A METHODOLOGY FOR ELICITING AND RANKING CONTROL POINTS FOR ADAPTIVE SYSTEMS

PARISA ZOGHI

A THESIS SUBMITTED TO
THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF ARTS

GRADUATE PROGRAM IN INFORMATION SYSTEMS AND TECHNOLOGY
YORK UNIVERSITY
TORONTO, ONTARIO

APRIL 2014

Abstract

Designing an adaptive system to meet its quality constraints in the face of environmental uncertainties, such as variable demands, can be a challenging task. In cloud environment, a designer has to also consider and evaluate different control points, i.e., those variables that affect the quality of the software system. This thesis presents a method for eliciting, evaluating and ranking control points for web applications deployed in cloud environments. The proposed method consists of several phases that take a high-level stakeholders' adaptation goal and transform it into lower level MAPE-K loop control points. The MAPE-K loop is then activated at runtime using an adaptation algorithm. We conducted several experiments to evaluate the different phases of the methodology and we report the results and the lesson learnt.

Dedication

To my beloved family, who have always stood by me.

Acknowledgements

This thesis would have never been possible without the encouragement and support of amazing people whom I would like to mention and thank.

First and foremost, I am extremely grateful to my supervisor, Professor Marin Litoiu, whose guidance, advice, support, and infinite patience are the great examples of a model supervisor. I am honoured to be his student and feel privileged to have him as my mentor for many great things that I learned from him and all the advices he gave me along the way. I would like to thank him for introducing me to the area of adaptive systems, for the unconditional support, and for always assisting me in every possible way during my thesis and my time at York University. I learned to be hopeful, to be patient, and to cope with difficulties without giving up! I hope this thesis lives up to his expectations.

I am grateful to have been taught by Prof. Campeanu and Prof. Cysneiros who contributed in my knowledge for Java Programming and Requirements Engineering during my undergraduate studies at York. It was Prof. Cysneiros' Requirements

Management class which made me interested in requirements engineering and taught me the critical role of NFRs in software systems. I would like to show them my gratitude for accepting to be in my thesis committee, evaluating my work, and making useful comments. I would also like to thank Prof. Kim for participating in my thesis defense as an external examiner and providing feedback.

The constant encouragement and enthusiasm of Dr. Mark Shtern and Dr. Bradley Simmons with whom I worked closely was invaluable to my research. Our meetings and discussions had a great impact on my research and provided me with useful suggestions. I admired their hard work and dedication.Thanks for supporting and advising me, especially for the times I was discouraged. I also like to express my gratitude to Mark for his patience and mentorship during my thesis and publishing a paper together. His passion inspired me to do my best in developing my research skills. I can never thank him enough!

I am also thankful to Nikolay Yakovets and all my colleagues at the Adaptive Systems Research Lab (ASRL) who participated in my empirical study and supported me: Cornel Barna, Ian Gergin, Hamoun Ghanbari, Mihai Iacob, Hongbin Lu, Przemyslaw Pawluk, Roni Sandel, Dr. Mike Smit, and Vasileios (Billy) Theodorou. I am indebted to Hamoun Ghanbari, Cornel Barna, and Hongbin Lu who were the subjects of my pilot study and pointed out insightful comments about my preliminary work.

Last but not least, I would like to thank my parents, Mahnaz and Mohammad, and my brother, Sina, for their unconditional love and support. Thanks for always being there for me, believing in me, and cheering me up when I encountered difficulties during my graduate years at York. I dedicate this thesis to you. I love you!

Table of Contents

Abstract **ii**

Dedication **iii**

Acknowledgements **iv**

Table of Contents **vii**

List of Figures **x**

List of Tables **xii**

1 INTRODUCTION **1**

1.1 Problem and Motivation 1

1.2 Research Objectives 4

1.3 Thesis Contributions 6

1.4 Thesis Organization 8

2 BACKGROUND **10**

2.1 Cloud Computing 10

2.2 Autonomic Computing 13

2.2.1 Feedback Loop 15

2.3 Summary 20

3 RELATED WORK **21**

3.1 Modeling Adaptive Software Systems 22

3.2 Goal-Oriented Approaches to Adaptive Systems 26

3.3 NFR Elicitation for Adaptive Systems 30

3.4 Summary 34

4 METHODOLOGY FOR ELICITING AND RANKING CONTROL POINTS **36**

4.1 Elicit Control Points 37

4.1.1 Control Point Model 38

4.1.2 Control Point Catalogue 41

4.2 Rank Control Points 44

4.3 Implement a MAPE-k loop Adaptation Strategy 48

4.4 Summary 50

5 CASE STUDY: ADAPTIVE SHOPPING CART FOR CLOUD 52

5.1 Control Points for Online Shop 52

5.1.1 Elicit Control Points 54

5.1.2 Rank Control Points 54

5.2 Experiment 1: Validate Ranking Method 56

5.2.1 Experimental Setup 58

5.2.2 Experiments 60

5.2.3 Aggregate Model-based Ranks 69

5.2.4 Discussion 70

5.3 Experiment 2: Evaluation of Adaptation Strategies 70

5.3.1 Random Ranking 71

5.3.2 Leverage Points Ranking 71

5.3.3 Experiments 73

5.3.4 Experiment Result 79

5.4 Summary 82

6 CONCLUSIONS AND FUTURE WORK 84

Bibliography 87

Appendix A (Empirical Questionnaire) 96

List of Figures

1.1 Overview of our approach to design an adaptive system. 7

2.1 Four Adaptation Processes in Self-Adaptive Software [39]. 16

2.2 Autonomic control loop [6]. 17

2.3 IBM's autonomic element [6]. 18

3.1 Modeling Dimensions for Adaptive Systems [1] 23

4.1 Control Point Model for Runtime Cloud Cost 42

4.2 Control Point Model for Application Response Time 43

5.1 Pairwise Comparisons for Low Response Time 56

5.2 Impact of Adding Web Servers on Performance 64

5.3 Impact of Changing Instance type on Performance 65

5.4 The Effect of Designer Ranking on Response Time 76

5.5 The Effect of Random Ranking on Response Time 77

5.6 The Effect of Leverage Points Strategy on Response Time 78

5.7 Average number of iterations required to reach the adaptation goal . 80

5.8 Histogram of iterations required to reach the adaptation goal 81

5.9 The effect of ranking on the speed of achieving the adaptation goal . 82

List of Tables

4.1 NFRs and their corresponding Metrics 44

4.2 Elicited Control Points for performance goal (application) 44

4.3 Elicited Control Points for adaptation goals (cloud) 45

5.1 Resource Boundaries for Online Shop 53

5.2 Elicited Control Points and their definition 54

5.3 Direct Rank for Participant A 57

5.4 Designer Ranking- Final Rank Result 57

5.5 Minimum and Maximum Range for Application Parameters 59

5.6 Steps used for changing control points. 61

5.7 Initial Settings for Model M's Parameters 62

5.8 Performance Metrics for Model M 63

5.9 Experimental Ranking- Final Rank Result 69

5.10 Random Ranking 72

5.11 Leverage Points Ranking 73

5.12 Distribution of Response Time Deviation 74

5.13 Performance Metrics for M1 75

5.14 Number of Iterations for strategies 79

1 INTRODUCTION

1.1 Problem and Motivation

Web applications deployed on the cloud allow companies to lower their costs and scale them dynamically with 'on-demand' provisioning of cloud resources. However, despite numerous innovations, the industry still struggles to satisfy key Non-Functional Requirements (NFRs) such as scalability[1], performance, availability, and cost. For example, in 2011, Target.com[2] crashed twice as it was flooded with many online shoppers for a new high-end clothing line. In 2013, Amazon.com[3] went down for unknown reasons for at least 20 minutes. Evidently, achieving NFRs in the dynamic cloud environment in a cost-effective manner remains a largely unresolved problem due to the changing nature of the operational environment, complex requirements,

[1]In cloud, performance goals are achieved through scalability; hence, we look at the scalability as a subset of performance goals.

[2]`http://www.computerworld.com/s/article/9221221/Target.com`

[3]`http://venturebeat.com/2013/08/19/amazon-website-down/`

unexpected failures, etc.

Adaptive design such as autonomic computing [24] [26] [18] can help achieve the NFRs. Consequently, *self-adaptation*- the ability of software systems to modify their behaviour in accordance with changes in their operational environment and in the system itself- has gained a lot of attentions and has become an important concept in research [12]. Survey papers such as [10] and [39] discuss the challenges of designing and engineering such systems.

Brun et al. [6] emphasize the importance of feedback loop in designing adaptive systems. The authors consider feedback loops as first-class entities since they are the essential feature in controlling and managing uncertainties in software systems. They assert that without visible feedback loops, the impact of these feedback loops on overall system behaviour could not be identified and hence the most important properties of self-adaptation would be failed to be addressed.

Designing an adaptive system entails decisions such as how to monitor the system's environment, how to select and activate adaptations, etc. Currently, this is done in an ad-hoc manner [5]. Recently, Brun et al. introduced the concept of design space for adaptive systems, which contains key questions when attempting to design a self-adaptive system [5]. The authors present a conceptual model of how to identify different components of an adaptive systems by answering a set of questions along five dimensions: identification, observation, representation, control and adaptation mechanisms. For each dimension, they identify key design decisions to guide the design.

We conducted an exploratory study with a group of researchers to design an adaptive software system for an online shopping cart. The researchers were members of our Adaptive Systems Research Lab (ASRL)[4]; graduate students from IT and Computer Science Programs, as well as postdoctoral fellows. Some of the participants work in industry. All researchers have practical experience with the cloud. Moreover, they all have applied adaptations in their research work such as add/remove servers. Some have applied change instance type adaptations, and few have applied other type of adaptations such as adding threads, reducing network latency, and changing disk type.

The goal of our study was to assess whether they can design an adaptive system using accumulative knowledge from their research. We identified NFR goals for a multi-tier online shopping cart application running in the cloud as a low response time and low cloud resource cost. The participants were required to perform three tasks: 1) Identify 10 adaptation operations, which we call control points [45] throughout this thesis. We define control points as those artifacts in a system that modified

[4]`http://www.ceraslabs.com`

at run-time cause controlled changes in the system; thus, affect the quality of the services offered by the system, 2) Elicit feedback loops for them, 3) Rank feedback loops in order of importance to determine their impact on response time and cost.

The outcome of the study was a set of complex feedback loops, which were difficult to interpret. We observed that the participants did not have good intuitions to identify the control points and eliciting feedback loops around them. They appeared to rank the feedback loops randomly due to an apparent inability to prioritize which one to implement with regard to the adaptation goals.

This study motivated us to develop a methodology for designing adaptive web applications deployed on the cloud in order to meet their quality requirements.

1.2 Research Objectives

The objective of our research is to define a systematic approach for eliciting, selecting and ranking control points for cloud-based adaptive software systems. We decompose our objective into the following research questions:

- **RQ1:** How can we achieve adaptation goals defined for an adaptive system?

  We define adaptation goals as NFR goals that are achieved through adapting the software and infrastructure. In other words, a system is designed and adapted with respect to its predefined adaptation goals. In order for a system to achieve its adaptation goals, we construct an approach to elicit and rank a set of control points as shown in Figure 1.1. Using rank order of the control points, we then build an adaptation (runtime) strategy that will be implemented by a MAPE-K loop. Our approach is thoroughly described in Chapter 4.

- **RQ2:** How can we develop a systematic process to elicit control points for an adaptive system deployed on the cloud?

  We exploit the existing techniques in the literature and develop a method for elicitation of control points. Using a hierarchical approach, we propose a control point model, which facilitates the identification and elicitation of control points. A control point model has a structure of a tree, which takes the adaptation goal as the root of the tree, and decomposes it into lower level goals until it reaches the adaptation operations (i.e., control points). We then create a catalogue of the elicited control points that can be reused. The first element in Figure 1.1, Elicit Control Points, corresponds to our elicitation process. We introduce our elicitation method in Chapter 4.

- **RQ3:** How realistically can we capture and model the adaptation efficiency at design time?

  To answer this question, we acquire ranks for control points from two sources:

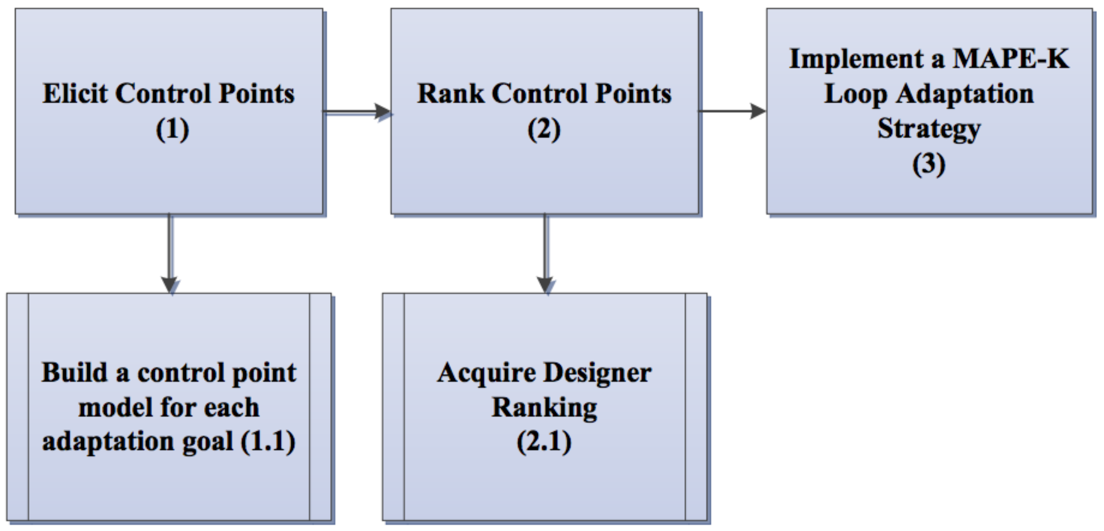
1) Model-based simulation, and 2) Group of designers. We expect that the ranks obtained from the designers (Designer Ranking) are as reliable as the ranks acquired by running a series of experiments with the simulation tool (Experimental Ranking). In this thesis, we explore whether the Designer Ranking can be utilized and relied on for designing an adaptive system. We show our experiments and findings in Chapter 5.

1.3 Thesis Contributions

This thesis focuses on designing cloud-based adaptive web applications by creating the mechanisms for achieving the adaptation goal. Our contribution consists of elicitation and ranking of control points. The main advantage of our approach is the ranking of control points in the early design phase, which enables prioritization of control points and an informed decision-making about which control points are essential to meet the stakeholders' objectives. Our approach, which answers research questions RQ1-RQ3 above, is summarized in Figure 1.1.

As shown in Figure 1.1, our process encompasses three steps:

1. **Elicit Control Points.** We formulate a set of alternatives through elicitation of control points. To facilitate the elicitation of control points, we propose



Elicit Control Points (1)
Rank Control Points (2)
Implement a MAPE-K Loop Adaptation Strategy (3)
Build a control point model for each adaptation goal (1.1)
Acquire Designer Ranking (2.1)

Figure 1.1: Overview of our approach to design an adaptive system.

control point models, which map the control points to NFRs. The control point models act as aids in elicitation process. Our hierarchical method constantly decomposes an adaptation goal until it reaches the operations, which are control points in the system that affect the adaptation goal. Our elicitation process is defined in Chapter 4.

2. **Rank Control Points.** Another contribution of this thesis is to find ranks for the previously elicited control points. In order to acquire ranks, we propose a process using pairwise comparisons and direct rank as discussed in Chapter 4.

3. **Implement a MAPE-k Loop Adaptation Strategy.** We use the ranked control points to build a feedback loop. The feedback loop will apply the control point to act on at any point in time in order to achieve the adaptation goal. Section 4.3 discuses our adaptation strategy algorithm.

