2016.

[252] A. Ullah, J. Li, A. Hussain, and E. Yang, "Towards a Biologically Inspired Soft Switching Approach for Cloud Resource Provisioning," *Cognitive Computation*, pp. 1–14, 2016.

[253] A. Ullah, J. Li, A. Hussain, and Y. Shen, "Genetic optimization of fuzzy membership functions for cloud resource provisioning," in *Computational Intelligence (SSCI), 2016 IEEE Symposium Series on*. IEEE, 2016, pp. 1–8.