Our final contribution is a case study that demonstrates the capability of our methodology in designing an adaptive application in cloud.We utilized a simulation tool to acquire ranks so as to evaluate the Designer Ranking. Our experimental evaluation shows that the Designer Ranking is as valid as the Experimental Ranking (ranks obtained using the simulation). Moreover, we created a set of run-time strategies and evaluated the effect of each on response time. Our result shows that the Designer Ranking is the most effective strategy. Chapter 5 is allocated to discuss our findings.

The work presented in this thesis is a groundwork for a full paper accepted at the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS). Full citation can be seen in [48].

1.4 Thesis Organization

This thesis is structured as follows. Chapter 2 provides a brief background on important concepts related to our research. Chapter 3 presents the literature review in our research field. Chapter 4 presents the details about our proposed methodology, which consists of eliciting and ranking control points. We conduct a set of experiments in Chapter 5 to assess our methodology in designing adaptive web-based applications for cloud. Finally, we summarize the thesis and present possible future work in Chapter 6.

2 BACKGROUND

This chapter provides an overview of the main materials related to our research. Since this thesis focuses on designing cloud-based adaptive web applications, we present an overview of the following areas: Cloud Computing and Autonomic Computing.

2.1 Cloud Computing

Cloud computing has emerged as a promising model with a potential of transforming the IT industry [17]. Cloud computing makes infrastructure (e.g., virtual machines), platforms (e.g., web servers, application servers) and software (e.g., e-mail) services available to clients, on-demand, using a pay-as-you-go (i.e., utility) model over the Internet. Cloud users can subscribe to these services, which are referred to as Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). Cloud computing follows the utility pricing model that charges the cloud users with respect to the utilized services, 'pay-as-you-go'. [8].

The utility model benefits both the cloud users and the cloud providers. Cloud-adopted users can reduce costs by subscribing or renting (e.g. per hour) services from service providers as opposed to buying the infrastructure, for example. The cost saving includes both IT capital and operational expenses. Since there are various service providers offering computation, storage, software as well as Service-Level Agreement (SLA) for their services, the cloud users can choose from the pool of providers whose services are cheaper or delivering a better SLA. As a result of 'on demand' delivery of services, the cloud users are allowed to adjust their usage according to their computation needs [8]. The prominent benefits of adopting cloud can be summarized in the following points:

- **Elasticity (i.e., scalability):** Stakeholders can scale their applications up and down based on the workload and computational needs.

- **Self-service on-demand provisioning:** Resources can be acquired/released as needed from/to the provider using a pay-as-you-go model.

For instance, at peak hours, sever(s) can be added in order for an application to reach its adaptation goals (e.g., response time and cost). Instances (virtual machines) can be added or removed as many as necessary; however, designers/developers must be able to determine how many instances are needed at any given point in time. Additionally, they must be aware of what type of instances (e.g., small, medium, large depending on memory, storage, etc.) they are using. Instances can also be run in multiple locations (regions), which improves the availability of their applications.

Despite advantages of cloud, there are risks (e.g., security, availability, fault-tolerance, and disaster recovery) associated with adopting cloud. For instance, security is a big setback for large companies in adopting cloud. Although cloud computing allows for lowering the cost, one should also be aware of the factors affecting the cost of running applications on the cloud. We mention a few here:

- Number of instances purchased.
- Type of instances acquired. For instance, on Amazon EC2 website, a small instance with 1.7 GB of Memory will cost less than a medium instance with 3.75 GB of Memory.
- Type of instances run in different regions (different regions charge differently). In the US East region of Amazon EC2, on-demand small instances cost \$0.06 per hour, whereas in South America (Sao Paulo) region they cost \$0.080 per hour [5].
- Network Bandwidth charges (based on number of bytes transferred).

[5] `http://aws.amazon.com/ec2/pricing/`

2.2 Autonomic Computing

As computing systems' complexity has grown, building, managing, and maintaining systems has become difficult. Information Technology (IT) environment consists of heterogeneous software, hardware, and middleware from various providers. Installing, configuring, integrating, and maintaining these heterogeneous resources are intricate tasks to be administered by IT professionals [26].

Kephart and Chess [24] state that the only solution to ever increasing complexity of software systems is to enable them to self-manage while conforming to the end-users' objectives [24]. The authors argue that systems that are able to monitor, (re) configure, (re) construct, heal and tune themselves at runtime are much needed these days for reducing the complexity of computing systems, which are operating in unstable and volatile environments.

Autonomic computing aims to automate their tasks and to eliminate human intervention in order to manage systems. For doing so, autonomic systems should possess properties of self-managing systems such as:

- **Self-configuring:** Ability of a system to configure and reconfigure itself on the fly (e.g., installing, updating, integrating software entities) [39]. For instance, new servers can be added without interrupting the service.

- **Self-healing:** Ability of a system to recover itself from disruptions and failures. Autonomic systems can detect and diagnose problems as well as to foresee them. They can also take appropriate actions to react to interruptions [39].

- **Self-optimizing:** Ability of a system to maximize resource utilization without human intervention [39].

- **Self-protecting:** Ability of a system to protect itself from any attack. Hence, preventing security breaches and malicious attacks is an important aspect of the self-protecting property [39].

Consequently, autonomic systems sustain their operation in response to changes in the environment, demands, workload, hardware and software failures, etc. They can constantly monitor their environment, check for upgrade, install, and detect errors [24].

Many researches have pointed to self-manageability or runtime adaptability as a key requirement in complex software systems [3] [19] [26] [30] [34]. To achieve self-management, a system should be monitored and analyzed. If adaptation is required, it should be planned and executed. These processes are carried out by autonomic managers, which are external to the system. In the following section, we show how these processes enable self-adaptation in adaptive software systems through feedback loops.

2.2.1 Feedback Loop

Feedback loops are fundamental mechanism for self-adaptation in adaptive systems. Self-adaptation is applied in many ways in software systems, but what is common among (self-)adaptive systems is moving design decisions to runtime to manage dynamic behavior and the systems reason about their environments [6]. For instance, in order to manage web services running constantly, a system should collect information about its environment, analyze the collected data to discover and detect performance problems and failures, decide how to resolve them, and finally execute the planned changes [6]. In this section, we explain different representations of feedback loop.

The adaptation loop adopted from Salehie and Tahvildari [39] includes monitoring, detecting, deciding, and acting processes as depicted in Figure 2.1. The adaptation process starts with monitoring process, which is in charge of collecting and correlating data from sensors to realize the current state of the system. Moreover, monitoring process provides symptoms by converting the collected data from sensors. The detecting process is in charge of analyzing the symptoms produced by the monitor in order to detect when a change is required. The deciding process is in charge of deciding what needs to be changed and how to change it. This is done by comparisons of different course of actions to apply the change. Finally, the acting process executes the changes identified by the deciding process through the effectors [39].

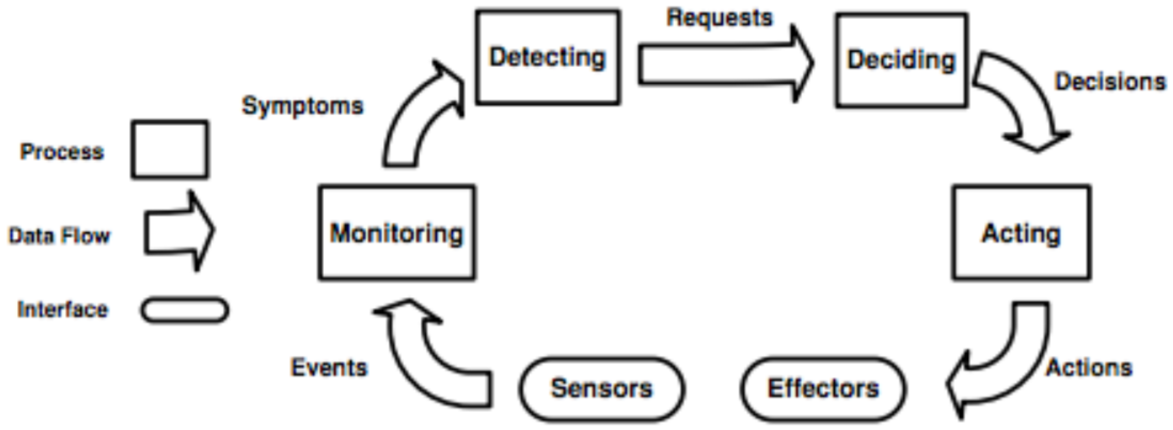


Figure 2.1: Four Adaptation Processes in Self-Adaptive Software [39].

Another version of feedback loop presented in [6], consists of collect, analyze, decide, and act as shown in Figure 2.2.

The sensors or probes are responsible to collect data from the running system and its environment. The diagnosis then analyzes the data in order to identify trends and problems. Next, a decision is made on how to modify the system in order to meet the original objectives. Finally, the system acts to execute the decision via effectors or actuators.

The autonomic element, Figure 2.3, established by Kephart and Chess [24] and then utilized by IBM's architectural blueprint for autonomic computing [12] is the

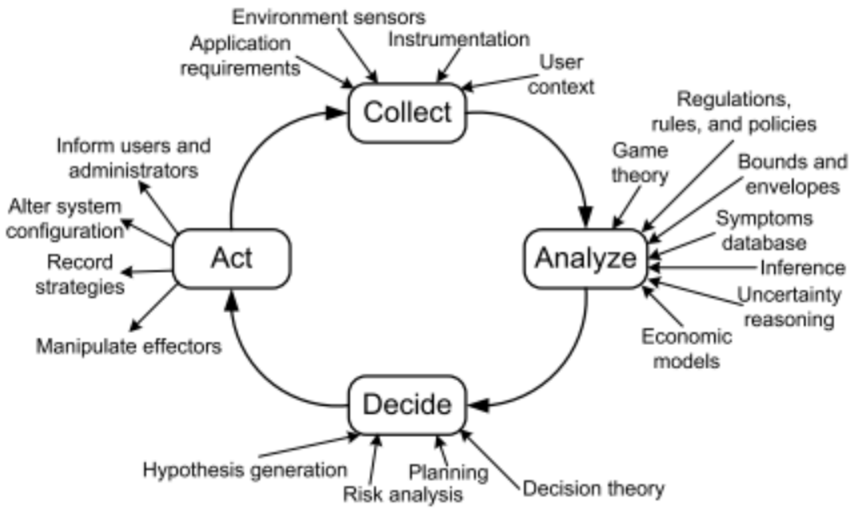


Figure 2.2: Autonomic control loop [6].

first architecture for self-adaptive systems, which emphasizes the explicit feedback loop. The autonomic element is used as a reference model for autonomic control loops to manage autonomic computing as suggested by IBM [12].

The autonomic element has two parts: Autonomic Manager and Managed Element, which are at the core of the autonomic loop.

Autonomic Manager implements the feedback loop as well as controlling the Managed Element. The manager consists of two manageability interfaces: sensor and effector, and monitor-analyze-plan-execute (MAPE-K) engine composing of a monitor, an analyzer, a planner, and an executor that share a common knowledge base.

The monitor senses the managed process and its environment, filters the gathered

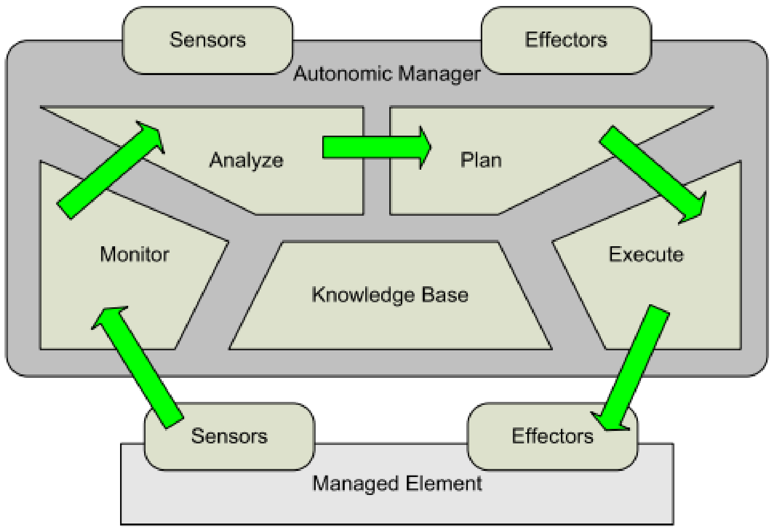


Figure 2.3: IBM's autonomic element [6].

data, and stores them in the knowledge base for future reference. The analyzer compares the gathered data with exiting patterns in knowledge base to detect symptoms, and stores the symptoms in the knowledge base for future reference. The planner infers these symptoms and creates a plan to execute a change in the managed process through effectors. Hence, planning involves producing adaptation plans based on the monitored data from the sensors. The Autonomic Manager collects measurements form the Managed Element as well as the information about the past and the current states from the knowledge base to adjust the Managed Element if required.

Managed Element consists of resources such as operating system, CPU, web server, database, etc. Sensors, which are sometimes called probes or gauges, are in charge of collecting information about the Managed Element. For instance, sensors collect information on response time, CPU utilization, and network usage, and so on for a web-server. On the other hand, effectors execute changes to the Managed Element. Example includes adding or removing instances or changing configuration in a web server [22]. The Autonomic Manager monitors the Managed Element through collected data provided by the sensors, and executes changes through effectors. The Autonomic Manager is a software component configured using high-level goals by administrators. These goals are expressed as event-condition-action (ECA) policies, goal policies, or utility function policies [25].

One of the main influences in designing adaptive systems is the use of feedback loops. For instance, Garlan et al. [19] proposed an architectural model, called Rainbow, in order to support all aspects of the self-management capabilities aiming at optimizing a system's performance and cost. Using an architectural model, the framework monitors the system and decides on suitable adaptations. Adaptation policies are explicitly defined in Rainbow, which enable the engineers to reuse these policies in similar systems.

Besides Rainbow, another architectural model based on MAPE-K loop is Menasce et al. [32]'s framework for self-architecting service-oriented systems. The authors' framework, Self-Architecting Software Systems (SASSY), aims to dynamically adapt the architecture in order to maintain QoS goals. The monitoring component in SASSY collects QoS metric values and sends it to the analyzer. The analyzer aggregates these data and calculates the utility of the system. The system then sends a request to the planner in case the system's utility is not satisfied by the stakeholder-specified threshold. The architecture planner then automatically identifies 'near-optimal' architecture. Changes to the system in operation are then executed through adaptation patterns by self-adaptation component.

2.3 Summary

In this chapter, we covered important concepts relevant to our research. We described the advantages of cloud computing provisioning on-demand services over the internet, and as a result providing the cloud users with a variety of services and cloud providers to choose from. We also highlighted some of the factors affecting the performance and cost of running applications on the cloud, which are essential to consider when designing an adaptive system for cloud. We then defined autonomic computing and explained the self-* properties of autonomic systems. In the end, we discussed different representations of feedback loop that are counted as the main force behind the adaptations in adaptive software systems. Moreover, we showed how they are utilized in software architectures to fulfill self-management capabilities.

3 RELATED WORK

A key challenge posed by autonomic computing [18] [24] is to manage systems to handle uncertainty in their execution and environment. The growing demand for self-adaptation has caused a tremendous attention in software engineering for self-adaptive systems. A good example is the comprehensive research roadmap by Cheng et al. [10] on software engineering for self-adaptive systems, which discusses the research challenges posed in the fields of modelling, requirements, and engineering of self-adaptive systems. Software architectures have been widely used in the literature to provide flexible adaptations [26] [19] [34]. In addition, there are works in modeling and monitoring requirements for adaptive systems. Goal modeling approaches such as i* [46] and KAOS [16] have been applied to model and reason about runtime adaptations as well as exploring alternative requirements when system's environment changes [20] [35] [2].

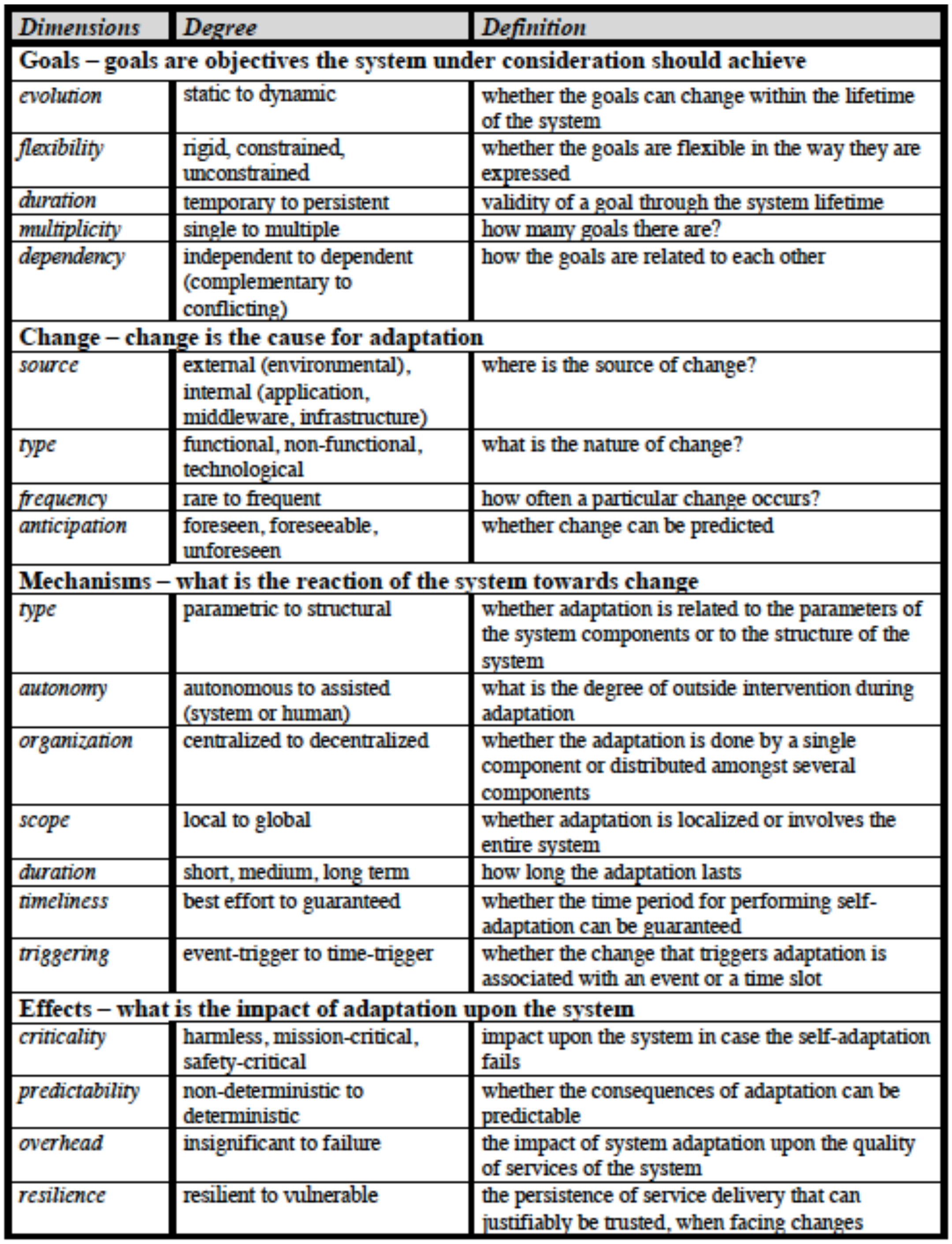
In this chapter, we present the relevant literature on adaptive systems from different perspectives. In Section 3.1, we review the state-of-the-art on modelling adaptive software and how it relates to our research. We present goal-oriented approaches to design adaptive systems in Section 3.2. Section 3.3, reviews the literature in elicitation of adaptation requirements. Finally, Section 3.4 summarizes the chapter.

3.1 Modeling Adaptive Software Systems

Adaptive systems are built based on several aspects of the software such as system's goals, characteristics of the environment, the end-user's needs, etc. For an adaptive system to embody the mentioned aspects, accurate modeling of the system is necessary [10]. Therefore, self-adaptive systems are developed depending on a conceptual model of adaptation regardless of the technology and tools used for its implementation.

Andersson et al. [1] proposed modeling dimensions that describe various aspects of adaptation, which allow for engineers to identify properties of self-adaptation and select a proper solution. Their study aims to identify and compare important aspects of self-adaptive systems. Thus, this classification of modeling dimensions needs to be contemplated when modeling an adaptive system. Figure 3.1 adopted from [1] summarizes the modeling dimension for self-adaptive systems.

The authors categorize the points of variation, modeling dimensions, into four classifications. The first classification, *Goals*, is associated with achieving the objec-



Dimensions	*Degree*	*Definition*
Goals – goals are objectives the system under consideration should achieve		
evolution	static to dynamic	whether the goals can change within the lifetime of the system
flexibility	rigid, constrained, unconstrained	whether the goals are flexible in the way they are expressed
duration	temporary to persistent	validity of a goal through the system lifetime
multiplicity	single to multiple	how many goals there are?
dependency	independent to dependent (complementary to conflicting)	how the goals are related to each other
Change – change is the cause for adaptation		
source	external (environmental), internal (application, middleware, infrastructure)	where is the source of change?
type	functional, non-functional, technological	what is the nature of change?
frequency	rare to frequent	how often a particular change occurs?
anticipation	foreseen, foreseeable, unforeseen	whether change can be predicted
Mechanisms – what is the reaction of the system towards change		
type	parametric to structural	whether adaptation is related to the parameters of the system components or to the structure of the system
autonomy	autonomous to assisted (system or human)	what is the degree of outside intervention during adaptation
organization	centralized to decentralized	whether the adaptation is done by a single component or distributed amongst several components
scope	local to global	whether adaptation is localized or involves the entire system
duration	short, medium, long term	how long the adaptation lasts
timeliness	best effort to guaranteed	whether the time period for performing self-adaptation can be guaranteed
triggering	event-trigger to time-trigger	whether the change that triggers adaptation is associated with an event or a time slot
Effects – what is the impact of adaptation upon the system		
criticality	harmless, mission-critical, safety-critical	impact upon the system in case the self-adaptation fails
predictability	non-deterministic to deterministic	whether the consequences of adaptation can be predictable
overhead	insignificant to failure	the impact of system adaptation upon the quality of services of the system
resilience	resilient to vulnerable	the persistence of service delivery that can justifiably be trusted, when facing changes

Figure 3.1: Modeling Dimensions for Adaptive Systems [1]

tives of a system. The second classification, *Change*, is associated with the cause of change in a system. The third classification, *Mechanism*, deals with mechanisms to achieve change in a system, and the fourth classification, *Effects*, deals with the effect of adaptation on a system. Within each group, the authors have identified several dimensions, which focuses on specific parts of the system that is pertinent to self-adaptation. For instance, the dimensions associated with *Goals* are:

- **Evolution:** This dimension is associated with goals of the system in two ways: static and dynamic. Moreover, the dimension evaluates if goals can change during the lifetime of the system. Static goals are not changeable whereas dynamic ones can change at runtime, i.e., system generates new goals.

- **Flexibility**: This dimension deals with the level of uncertainty associated with goals. Rigid goals are prescriptive while unconstrained goals are flexible in dealing with uncertainty. Finally, constrained goals provide flexibility as long as certain constraints hold.

- **Duration:** This dimension is related to goal validity during lifetime of the system. Persistent goals are valid all the time within the lifetime of adaptive system whereas the temporary ones are valid for a specific time range such as short, medium, and long. For instance, in our approach, our adaptation goal is considered to be persistent.

- **Multiplicity:** This dimension deals with the number of goals related to adaptivity aspect of the system. For example, our adaptive system has a single goal to achieve.

- **Dependency:** This dimension deals with how multiple goals in a system relate to each other. Goals can be independent of each other or they can depend on each other.

While Andersson et al. identify the challenges in modelling adaptive systems, our work is a concrete study of the dimensions and a methodology to guide the design.

Recently, Brun et al. [5] introduced the concept of design space for adaptive systems, which contains key questions when attempting to design a self-adaptive system. The authors present a conceptual model of how to identify different components of an adaptive systems by answering a set of questions along five dimensions: identification, observation, representation, control and adaptation mechanisms. For each dimension, they identify key design decisions. Example of questions: ”what information is made available to the components of the adaptive systems?”, ”how does the system decide what and how much to change to modify its behaviour?” , etc. We used the authors' approach as a blueprint for designing adaptive systems. Our work focuses on the control points as being the main artifacts that derive the design of feedback loops. Therefore, our proposed method enhances other methodologies.

Moreover, we focus on NFR goals in the context of cloud, and we use quantitative methods to guide the design.

3.2 Goal-Oriented Approaches to Adaptive Systems

The field of Requirements Engineering (RE) for adaptive systems is a wide open research area [9]. Designing software systems, which deals with incomplete, vary, uncertain requirements at runtime multiplies the difficulties of requirements engineering. Dealing with uncertainty is one of the key challenges in designing adaptive systems since all possible adaptations are not known in advance. This is due to the fact that we cannot realize all sets of environmental conditions and their relative adaptation specifications. Therefore, requirements for adaptive systems are dealt with uncertainty and incompleteness [10]. In this section, we review some goal-oriented approaches to adaptive systems.

Goal-Oriented Requirements Engineering (GORE) is a distinguished approach within RE. The notion of goal has been used extensively in RE. Goals, "desired states-of-affairs", are stakeholders' needs, which are elicited, modeled, and analyzed. In GORE approaches, goals are captured by goal models. Goal models are effective in identifying system's requirements by studying the stakeholders' intentions [47] [46] [16]. Hence, they can be used as a communication medium between the requirements engineer and the stakeholders.

One example of goal-oriented modeling in adaptive requirements is [27]. Lapouchnian et al. [27] used goal models to develop an adaptive software system. Their approach supports using requirements goal models to support all configurations in order to enable a system to select the best behaviour at runtime based on the current system's environment. They also show that an autonomic system architecture can be derived from the goal model. In their approach, the authors add annotations such as priority, conditions as well as contributions to identify the best configuration of the system to meet its main goal.

Souza et al. [41] proposed a new class of requirements, called awareness requirements. Awareness requirements refer to success and failure of other requirements or domain assumptions at runtime, and identify the critical requirements, whose success/failure should be aware as well as the situations in which the system is adapted. The authors provide a monitoring framework to monitor awareness requirements. When an awareness requirement fails at runtime, the adaptation strategy modifies the parameter's value to improve the failed indicator. Their approach relies heavily on monitoring requirements and switching the system's behaviour in case of failures. Our approach is different from [41] in that our adaptation strategy algorithm detects discrepancies by monitoring the adaptation goal as oppose to detecting the violation of monitored requirements caused by changes in context.

Furthermore, Souza et al. extended their work in [43] to introduce evolution requirements. The authors introduced a process to execute adaptation strategies in response to the system's failure. They model evolution requirements as Event-Condition-Action (ECA) rules, in which the rules are activated if a certain condition holds when an event occurs. Therefore, evolution requirements specify strategies and other requirements that need to be changed at runtime to fulfill the stakeholders' objectives.

Our work is different from the above works. Our proposed adaptation strategy, in contrary, monitors the adaptation goal, and measures the value of a metric against the pre-defined one. It then selects the most effective operation (based on ranking) in meeting the objectives. Moreover, our adaptation strategy uses multiple control points to accomplish the adaptation goal of the system.

Salehie and Tahvildari [38] proposed a decision-making mechanism for selecting adaptations at runtime. Their decision model consists of Goal, Action, and Attribute (GAAM), which aims at a goal-driven approach to make runtime decision based on adaptation goals, which are explicitly defined. Their framework selects the best appropriate Actions to satisfy Goals under different conditions (Attributes).

We can relate our work to [38] in that we design an adaptive software by explicitly achieving the adaptation goal. Their framework also deals with multiple adaptations. However, the authors utilize a weighted voting mechanism to select the appropriate adaptations, and only consider application-level adaptations. We use a different approach in selecting appropriate control points by direct ranking. Another difference is that our work supports adaptations at different levels (e.g., application, cloud, multi-cloud[6].

Qureshi and Perini [36] proposed a framework for Continuous Adaptive Requirements Engineering (CARE) supporting self-adaptive service-based applications. The authors point out that service-based applications require continuous reappraisal of requirements, which means refining and adding new requirements. Their CARE framework is tailored to address such issues by enabling the system to perform continuous RE with respect to the users goals and preferences. Moreover, users can add/modify new requirements at runtime.

The authors differentiate between the activities involved at design-time and run-time. RE at design time involves activities ranging from eliciting stakeholders intentions to specifying requirements. RE at run-time is performed by the system itself involving the user to deal with continuous changes. Thus, at runtime, new requirements are emerged, given by the end user or the system monitoring the en-

[6]The multi-cloud refers to a combination of computing resources from multiple clouds.

vironment/end user. The authors call requirements at runtime as service requests, which entail functional goals, quality constraints, preferences, context information, and son on. In our approach, we also separate the design time activities from runtime. At design time, we elicit control points as well as ranking them, and at runtime, our adaptation strategy, implemented by MAPE-k loop, select the most appropriate adaptation for the runtime system.

3.3 NFR Elicitation for Adaptive Systems

Requirements elicitation, as a first step in requirements engineering, is a crucial task, which studies the needs of different stakeholders, the software system, context, etc. The success of a system depends on how well its requirements are elicited. Requirements engineer is responsible to elicit both functional (what the system should accomplish) and non-functional (constraints on functionalities of system) requirements. Regrettably, there have been a lot of attentions to functional requirements and treating them as first class requirements while the non-functional requirements have been neglected and only being dealt with in design and implementation phases [44].

The most popular work in specifying and analyzing NFRs is the NFR Framework [11]. The importance of NFRs has been stressed in the NFR framework, proposed by Mylopoulos et al. [33] and further developed in [11]. The framework has been developed to model and analyze NFRs. During software development, NFR framework helps the developer/architect to make design decisions by treating NFRs as selection criteria. The aim of the framework is to capture NFRs of the system-to-be, decompose them, detect possible operationalization (design alternatives) and select them, deal with priorities, tradeoffs, ambiguities, and interdependency among the NFRs, support decisions with design rationales, and evaluate the impact of decisions. Thus, the NFR framework is used to model and decompose NFRs and to explore different design alternatives in relation to NFRs. Moreover, each design alternative contributes to achieving goals either positively or negatively. The NFR framework exposes the impact of each design decision on these requirements.

As with goal modelling methodology, goals in NFR Framework (denoted by softgoal) are decomposed using AND or OR refinements. Interdependencies between softgoals are captured through positive and negative contribution links. Such model embedded with these concepts is called Soft-goal Interdepency Graph (SIG). The SIG graphically illustrates softgoals and their AND/OR refinements, sofgoals positive/negative contributions, operalizations and claims. Sofgoals are recursively refined until they cannot be further decomposed, which is reaching operationalization softgoals. Our elicitation method also has a tree structure, but only deals with one adaptation goal at a time. We will thoroughly describe our method in Chapter 4.

Jiang and Yang [23] proposed performance requirements elicitation from different stakeholders such BAs and developers based on cognitive approach. The authors introduced a technique to elicit performance requirements for financial information system based on ontology. Their method splits the requirements into three parts: system level, subsystem level and component level. For each level, they then define metrics and range. Although their work benefited us to identify metrics at different granularity level, the purpose of our work is not eliciting NFRs, but eliciting the operations that can be mapped to NFRs.

Taxonomies have also been used in the literature to depict different aspects of self-adaptation. Brake et al. [4] classified parameters based on their behaviour. The authors created a taxonomy from different kinds of tuning parameters in adaptive systems. In their approach, Brake et al. employed a method to automatically discover tuning parameters in the source code. They analyzed user/system documentation to compile catalogue of parameters.Then the syntactical search of the source code is performed to find fields that match the parameters. The authors aimed to use the tuning parameters as effectors in autonomic systems. Such taxonomy of parameters helped them to automate the identification of tuning parameters in existing software systems. Furthermore, Ghanbari and Litoiu [21] extended the classification proposed in [4]. Basing their approach on reverse engineering, the authors proposed a technique to identify tuning parameters in software systems. They categorized the taxonomy of tuning parameters into control (output parameters) patterns and reflexive (input, state, environment parameters) patterns.

Salehie and Tahvildari [39] matched the runtime adaptation changes to software evolution based on Buckley et al. [7]'s taxonomy. The authors proposed a taxonomy of adaptation that covers questions of *where, when, what, why, who,* and *how* behind self-adaptive software systems. The questions are helpful in eliciting adaptation requirements. For instance, *The Object to adapt* facet in their taxonomy covers the *where* and *what* aspect of change, which is further refined to three sub-facets such as:

- **Layer:** Which layer of the system needs to be changed and can be changed? Different adaptation actions can be applied to different layers.
- **Artifact and Granularity:** What artifacts and at which level of granularity needs to be changed? This reflects the module and architecture.
- **Impact and Cost:** This facet reflects the impact of change on the system and its cost based on time, resources, etc. Based on impact and cost factors, the adaptation actions can be defined as weak and strong classes. Weak adaptation deals with changing parameters and low cost/limited impact actions whereas strong adaption deals with high cost/extensive-impact actions [66]. Examples

of weak adaptation include bandwidth limit, load balancing, etc. Adding, removing, replacing components are examples of strong adaptation.

In our elicitation method, we implied the *what*, *how*, and *where* aspects of adaptation to formulate a series of questions to facilitate the elicitation of control points. Moreover, our method elicits both weak and strong operations (control points).

Using multiple control points to adapt the system has been attempted previously [40] [29]. For example, Litoiu et al. [29] proposed a hierarchical model-based adaptation for tuning parameters in service-oriented architecture applications. Their architecture consists of a hierarchy of controllers (Component Controller, Application Controller, Provisioning Controller). Each layer has its own model and evaluates decisions before executing any change. The authors present the need for having multiple control loops for non-functional requirements. They show a general architecture of how these loops can work at different levels of granularity. The above works are beneficial although they do not focus on eliciting operations or ranking them. Neither do they consider strategies for combining multiple control points.

3.4 Summary

In this chapter, we reviewed the state-of-the-art related to our research. We represented some important works in designing adaptive systems, and identified the differences and similarities to our approach. We showed how we adopted some concepts from modelling and designing adaptive systems from the literature. In the next chapter, we introduce our methodology and the main contributions.

4 METHODOLOGY FOR ELICITING AND RANKING CONTROL POINTS

In this chapter, we present our approach for designing cloud-based web applications by creating an adaptation loop. Our target is web applications that are similar to Znn.com[7] architectural style [19].

We present a methodology, which discovers a set of control points, derived from iteratively refining an adaptation goal, and utilizes them in a rank order for building an adaptation loop. In this thesis, we assume the adaptation goal (such as response time below a threshold) and resource boundaries (such as x number of virtual machines) have been already identified by the application's stakeholders. Thus, our task is to design and implement the feedback (or adaptation) loop that achieves the adaptation goal.

We introduce a process for eliciting, ranking, and executing control points. Our

[7]http://seams.self-adapt.org/wiki/Exemplars

elicitation methodology decomposes an adaptation goal into a hierarchy structure as described in Section 4.1.1. Having the control points elicited, we then propose a ranking methodology to prioritize them. Our ranking method utilizes pairwise comparisons and direct ranking of control points. To acquire the final ranks, we aggregate the result. Finally, using MAPE-K loop, we design an adaptation loop.

This chapter has the following structure. In Section 4.1, we discuss our elicitation method. Section 4.2 demonstrates our ranking process. We explain our adaptation loop algorithm in Section 4.3. Finally, we summarize the chapter in Section 4.4.

4.1 Elicit Control Points

The process of eliciting control points is somehow similar to NFR elicitation. One such approach is the NFR Framework [11], in which higher level goals are decomposed into sub-goals using AND/OR trees until they cannot be decomposed any further (operationalized). In our approach, we propose control point models to assist us with identifying and eliciting control points. To elicit control points, one can build a control point model as will be discussed in 4.1.1 or reuse our catalogues (see Section 4.1.2), or the combination of both. This section is allocated to control point models, how to construct one, and their reusable functionality through catalogues.

4.1.1 Control Point Model

A control point model is a tree with the following properties:

1. The root node is the stakeholders' adaptation goal, represented as a double boundary rectangle.
2. The model expresses the relationship between the root node and the operations (leaf nodes) that is required for meeting the high level goal, the adaptation goal. This understanding gives the reason why operations are essential.
3. The complex adaptation goal is decomposed into different sub-goals.
4. A sub-goal is represented by a cloud.
5. The operations are denoted by rectangles.
6. (Optional) Adaptation goal or sub-goals can have a metric attached to them. (Metric is denoted by a diamond).

The construction of a control point model is an iterative process. This systematic approach helps decompose a complex adaptation goal/sub-goals into smaller sub-goals. It also helps reduce the risks associated with missing operations or misunderstanding adaptation goal/sub-goals.

A difference between our control point model and the NFR Framework is that our control point model represents the decomposition of one adaptation goal as oppose to refining multiple NFRs as in soft goal interdependency graph (SIG). Therefore, our proposed control point model is conflict-free by design since all operations/sub-goals only contribute for meeting the top-level adaptation goal.

Applications deployed in the cloud also have the feature to have environmental parameters dynamically configured in addition to conventional application parameters. Therefore, to simplify the process of control points elicitation, different control point models can be constructed on Application, Cloud, and Multi-cloud. We extend a control point model by asking a series of questions for Application, Cloud, and Multi-cloud as follow:

- **Cloud.** How does cloud agility/elasticity help an application to meet its objectives? What services offered by the cloud provider can aid in meeting the adaptation objectives? For instance, consider the adaptation goal is less than 1 second response time. Amazon Web Services (AWS) Elastic Beanstalk [8] is an Elastic Compute Cloud (EC2)[9]service, which provides auto scaling capability for a web application. We can define a control point, which enables auto scaling using the AWS Elastic Beanstalk service.

[8]`http://aws.amazon.com/elasticbeanstalk/`

[9]`http://aws.amazon.com/ec2/`

- **Multi-cloud.** How will migrating from one cloud to another one align the application with its adaptation goal? How does utilizing additional clouds help the application to meet the stakeholders' objectives? Assume the adaptation goal is less than 1 second response time and the application will be deployed on a private cloud. Utilization of public cloud will enable massive scaling by bursting application process into public cloud. We can define a control point, which will enable the offloading of non-sensitive service into a public cloud [42].

- **Application.** Which web application scenarios exhibit similar runtime behaviour in relation to adaptation goal or sub-goal? How can management systems control the runtime behaviour of these scenarios? What configuration parameters (existing or defined) will control the runtime behaviour? For instance, assume the adaptation goal is less than 1 second response time. Often response time of web applications depends on the number of threads defined for the application server. In this case, all performance scenarios of the web application will, to some degree, depend on the number of threads. Hence, the number of threads is a control point.

Figure 4.1 and 4.2 show our elicited control points for meeting a low cloud cost in cloud and a low response time in application environment respectively. As shown in Figure 4.1, in order to elicit control points, we first decompose the adaptation goal into a metric, the cost of runtime system. Next, thinking of what influences the cost in cloud, we decompose the adaptation goal into sub-goals such as Consolidate resources, Decommission of under-utilized resources, or Find cheaper resource/service. Now that we have our subgoals, we decompose them further to reach the control points. Looking at the decommission of under-utilized resources sub-goal, we can remove or change VM instance types to achieve a lower runtime cloud cost. Hence, removing a web server or changing its size to a smaller one are the examples of control points.

4.1.2 Control Point Catalogue

Creating catalogues to assist with eliciting NFRs have been used in [15] [11] [14]. Cysneiros conducted an experimental study to investigate the use of catalogues in eliciting NFRs [13]. The result showed the higher number of operationalizations were found by using catalogues. Inspired by these works, we built catalogues of control points for application and cloud environments, which can be extended and/or reused as an alternative way to elicit control points.

Table 4.1 shows the most important NFRs for web-applications that can be met through adaptation.

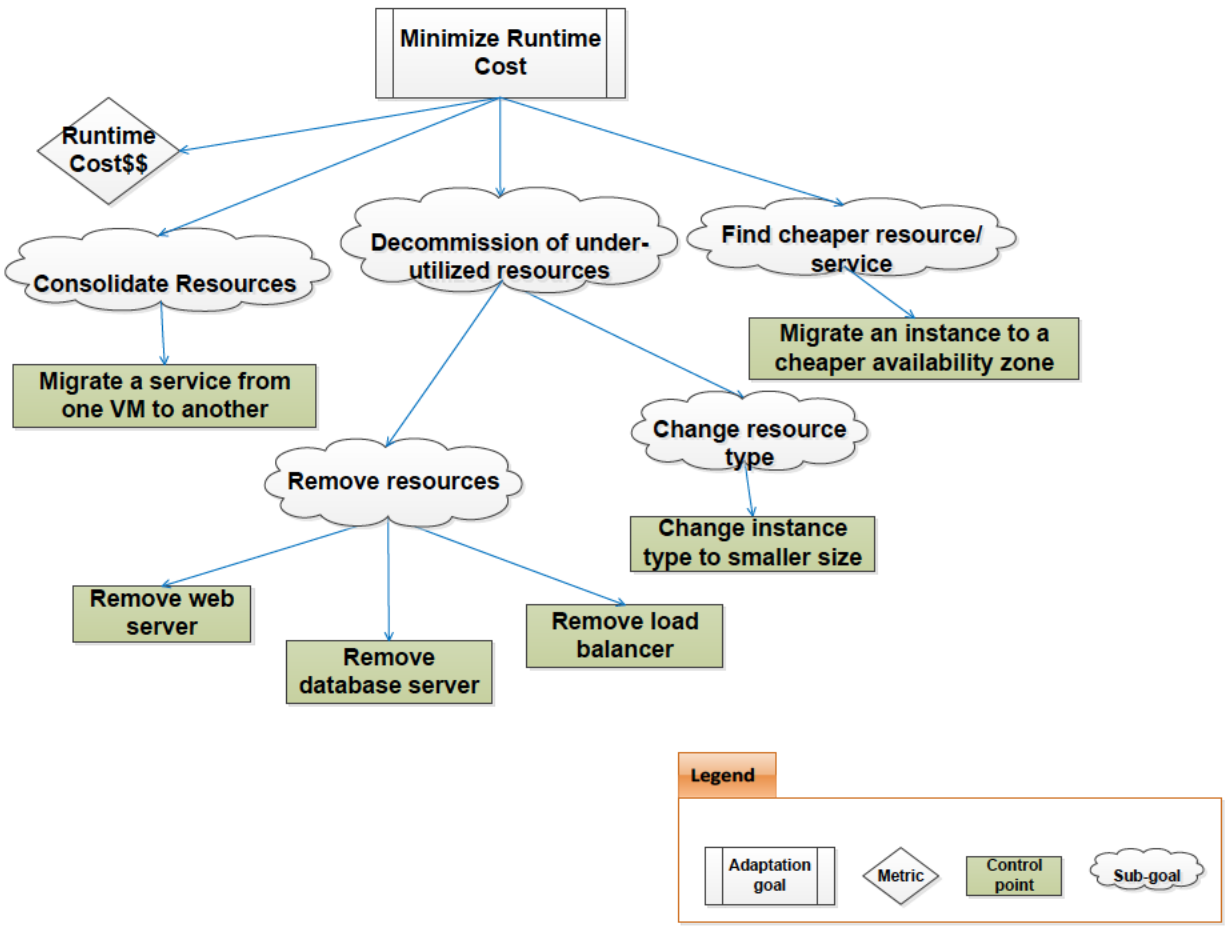


Figure 4.1: Control Point Model for Runtime Cloud Cost

Table 4.2 shows the control point catalogue for application performance, and Table 4.3 displays the control point catalogue with respect to different NFRs for cloud. All the control points listed in Table 4.3 are applicable for web servers, load balancers, and database servers. Therefore, these control point catalogues can be reused for elicitation purposes.

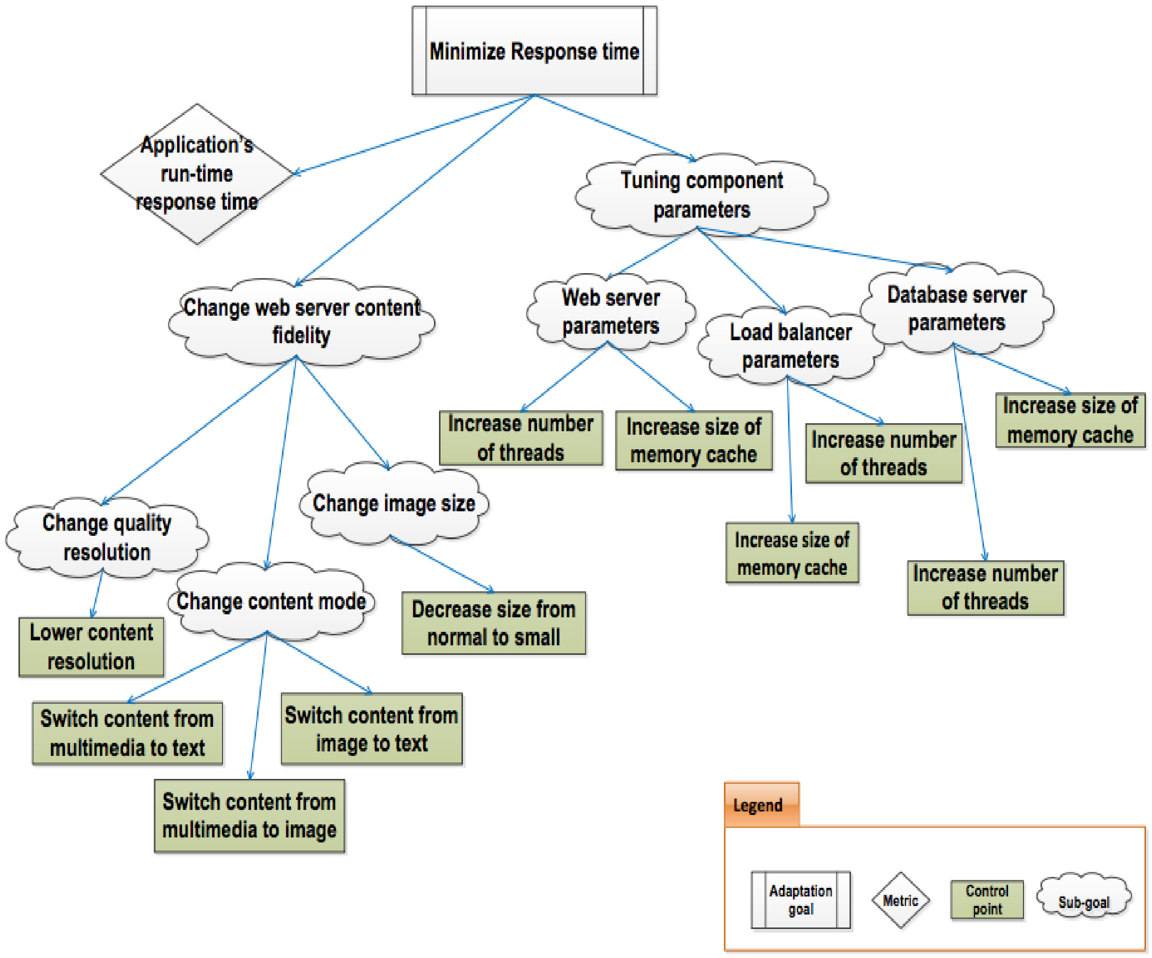

Minimize Response time
Application's run-time response time
Tuning component parameters
Change web server content fidelity
Web server parameters
Load balancer parameters
Database server parameters
Increase number of threads
Increase size of memory cache
Increase number of threads
Increase size of memory cache
Increase size of memory cache
Increase number of threads
Change image size
Change quality resolution
Lower content resolution
Change content mode
Decrease size from normal to small
Switch content from multimedia to text
Switch content from image to text
Switch content from multimedia to image
Legend
Adaptation goal
Metric
Control point
Sub-goal

Figure 4.2: Control Point Model for Application Response Time

NFRs	**Metrics**
Performance	Response time, servers' utilization, throughput
Cost	Total no. of machines, no. of users, system's runtime cost
Security	Users' activity, credit card/IP velocity
Availability	Latency, abandon rate

Table 4.1: NFRs and their corresponding Metrics

Control Points for Performance
Lower content resolution
Switch content from multimedia to text
Switch content from multimedia to image
Switch content from image to text
Decrease image size from normal to small
Increase number of threads
Increase the size of memory cache

Table 4.2: Elicited Control Points for performance goal (application)

4.2 Rank Control Points

This section presents how to rank the elicited control points. A rank reflects the relative impact of a control point on the adaptation goal. For example, a control

Control Points	**Performance**	**Cost**	**Security**	**Availability**
Add bandwidth	x			
Add instances to different regions/different cloud providers	x			x
Change instance type to smaller size		x		
Change instance type to bigger size	x			
Migrate instances to cheaper zones/cloud providers		x		
Migrate instances to public cloud provider	x			
Migrate instances to private cloud		x	x	
Migrate instances to virtual private cloud			x	
Migrate a service from one VM to another		x		
Reduce latency by migrating instances closer to each other	x			
Reduce network latency	x			
Remove instances		x		
Remove bandwidth		x		

Table 4.3: Elicited Control Points for adaptation goals (cloud)

point with rank 1 means that control point has the highest positive impact on the quality of service under control. We introduce the following steps in order to acquire ranks from a group of designers.

1. **Conduct pairwise comparisons.**

   We use pairwise comparisons as in the Analytic Hierarchy Process (AHP) [37], which compares two alternatives against another based on the higher level goal. To conduct pairwise comparisons, the designers are asked to choose between three options ($A > B$ meaning A is more preferred to B; $A < B$ meaning A is less preferred to B (or B is more preferred to A), and $A = B$; meaning A and B are equally preferred). We assign labels "More preferred", "Less preferred", and "Equally preferred" in order for the designers to perform comparisons. Each designer populates the upper triangle of a matrix[10] with the labels indicating their preferences. We then test to ensure validity of the values. If the values are not logically consistent with each other, the designers are asked to review their preferences until the values are consistent.

2. **Conduct direct rank.**

   Using a scale of 1 to n (n = number of control points), the designers rank the control points from the most effective to the least effective with respect to

[10]We ignore the lower triangle matrix since they are reciprocal values.

an adaptation goal. For instance, if there are 5 control point alternatives, the designers use a scale of 1 to 5 to rank them. They are also given an option to use a number more than once to show their equal preference between two or more alternatives.

The reason for conducting pairwise comparisons prior to direct ranking is that the pairwise comparisons help the designers build a mental model about the influence of control points on the adaptation goal. The consistency of pairwise comparisons help to measure the quality of the designers' mental model. After the designers build a good mental model, they are ready to perform the direct rank. The result of the direct ranks obtained from the designers is then aggregated as shown in the next step.

3. **Aggregate the obtained ranks.**

   To obtain the final rank, we count the number of times each control point was ranked 1st, 2nd, 3rd, etc. The rank of the control point is the most agreed rank among all designers. If two control points happen to have the same rank, in order to solve the conflict, we look at which control point gets more agreements by the designers. The ranked control points are added to an ordered list, CP-List.

Having the control points ranked, now a decision can be made on how many control points to consider for implementation. This decision is not within the scope of this thesis, we assume all are implemented.

4.3 Implement a MAPE-k loop Adaptation Strategy

Our adaptation strategy is implemented by a MAPE-k loop. The MAPE-k loop monitors the application's metric associated with the adaptation goal and analyzes it. The MAPE-k loop actions are triggered periodically or when a substantial discrepancy is detected between the desired metric and the measured one. Then the adaptation algorithm is executed as shown in Algorithm 1.

The adaptation algorithm (See Algorithm 1) reduces the gap between the measured and desired metric by iteratively changing a single control point until no further improvement can be found. It then moves to the next control point. In Algorithm 1, `CP` is the control point, `CP-List` is the ordered list of elicited control points aiming to achieve the adaptation goal, `R-Consumption` is the application's resource consumption, and `R-Boundary` is the boundary for resources defined by the application's stakeholders.

The algorithm starts by selecting the highest ranked control point for achieving the adaptation goal. Next, the selected control point is changed until the adaptation

Algorithm 1: Adaptation loop algorithm.

```
begin
    repeat
        is-any-improvement=false
        foreach CP from CP-List do
            repeat
                measure R-Consumption
                if R-Consumption[CP]+$STEP_R(CP)$ <R-Boundary[CP] then
                    is-executable = true
                end
                else
                    is-executable=fales
                end
                if is-executable AND Goal is NOT met then
                    execute CP operation
                    Store the value of the metric into m
                    if m< pre-defined threshold then
                        is-improved = true
                        is-any-improvement=true
                    end
                    else
                        is-improved = false
                    end
                end
                else
                    is-improved = false
                end
            until is-improved AND is-executable;
        end
    until is-any-improvement;
end
```

goal is satisfied or no further improvements in the system's behaviour can be found (no further improvements means the modification of the selected control point has no further effect on the adaptation goal or the pre-defined resource boundary is reached). Then the adaptation algorithm selects the next control point and repeats the cycle.

The algorithm also checks whether a control point can be executed or not based on the current and future resource consumptions (See line 7 in Algorithm 1). $\texttt{STEP}_R(CP)$ in line 7 is the amount of increment or decrement of changing the control points that impacts the (future) resource consumption, and `R-Consumption[CP]` is the current resource consumption. The algorithm measures the current resource consumption and checks whether $\texttt{R-Consumption[CP]+STEP}_R(CP)$ is less than the pre-defined boundary. If the amount is not within the resource boundary, the control point cannot be executed and the algorithm selects the next control point from the `CP-List`.

4.4 Summary

In this chapter, we introduced our approach to design an adaptive cloud-based web application by performing the following tasks. First, we proposed a control point model to elicit control points, and collected them in a table as a catalogue for reuse purposes. Second, we introduced a process to acquire ranks for control points from the designers, where we used the pairwise comparisons and direct ranking. Third, we designed an adaptation strategy, which was implemented by MAPE-K loop in order for a runtime system to meet its adaptation goal. Therefore, in this chapter, we showed that eliciting and ranking control points are imperative steps in creating a runtime strategy. In the next chapter, we conduct a series of experiments to assess the capability of our proposed methodology in designing adaptive web based applications for cloud.

5 CASE STUDY: ADAPTIVE SHOPPING CART FOR CLOUD

This chapter presents a case study, the design of an adaptive shopping cart web application for cloud. In this case study, we evaluate our methodology by following the steps described in Chapter 4. We focus on two important issues: the validation of control point ranking and the validation of the adaptation strategy algorithm.

The chapter has the following structure. We define a set of control points and ask a group of designers to rank them in 5.1. In 5.2, we show how we validate the control points ranking. In 5.3, we discuss the efficiency of the adaptation algorithm using the ranking. We summarize the chapter in 5.4.

5.1 Control Points for Online Shop

Consider a shopping cart system, Online Shop, with a typical multi-tier client-server architecture. The architecture consists of a load-balancer, web servers, and database servers. The stakeholders of the Online Shop identified response time less than 500 ms as the application's adaptation goal. They also defined boundaries for resource consumption as depicted in Table 5.1. The **Min** and **Max** columns in Table 5.1 reflect the minimum and maximum boundaries for resources.

Request no. varies from 100 to 10,000 requests.

Name	**Min**	**Max**
Number of web servers	1	50
Instance size	1	4
Disk size	1	4
Latency	10 ms	500 ms
Bandwidth	100Mb/ms	1 Gb/ms
Number of Threads	Request no./2	Request no.

Table 5.1: Resource Boundaries for Online Shop

In this section, we follow the steps to define the control points and the ranking of control points.

5.1.1 Elicit Control Points

We selected control points from the catalogues for application and cloud (see Table 4.2 and Table 4.3). Table 5.2 shows the control points selected for this study.

Control Points	**Definition**
Add more Bandwidth	Add more bandwidth between web server and database server
Change Instance type	To increase instance size for web server cluster
Change Disk type	To increase disk size for web server cluster
Reduce Network Latency	To decrease network latency between the servers[a]
Increase No. of Threads	To increase number of threads in the web server cluster
Add web servers	To add additional web servers to the application environment

Table 5.2: Elicited Control Points and their definition

[a]Web server and database server

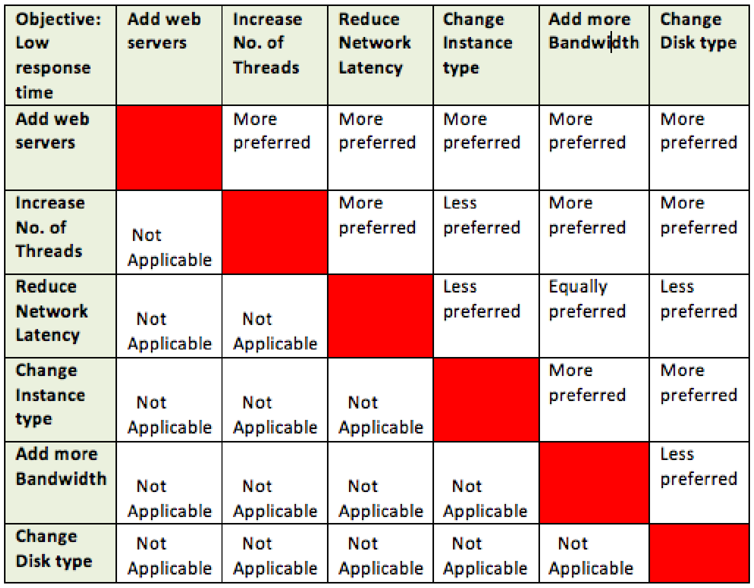
5.1.2 Rank Control Points

We asked a team of 9 researchers to assist us with designing the adaptations for the Online Shop application. Our participants were graduate students from IT and Computer Science Programs, as well as postdoctoral fellows. They all have practical experience in cloud and adaptive software domains. The participants were given the architecture of the application and a scenario in which a sudden increase in the number of users yields a drastic increase in the response time. We asked the participants to rank the selected control points identified in the previous step with regard to their impact of bringing the performance under control. In a controlled environment, each participant performed the pairwise comparisons followed by the direct rank. All participants performed the tasks individually without consulting on another. A sample of our questionnaire is attached in Appendix A (Empirical Questionnaire).

Figure 5.1 and Table 5.3 show a sample of the comparison matrix[11] and direct rank for a participant, Participant A. The aggregated result for all participants is shown in Table 5.4.

After comparing the participants' individual rank with the aggregated final rank, we noticed that the majority of the participants' ranks were different from the aggregated result. In section 5.2, we show that the group ranking agrees with the objective ranking, which resulted from a model-based simulation. This may suggest that ranking control points should be based on the group decision rather than the

[11]We populated the lower triangle comparison matrix as Not Applicable since they are reciprocal values.



Objective: Low response time	Add web servers	Increase No. of Threads	Reduce Network Latency	Change Instance type	Add more Bandwidth	Change Disk type
Add web servers	🟥	More preferred	More preferred	More preferred	More preferred	More preferred
Increase No. of Threads	Not Applicable	🟥	More preferred	Less preferred	More preferred	More preferred
Reduce Network Latency	Not Applicable	Not Applicable	🟥	Less preferred	Equally preferred	Less preferred
Change Instance type	Not Applicable	Not Applicable	Not Applicable	🟥	More preferred	More preferred
Add more Bandwidth	Not Applicable	Not Applicable	Not Applicable	Not Applicable	🟥	Less preferred
Change Disk type	Not Applicable	Not Applicable	Not Applicable	Not Applicable	Not Applicable	🟥

Figure 5.1: Pairwise Comparisons for Low Response Time

individual designer decision.

5.2 Experiment 1: Validate Ranking Method

In this experiment, we set to compare the ranks obtained from the designers, Designer Ranking (see Table 5.4), with an objective ranking obtained through model-based simulation (we call this ranking Experimental Ranking). In the next sections, we continue our discussion on acquiring ranks from the model-based simulation in more depth.

Control Points	**Rank No.**
Add more Bandwidth	3
Change Instance type	1
Change Disk type	4
Reduce Network Latency	5
Increase No. of Threads	2
Add web servers	1

Table 5.3: Direct Rank for Participant A

Control Points (CPs)	**Rank No.**	**%**
Add web servers	1	89%
Change Instance type	2	67%
Increase No. of Threads	3	56%
Change Disk type	4	45%
Reduce Network Latency	5	34%
Add more Bandwidth	6	34%

Table 5.4: Designer Ranking- Final Rank Result

5.2.1 Experimental Setup

To obtain ranks from a model-based simulation, we considered the same scenario given to the participants, but this time we effectively applied actions on the 6 control points (as per Table 5.4) on the 1000 simulated deployments in cloud. Then we measured the effect of the control points on the response time.

For simulation, we used a performance tool, the Optimization, Performance Evaluation and Resource Allocator (OPERA)[12] tool. The OPERA is a layered queueing model used to evaluate the performance of web applications deployed on arbitrary infrastructures. With OPERA, one can model the application's architecture and performance characteristics, perform operations on control points and estimate response time, throughput, and utilization of resources (i.e., CPU and disk). The OPERA tool has been described in more detail in [28].

To model the uncertainty of the Online Shop performance parameters, we consider a set of 1000 different performance parameter sets (CPU Demands, Disk Demands, number of calls, etc.). This is equivalent to 1000 different implementations or deployments of the Online Shop. Table 5.5 depicts the minimum and maximum range specified for the parameters of the Online Shop.

[12]`http://www.ceraslabs.com/technologies/opera`

Parameter	**Range**
bytesReceived	[15 KB, 1.5 MB]
bytesSent	[10 KB,1 MB]
CPUDemand	[1,90] ms
DiskDemand	[1, 30] ms
Latency	[10,500] ms
Workload	[100,10,000] requests
Threads	[Workload/2] ms
Bandwidth	[100 Mb, 1 Gb] per s
Think Time	[100,10,000] ms

Table 5.5: Minimum and Maximum Range for Application Parameters

All of the values specified in Table 5.5 represent typical networks and application deployments.

- bytesSent and bytesReceived refer to the number of bytes sent and received during a call (e.g., web browser to web server).

- CPUDemand and DiskDemand refer to the time (in milliseconds) required at the CPU/Disk for one request to be processed.

- Latency refers to the network latency between the servers distributed across different availability zones in the cloud.

- Workload refers to the number of requests (users).

- Think Time refers to the idle time of the users between two requests in milliseconds.

- Threads refers to the number of threads running in web server.

- Finally, Bandwidth refers to the time (in milliseconds) required to transmit a bit.

Therefore, each application contained a specific Workload along with initial settings for Latency, Bandwidth, Number of Threads, CPU and DISK Demands, and bytesSent and bytesReceived as indicated in Table 5.5.

5.2.2 Experiments

Table 5.6 shows how we made the changes in the control points to bring the performance back to the 500 ms. The **Step** indicates the increment and decrement used for changing the control points. For each experiment, we changed the control points one at a time until one of the stopping conditions is met:

Name	**Step**
Number of web servers	1
Instance size	1
Disk size	1
Latency	10
Bandwidth	x2
Number of Threads	10

Table 5.6: Steps used for changing control points.

- No improvement in n successive iterations with a difference of less than threshold (In our experiments, n=3, and threshold=10 ms), or
- Reached the resource boundary (as per Table 5.1), or
- Reached the adaptation goal, response time (500 ms).

5.2.2.1 An Example Illustrated

This section demonstrates how we applied each control point to estimate its effect on response time. Table 5.7 displays parameters' specifications for a model (Model M, hereafter).

Parameter	**Value**
bytesReceived	29.676 KB
bytesSent	19.784 KB
Web server CPUDemand	19 ms
Web server DiskDemand	6 ms
Database server CPUDemand	12 ms
Database server DiskDemand	6 ms
Latency	138 ms
Workload	327 requests
Threads	163
Bandwidth	315 Mb/s

Table 5.7: Initial Settings for Model M's Parameters

First, we run the model with OPERA to attain the performance result. Table 5.8 highlights the performance metrics obtained from running the simulation.

As indicated in Table 5.8 , the response time is approximated at 5 s, the CPU is utilized at 99% and 66% for web server and database server respectively. Looking at the performance metrics, we anticipate Add web servers and Change Instance type would be the most effective control points in bringing down the response time since

Response Time	5098 ms (approx. 5 s)
Web server CPU Utilization	99%
Database server CPU Utilization	66%
Web server/Database server Disk Utilization	31%

Table 5.8: Performance Metrics for Model M

our application is CPU saturated. We will verify this in the next sections.

The following sections show how we implemented each control point on Model M.

The Effect of Adding Web Servers on Response Time

Considering Model M, the initial response time is greater than the adaptation goal (response time less than 500 ms). To achieve the adaptation goal, we start adding one web server at a time and capture the performance metrics so as to evaluate the impact of adding an additional web server on response time. As mentioned in Section 5.2.2, we compare the three consecutive response times while adding a web sever to the application environment. The process is stopped when any of the stopping conditions is met.

Figure 5.2 indicates the effect of adding additional web servers on response time for Model M. The x-axis shows the number of web servers, and the y-axis shows the corresponding response time in seconds. It is apparent from Figure 5.2 that adding

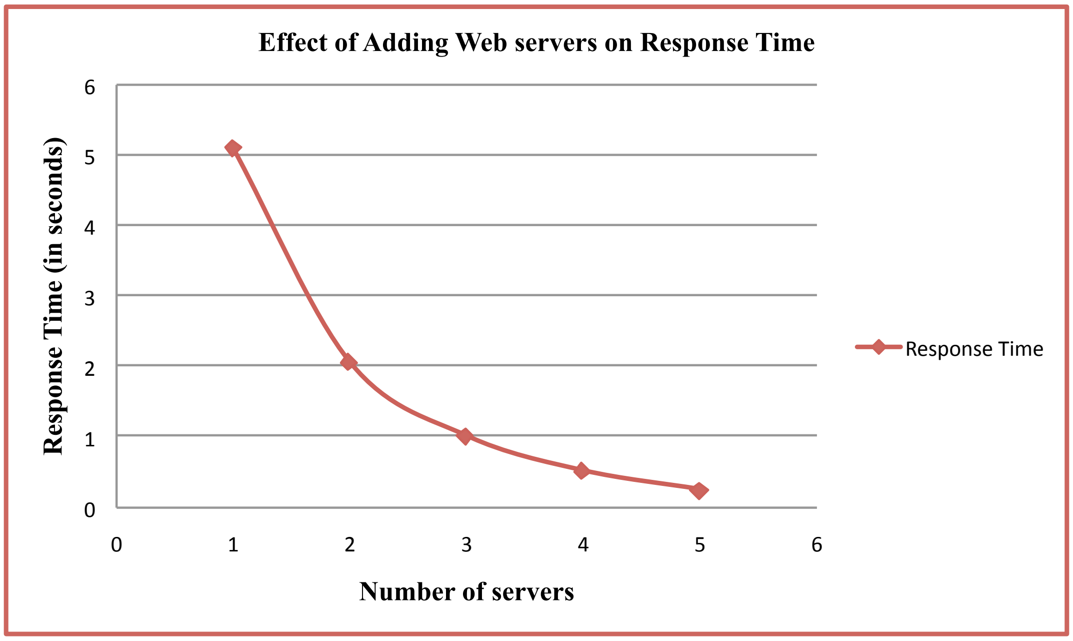


Figure 5.2: Impact of Adding Web Servers on Performance

an additional server, changes the response time significantly from 5 s to 2 s. We keep adding additional web severs and comparing the difference of response times. When the 5^{th} web server is added, the response time reaches 0.2 s, and the process stops since the adaptation goal is achieved.

The Effect of Changing CPU type on Response Time

Next, we apply another control point, Change Instance type, to assess its impact on response time. We go through the same process as discussed above for adding web servers and we capture the response time. Recalling our resource boundary, we can have only 4 instance types. At each iteration, we change the parameter of the instance one unit at a time. Figure 5.3 depicts the effect of changing instance type on response time.

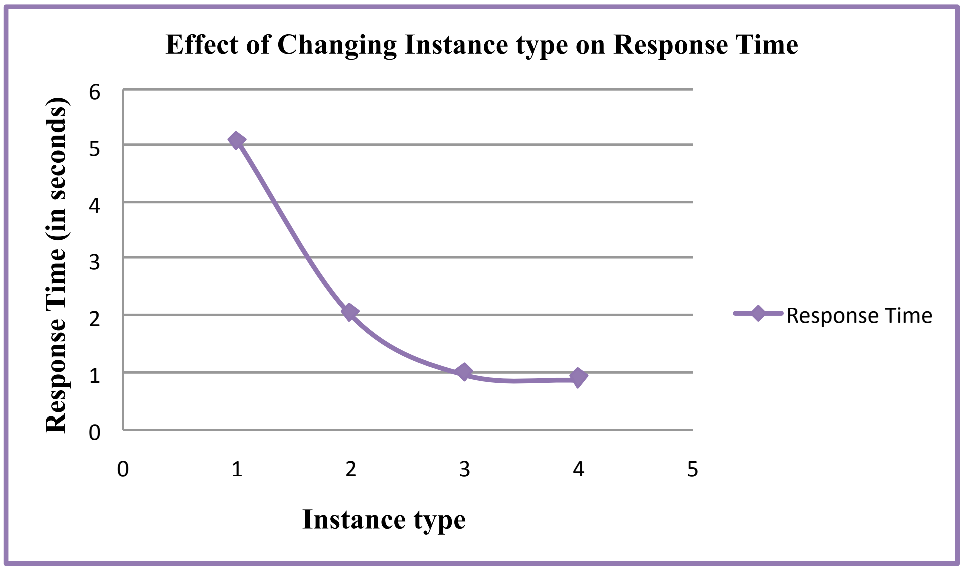


Figure 5.3: Impact of Changing Instance type on Performance

The points on the x-axis indicate the different types of an instance. For instance, 1 indicates the small size, 2 indicates the medium size, and 3 and 4 indicate the large and x-large instance sizes respectively. The y-axis shows the response time in seconds.

The process starts with changing a small instance to a medium instance. There is a significant change in response time when a small instance is changed to a medium one at the application tier. As shown in Figure 5.3, changing the instance type from small to medium brings the response time down from 5 s to 2 s. At the next iterations, the instance type is changed to a large and then to a x-large one (see point 3 and 4 on the x-axis). As depicted in Figure 5.3, after the large instance type (point 3 on the x-axis), modifying the control point does not have a big impact on response time. The process is then stopped since the resource boundary is reached. The response time is reduced to 0.9 s even though the adaptation goal is not yet met.

Recalling the performance metrics for Model M, the bottleneck was on CPU. Hence, increasing instance size brings down the response time significantly. The web server CPU is now 74% utilized (compared to the initial 99% utilized).

The Effect of Changing Disk type on Response Time

Here, we change the size of disk in web server cluster to assess its effect on response time. Like instance type, we have the same resource boundary about the different size of disks. Therefore, each model cannot have more than four types. Our result shows no significant improvement in response time as it remains around 5 s. The process iterates three times and stops since it meets the stopping condition (reaching resource boundary). This is because the disk was initially at the 31% utilization; thus, increasing disk size does not improve the response time.

We previously assumed that changing the disk type will not have a significant impact on response time since the application's bottleneck was not at the disk. Thus, this shows our assumption was right.

The Effect of Adding Threads on Response Time

Now, we show the impact of increasing the number of threads in web server cluster on response time. We add threads (10 at a time) to the web server cluster until it meets one of the stopping conditions. The initial number of threads is 163. Adding 10 threads to the web server cluster does not have an impact as the response time remains around 5 s. We continue increasing the number of threads. However, adding additional threads does not improve the response time since the response time is estimated at 4 s after 16 iterations. Therefore, the process stops since there is not a big improvement in response time when adding additional threads.

The Effect of Reducing Network Latency on Response Time

We then apply another control point, Reduce Network Latency, to evaluate its impact on response time. We decrease the network latency between the web server and the database server by 10 ms at a time to see the effect of change in response time.

Like adding number of threads, reducing network latency, does not have significant effect on response time. The initial latency is 138 ms. After three iterations, the network latency is decreased to 108 ms, and the response time is still around 5 s.

Our result shows that removing network latency brings up the response time slightly. Looking at the performance metrics, the web server CPU utilization is slightly higher than the initial values when the model was first solved by the OPERA. This indicates that reducing the network latency is not an effective adaptation for the Model M. Since the CPU is already saturated, network adaptation does not help the application to improve its response time. Hence, the process iterates three times and stops since there is no significant improvement in response time.

The Effect of Acquiring More Bandwidth on Response Time

Finally, we increase bandwidth between the web server and the database server in order to examine its effect on response time. When adding more bandwidth, we double the value at each iteration until the resource boundary (1Gb/s) is reached.

Our result shows that adding more bandwidth also does not have an impact on response time since the CPU utilization of servers becomes slightly higher. After three runs, the response time is still around 5 s. Therefore, adding more bandwidth saturates the CPU more. Since the bottleneck for Model M is on CPU, the control points aimed at reducing the CPU utilization such as adding web servers and changing instance type should be taken into account. Thus, adding more bandwidth is not an effective adaptation for the model M.

We used the same procedure described for Model M in applying each control point across all generated applications. Next section describes how we aggregated the result to conclude the final rank.

5.2.3 Aggregate Model-based Ranks

After obtaining the data, we counted the number of times each control point ranked 1st, 2nd, 3rd, etc.The rank of the control point is the most agreed rank across all experiments. Table 5.9 shows the result.

Control Points	**Rank #**	%	σ
Add web servers	1	99%	2.0%
Change Instance type	2	99%	3.5%
Increase No. of Threads	3	74%	0.70%
Change Disk type	4	75%	<0.01%
Reduce Network Latency	5	57%	0.05%
Add more Bandwidth	6	42%	0.17%

Table 5.9: Experimental Ranking- Final Rank Result

5.2.4 Discussion

As shown in Table 5.9, we found out that in 99% of time, the Add web servers control point was the highest rank. The second most effective control point to improve the response time across all applications was Change Instance type. Then Increase No. of Threads, Change Disk type, Reduce Network Latency, and Add more Bandwidth ranked 3$^{\text{rd}}$, 4$^{\text{th}}$, 5$^{\text{th}}$, and 6$^{\text{th}}$ respectively. We observed in several experiments that reducing network latency and adding more bandwidth did not have a significant impact on improving the response time. They also brought the response time higher. This could be due to the fact that the CPU/Disk was already saturated, therefore, network adaptation did not help the application to decrease its response time. Hence, we did not see a significant improvement on response time when they were applied.

Tables 5.4 and 5.9 reveal that both Experimental Ranking and Designer Ranking resulted in the same rank order. Our result confirms that our proposed Designer Ranking method was capable of producing a fair ranking.

5.3 Experiment 2: Evaluation of Adaptation Strategies

This experiment evaluates whether the ranking of control points is important in the adaptation algorithm and if the Designer Ranking performs well. Similar to the previous experiment, we adapted 1000 application deployments to meet the response time of less than 500 ms using MAPE-K adaptation loop with different order of control points (The order of the control points is used by the Algorithm 1). We used OPERA to measure the effect of adaptations on response time. Besides the Designer Ranking, we also use 2 other rankings, which are described in sections 5.3.1 and 5.3.2. The experimental result is discussed in 5.3.3.

5.3.1 Random Ranking

When two opposite ranking of control points are used (e.g., Experimental Ranking and the inverse of Experimental Ranking), the impact of ranking on the adaptation algorithm in order to meet the adaptation goal is more clear. Table 5.10 displays the inverse of Experimental Ranking, called Random Ranking.

5.3.2 Leverage Points Ranking

To develop an additional ranking of control points, we used the rank proposed for socio-economic complex systems by Donella Meadows. Meadows identifies 12 points (known as leverage points) from the least effective to the most effective, which can be seen as different ways to change a system [31]. We mapped Meadows' Leverage Points to our control points. Table 5.11 displays the mapping result.

Control Points	**Rank No.**
Add more Bandwidth	1
Reduce Network Latency	2
Change Disk type	3
Increase No. of Threads	4
Change Instance type	5
Add web servers	6

Table 5.10: Random Ranking

We mapped the "Increase No. of Threads" and "Add web servers" control points to "Power to add, change, evolve, or self-organize system structure" leverage point (point #4 in Meadows list). Adding new threads creates information flows, and adding web servers change the system structure by creating new information flow through the system. Hence, they both correspond to number 4 in leverage points list.

We mapped "Reduce Network Latency" control point to "Length of delays" leverage point (point #9) since it directly alters the delay. "Adding more Bandwidth" control point is also related to delay as it is the speed through which the information

Original Rank label is the result of mapping control points to leverage points ranks, and LP Rank label is the ranks used in our experiments.

Control Points	**Original Rank**	**LP Rank**
Add web servers	4	1
Increase No. of Threads	4	2
Reduce Network Latency	9	3
Add more Bandwidth	9	4
Change Instance type	12	5
Change Disk type	12	6

Table 5.11: Leverage Points Ranking

is delivered.

Both "Change Instance type" and "Change Disk type" control points are changed by modifying the parameter of the VM and the storage respectively. Therefore, we mapped them to the "Constants, parameters, numbers" leverage point (point# 12).

5.3.3 Experiments

In the experiments, for the 1000 random deployments, we applied the feedback loop adaptation algorithm using the three rankings of control points (i.e., Designer Ranking, Leverage Points Ranking, and Random Ranking) separately. The application deviation from the performance goal (500ms) was random for each experiment. To stress the adaptation, the deviation was larger than normally would happen in practice. In Table 5.12, we summarize the applications' response time prior to adaptation process across all experiments.

Min. Response time	693 ms
Max Response time	13 minutes
AVG. Response time	3 minutes
Standard Deviation	181100 ms

Table 5.12: Distribution of Response Time Deviation

For each random deployment, we ran our adaptation algorithm and captured the response time and the number of times a control point was iterated in order to meet the adaptation goal. Next, we walk through an example in order to demonstrate how we apply our algorithm on different rankings of control points (we call them adaptation strategies, hereafter).

5.3.3.1 An Example Illustrated

Let's consider a model, M1. We first solve the model with OPERA and capture the performance metrics as depicted in Table 5.13.

Response Time (s)	CPU utilization	Disk Utilization
304 (approx. 5.08 minutes)	99%	66%

Table 5.13: Performance Metrics for M1

In the following sections, taking the model M1, we apply our algorithm on each adaptation strategy, and discuss our findings.

Designer Ranking

Figure 5.4 shows that M1 reaches the predefined adaptation goal with 56 iterations in total. There is a continuous reduction in response time as web servers are added to and instance size is increased in M1. However, there is no improvement in response time when more threads are added. This is shown in the graph as a constant line (see points 52 to 55 in Figure 5.4). Increasing disk size then decreases the response time further, and finally M1 reaches the adaptation goal.

Random Ranking

Now, we apply our algorithm on Random Ranking strategy. Since the order of rank is the inverse of Designer Ranking, this strategy is assumed to have the worst effect on response time in comparison with Designer Ranking. Hence, we presume that on average this strategy should end up with more number of iterations in total

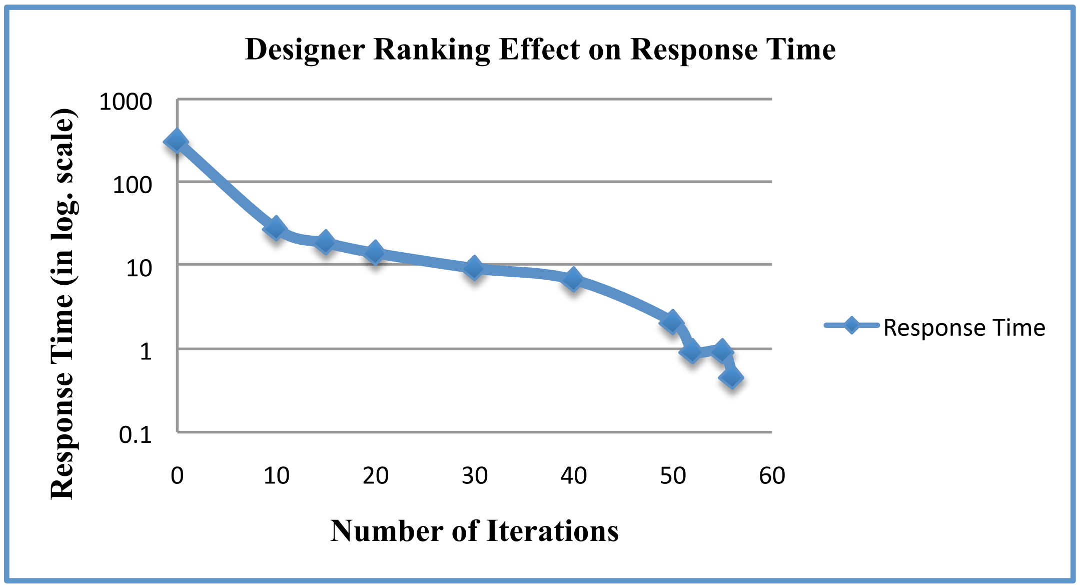


Figure 5.4: The Effect of Designer Ranking on Response Time

to achieve the adaptation goal.

Figure 5.5 displays the result of Random Ranking adaptation strategy on model M1. Our result conforms our assumptions. As seen in Figure 5.5, the strategy starts with adding more bandwidth. Since the CPU is already saturated, adding more bandwidth does not help M1 decrease its response time. Moreover, reducing network latency, adding more threads, and increasing disk size do not also bring the response time down. This is shown as a constant line in Figure 5.5. Improvement is only seen after increasing the instance size and adding web servers to M1. The steeping slope in Figure 5.5 indicates that increasing the instance size and adding additional web servers (see points 56 to 107 in the graph) significantly decrease the

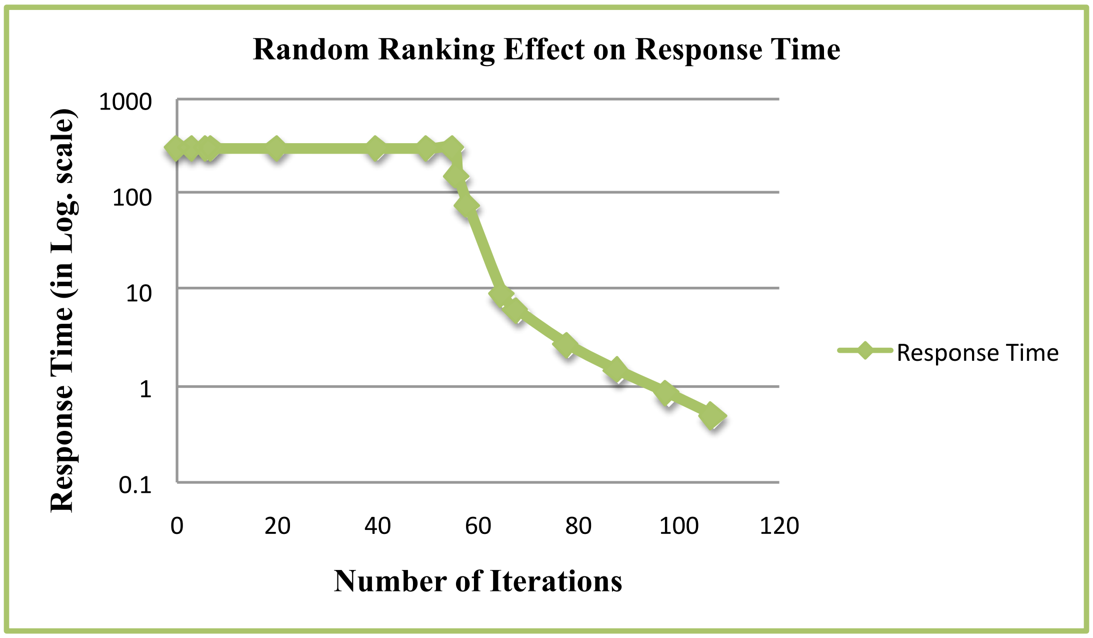


Figure 5.5: The Effect of Random Ranking on Response Time

response time. Therefore, the model ends up with the total of 107 iterations to achieve the adaptation goal.

Leverage Points Ranking

Finally, we apply our algorithm on Leverage Points Ranking strategy. Figure 5.6 shows our result.

The effect of control points on response time is apparent in Figure 5.6. Reduction of response time from 5 min to 5 s is the result of adding web servers, threads, bandwidth, and reducing network latency in M1. A sudden decrease in response

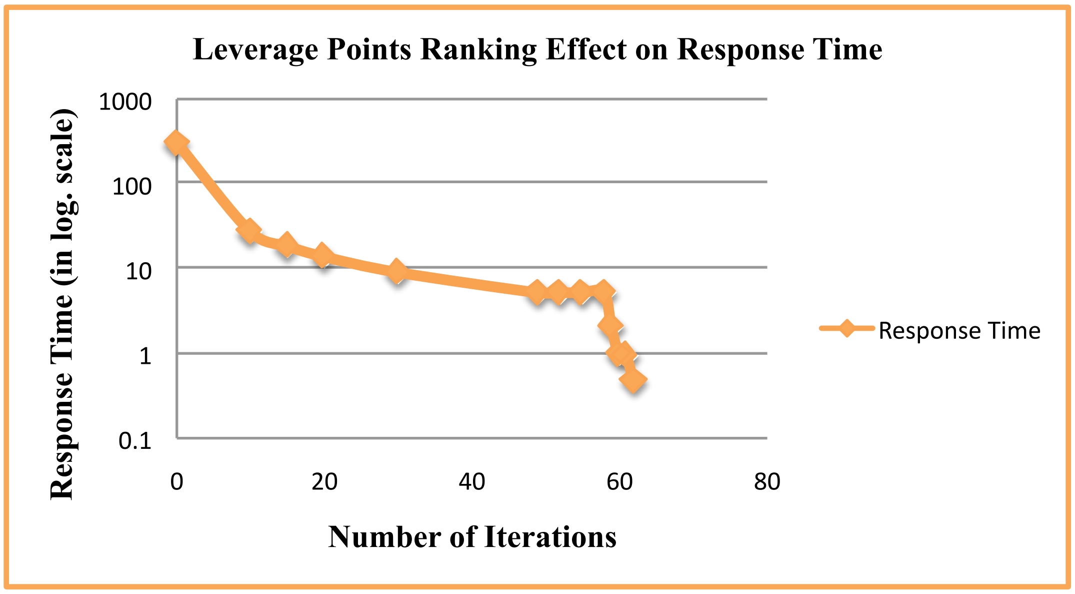


Figure 5.6: The Effect of Leverage Points Strategy on Response Time

time from 5 s to 0.9 s is due to increasing the instance size to x-large size (see points 59 to 61 in Figure 5.6). M1 is then achieved the adaptation goal by increasing the disk size to medium size. As a result, M1 reaches the adaptation goal by 62 iterations in total.

Discussion

Our result demonstrates that all strategies were able to reach the predefined adaptation goal. We made the total number of iterations as a measure to indicate how fast each strategy reaches the adaptation goal. Table 5.14 depicts the number of iterations across all strategies for M1.

Adaptation Strategy	**Total Iteration No.**	**Reached Adaptation Goal?**
Designer Ranking	56	Yes
Random Ranking	107	Yes
Leverage Points Ranking	62	Yes

Table 5.14: Number of Iterations for strategies

As shown in Table 5.14, the Designer Ranking strategy went through 56 iterations. As we assumed, the Random Ranking ended up with the most number of iterations (107) to reach the adaptation goal. Leverage Points Ranking strategy also resulted close to the Designer Ranking with the total of 62 iterations. We can conclude that Designer Ranking is the most preferred runtime strategy for model M1.

5.3.4 Experiment Result

This section presents the summary of the collected data across 1000 random deployments. The average number of iterations required for the Designer Ranking, Leverage Points Ranking, and Random Ranking strategies to reach the adaptation goal is shown in Figure 5.7. The result shows that the Designer Ranking required fewer iterations to achieve the adaptation goal. Also, the Designer Ranking ended up with smaller standard deviation. However, the difference between the average number of iterations is not significant enough to firmly conclude whether one ranking is better than the others.

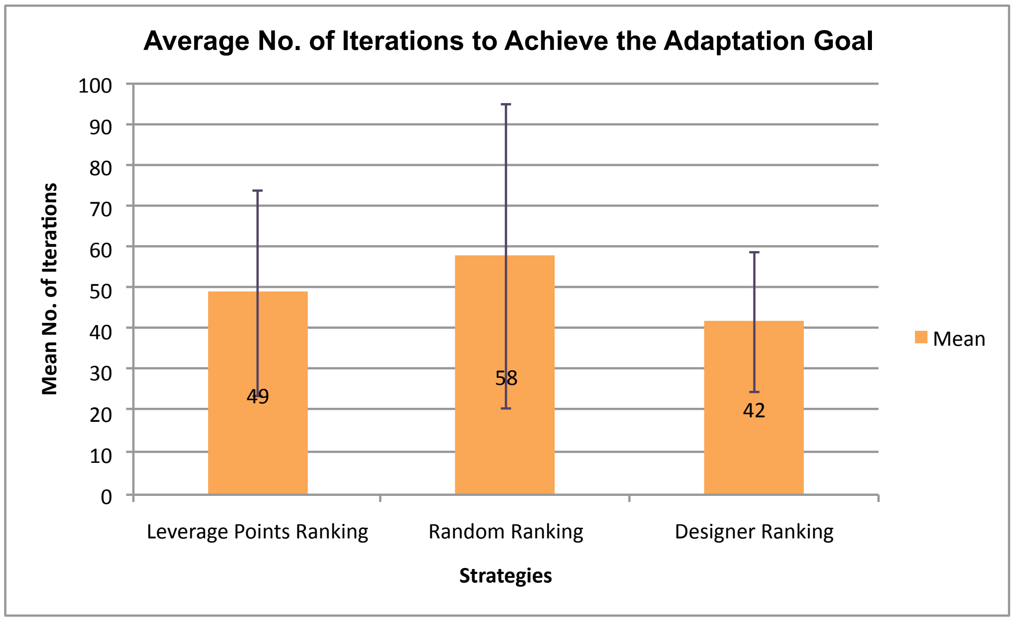


Figure 5.7: Average number of iterations required to reach the adaptation goal

To assess the performance of Designer Ranking against Leverage Point Ranking and Random Ranking, we calculated the frequency of the required iterations to meet the adaptation goal. Figure 5.8 shows the distribution of the number of iterations required to achieve the adaptation goal. From Figure 5.8, we noticed that all rankings yield approximately a similar number of failures (approximately 30%) to reach the response time of less than 500 ms. We can make two observations: (1) All rankings

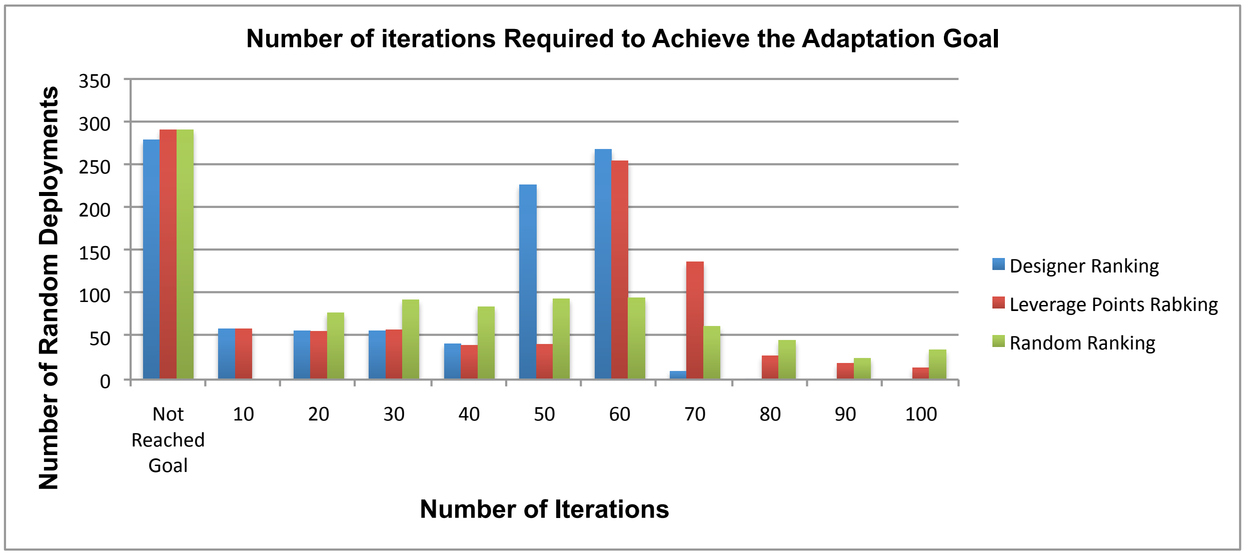


Figure 5.8: Histogram of iterations required to reach the adaptation goal

eventually reached the adaptation goal, given enough time; and (2) The rankings affect the speed of achieving the adaptation goal.

Figure 5.9 depicts the observation (2) above. It shows the relationship between the number of iterations and average response time across all models. The y-axis shows the average normalized response time, and the x-axis shows the number of iterations. As shown in Figure 5.9, the Designer Ranking and Leverage Points Ranking have a steep slope and they overlap, meaning they have a similar effect on response time. On the other hand, the slope of the Random Ranking strategy is not as steep as the Designer/Leverage Points Ranking strategies, and as a result, it does not converge fast enough to reach the adaptation goal. This is due to the effect of the

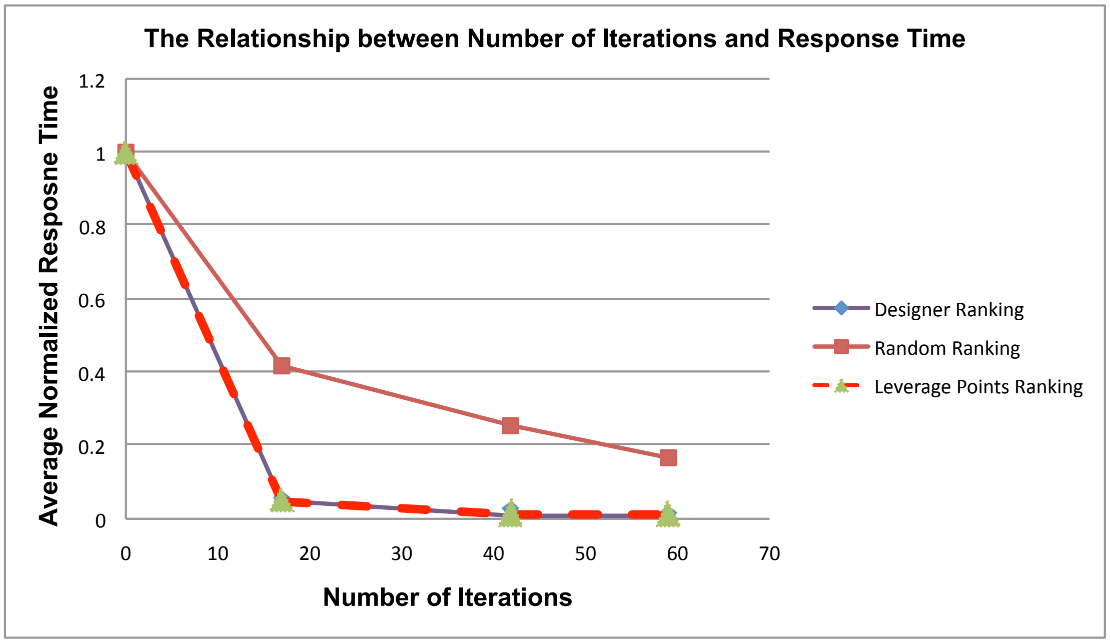


Figure 5.9: The effect of ranking on the speed of achieving the adaptation goal

order of control points. The Random Ranking starts with adding more bandwidth, thus, we do not see an effective impact on lowering the response time. Inversely, the Designer Ranking and Leverage Points Ranking strategies converge faster to reach the adaptation goal as the result off adding servers as their first control point.

5.4 Summary

In this chapter, we showed the capability of our methodology in eliciting and ranking control points. We conducted a series of experiments with the OPERA tool to validate our ranking method. Our result confirmed that Designer Ranking resulted in the same rank order as the OPERA tool suggested. In order to evaluate the Designer Ranking strategy, we applied our adaptation strategy algorithm on 1000 random deployments using Designer Ranking, Random Ranking, and Leverage Points Ranking strategies. We used OPERA to measure the effect of each strategy on the adaptation goal, and captured the number of iterations and response time. Our experiments concluded that given enough time, all strategies were able to reach the adaptation goal, however, the order of ranking affected the speed to meet the adaptation goal. Our result showed that the Designer Ranking strategy was the most preferred runtime strategy, which required fewer number of iterations on average to meet the adaptation goal.

6 CONCLUSIONS AND FUTURE WORK

In this thesis, we proposed a methodology to design an adaptive cloud-based web application. Our proposed methodology allows a mapping of high level adaptation goal to low level MAPE-K loop adaptive controls via elicited control points. In order to assess the capability of our methodology, we conducted user studies and a series of experiments to rank control points and to evaluate the efficiency of our adaptation algorithm for an online shopping cart application. Our collected data proves that we were able to attain reasonable adaptation results using our methodology. We summarize our contributions as follow:

- The main contribution of the thesis is the introduction of control points as the first class adaptation elements. We showed how to build control point models for clouds, multi-clouds, and applications to facilitate the elicitation process. We also showed our catalogues of control points that can be reused to elicit control points.

- We also explained how we utilized pair-wise comparisons and direct ranking to acquire rank for control points with respect to the adaptation goal.
- We then built an adaptation strategy based on MAPE-k loop. The proposed algorithm uses the most effective control points to achieve the adaptation goal. That is, when there is a deviation from an original goal, we execute commands that affect the control points. We start with the highest ranked control point and execute the commands as long as there is an improvement in the adaptation goal's metric. Then we select the next control point, etc.

We made the following observations with our experimental result:

- The Designer Ranking is compatible with what the model-based simulation suggested.
- The Designer Ranking strategy outperforms alternative approaches as it resulted in reaching the adaptation goal faster compared to the other strategies used in our experiments.
- Using different rank order of control points in our adaptation algorithm affected the speed of achieving the adaptation goal. This signifies the importance of eliciting, selecting, and executing the appropriate control points in order of importance.

- Using a controlled experiment, the group ranking of control points yields better control strategies than those identified by individuals.

We conducted our experiments using a simulation rather than running a real application on the cloud. The main reason we used simulation was to test our methodology on a large number of applications, which was not practical with real life applications. We simulated 1000 different deployments and workloads, but a bursting scenario was not simulated. We used OPERA, which is an accepted model for simulating applications running in cloud environment. Since we got satisfactory results, in the future, we are planning to apply this methodology to manage real life applications.

In the future, we would also like to extend our methodology to include multiple adaptation goals, and as a result ranking multiple feedback loops and implementing them in an prioritized order. We also plan for further experiments with other types of applications and adaptation algorithms.

Bibliography

[1] Jesper Andersson, Rogerio De Lemos, Sam Malek, and Danny Weyns. Modeling dimensions of self-adaptive software systems. In *Software engineering for self-adaptive systems*, pages 27–47. Springer, 2009.

[2] Luciano Baresi and Liliana Pasquale. Live goals for adaptive service compositions. In *Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, pages 114–123. ACM, 2010.

[3] Nelly Bencomo. On the use of software models during software execution. In *Proceedings of the 2009 ICSE workshop on modeling in software engineering*, pages 62–67. IEEE Computer Society, 2009.

[4] Nevon Brake, James R. Cordy, Marin Litoiu, et al. Automating discovery of software tuning parameters. In *Proceedings of the 2008 international workshop on Software engineering for adaptive and self-managing systems*, pages 65–72. ACM, 2008.

[5] Yuriy Brun, Ron Desmarais, Kurt Geihs, Marin Litoiu, Antonia Lopes, Mary Shaw, and Michael Smit. A design space for self-adaptive systems. In *Software Engineering for Self-Adaptive Systems II*, pages 33–50. Springer, 2013.

[6] Yuriy Brun, Giovanna Di Marzo Serugendo, Cristina Gacek, Holger Giese, Holger Kienle, Marin Litoiu, Hausi Müller, Mauro Pezzè, and Mary Shaw. Engineering self-adaptive systems through feedback loops. In *Software Engineering for Self-Adaptive Systems*, pages 48–70. Springer, 2009.

[7] Jim Buckley, Tom Mens, Matthias Zenger, Awais Rashid, and Günter Kniesel. Towards a taxonomy of software change. *Journal of Software Maintenance and Evolution: Research and Practice*, 17(5):309–332, 2005.

[8] Rajkumar Buyya, James Broberg, and Andrzej M. Goscinski. *Cloud computing: Principles and paradigms*, volume 87. John Wiley & Sons, 2011.

[9] Betty H.C. Cheng and Joanne M. Atlee. Research directions in requirements engineering. In *2007 Future of Software Engineering*, pages 285–303. IEEE Computer Society, 2007.

[10] Betty H.C. Cheng, Rogerio De Lemos, Holger Giese, Paola Inverardi, Jeff Magee, Jesper Andersson, Basil Becker, Nelly Bencomo, Yuriy Brun, Bojan Cukic, et al.

Software engineering for self-adaptive systems: A research roadmap. In *Software engineering for self-adaptive systems*, pages 1–26. Springer, 2009.

[11] Lawrence Chung, Brian Nixon, Eric Yu, and John Mylopoulos. Non-functional requirements. *Software Engineering*, 2000.

[12] Autonomic Computing et al. An architectural blueprint for autonomic computing. *IBM White Paper*, 2006.

[13] Luiz Marcio Cysneiros. Evaluating the effectiveness of using catalogues to elicit non-functional requirements. In *WER*, pages 107–115, 2007.

[14] Luiz Marcio Cysneiros, Vera Maria Werneck, and Andre Kushniruk. Reusable knowledge for satisficing usability requirements. In *Requirements Engineering, 2005. Proceedings. 13th IEEE International Conference on*, pages 463–464. IEEE, 2005.

[15] Luiz Marcio Cysneiros and Eric Yu. Non-functional requirements elicitation. In *Perspectives on software requirements*, pages 115–138. Springer, 2004.

[16] Anne Dardenne, Axel Van Lamsweerde, and Stephen Fickas. Goal-directed requirements acquisition. *Science of computer programming*, 20(1):3–50, 1993.

[17] Armando Fox, Rean Griffith, Anthony D. Joseph, Randy H. Katz, Andrew Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, and Ion Stoica. Above the clouds: A berkeley view of cloud computing. *Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS*, 28:13, 2009.

[18] Alan G. Ganek and Thomas A. Corbi. The dawning of the autonomic computing era. *IBM systems Journal*, 42(1):5–18, 2003.

[19] David Garlan, Shang-Wen Cheng, An-Cheng Huang, Bradley Schmerl, and Peter Steenkiste. Rainbow: Architecture-based self-adaptation with reusable infrastructure. *Computer*, 37(10):46–54, 2004.

[20] John C. Georgas and Richard N. Taylor. Towards a knowledge-based approach to architectural adaptation management. In *Proceedings of the 1st ACM SIGSOFT workshop on Self-managed systems*, pages 59–63. ACM, 2004.

[21] Hamoun Ghanbari and Marin Litoiu. Identifying implicitly declared self-tuning behavior through dynamic analysis. In *Software Engineering for Adaptive and Self-Managing Systems, 2009. SEAMS'09. ICSE Workshop on*, pages 48–57. IEEE, 2009.

[22] Markus C. Huebscher and Julie A. McCann. A survey of autonomic computingdegrees, models, and applications. *ACM Computing Surveys (CSUR)*, 40(3):7, 2008.

[23] Honghong Jiang and Xiaohu Yang. Performance requirement elicitation for financial information system based on ontology. In *TENCON 2009-2009 IEEE Region 10 Conference*, pages 1–5. IEEE, 2009.

[24] Jeffrey O. Kephart and David M. Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[25] Jeffrey O. Kephart and William 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*, pages 3–12. IEEE, 2004.

[26] Jeff Kramer and Jeff Magee. Self-managed systems: an architectural challenge. In *Future of Software Engineering, 2007. FOSE'07*, pages 259–268. IEEE, 2007.

[27] Alexei Lapouchnian, Yijun Yu, Sotirios Liaskos, and John Mylopoulos. Requirements-driven design of autonomic application software. In *Proceedings of the 2006 conference of the Center for Advanced Studies on Collaborative research*, page 7. IBM Corp., 2006.

[28] Marin Litoiu and Cornel Barna. A performance evaluation framework for web applications. *Journal of Software: Evolution and Process*, 25(8):871–890, 2013.

[29] Marin Litoiu, Murray Woodside, and Tao Zheng. Hierarchical model-based autonomic control of software systems. In *ACM SIGSOFT Software Engineering Notes*, volume 30, pages 1–7. ACM, 2005.

[30] Philip K. McKinley, Seyed Masoud Sadjadi, Eric P. Kasten, and Betty H.C. Cheng. Composing adaptive software. *Computer*, 37(7):56–64, July 2004.

[31] Donella Meadows. Places to intervene in a system. *Whole Earth*, 91:78–84, 1997.

[32] Daniel Menasce, Hassan Gomaa, Sam Malek, and Joao P. Sousa. Sassy: A framework for self-architecting service-oriented systems. *Software, IEEE*, 28(6):78–85, 2011.

[33] John Mylopoulos, Lawrence Chung, and Brian Nixon. Representing and using nonfunctional requirements: A process-oriented approach. *Software Engineering, IEEE Transactions on*, 18(6):483–497, 1992.

[34] Peyman Oreizy, Dennis Heimbigner, Gregory Johnson, Michael M. Gorlick, Richard N. Taylor, Alexander L. Wolf, Nenad Medvidovic, David S. Rosen-

blum, and Alex Quilici. An architecture-based approach to self-adaptive software. *IEEE Intelligent systems*, 14(3):54–62, 1999.

[35] Nauman A. Qureshi and Anna Perini. Continuous adaptive requirements engineering: An architecture for self-adaptive service-based applications. In *Requirements@ Run. Time (RE@ RunTime), 2010 First International Workshop on*, pages 17–24. IEEE, 2010.

[36] Nauman A. Qureshi, Anna Perini, Neil A. Ernst, and John Mylopoulos. Towards a continuous requirements engineering framework for self-adaptive systems. In *Requirements@ Run. Time (RE@ RunTime), 2010 First International Workshop on*, pages 9–16. IEEE, 2010.

[37] Thomas L. Saaty. *The Analytic Hierarchy Process*. Mcgraw-Hill International, 1980.

[38] Mazeiar Salehie and Ladan Tahvildari. A weighted voting mechanism for action selection problem in self-adaptive software. In *Self-Adaptive and Self-Organizing Systems, 2007. SASO'07. First International Conference on*, pages 328–331. IEEE, 2007.

[39] Mazeiar Salehie and Ladan Tahvildari. Self-adaptive software: Landscape and research challenges. *ACM Transactions on Autonomous and Adaptive Systems*

(TAAS), 4(2):14, 2009.

[40] Patrizia Scandurra, Claudia Raibulet, Pasqualina Potena, Raffaela Mirandola, and Rafael Capilla. A layered coordination framework for optimizing resource allocation in adapting cloud-based applications. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, pages 471–472. ACM, 2012.

[41] Vítor E Silva Souza, Alexei Lapouchnian, William N. Robinson, and John Mylopoulos. Awareness requirements for adaptive systems. In *Proceedings of the 6th international symposium on Software engineering for adaptive and self-managing systems*, pages 60–69. ACM, 2011.

[42] Michael Smit, Mark Shtern, Bradley Simmons, and Marin Litoiu. Partitioning applications for hybrid and federated clouds. In *Proceedings of the 2012 Conference of the Center for Advanced Studies on Collaborative Research*, pages 27–41. IBM Corp., 2012.

[43] Vıtor E Silva Souza, Alexei Lapouchnian, and John Mylopoulos. (requirement) evolution requirements for adaptive systems. In *Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2012 ICSE Workshop on*, pages 155–164. IEEE, 2012.

[44] Saeed Ullah, Muzaffar Iqbal, and Aamir Mehmood Khan. A survey on issues in non-functional requirements elicitation. In *Computer Networks and Information Technology (ICCNIT), 2011 International Conference on*, pages 333–340. IEEE, 2011.

[45] Jeffrey S. Vetter and Daniel A. Reed. Real-time performance monitoring, adaptive control, and interactive steering of computational grids. *International Journal of High Performance Computing Applications*, 14(4):357–366, 2000.

[46] Eric S.K. Yu. Towards modelling and reasoning support for early-phase requirements engineering. In *Requirements Engineering, 1997., Proceedings of the Third IEEE International Symposium on*, pages 226–235. IEEE, 1997.

[47] Eric S.K. Yu and John Mylopoulos. Understanding why in software process modelling, analysis, and design. In *Proceedings of the 16th international conference on Software engineering*, pages 159–168. IEEE Computer Society Press, 1994.

[48] Parisa Zoghi, Mark Shtern, and Marin Litoiu. Designing search based adaptive systems: A quantitative approach. In *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '14. ACM, 2014.

Appendix A

Experiment

Consider a shopping cart application with a typical multi-tire architecture as shown in Figure 1. The application architecture also includes a load balancer between the presentation and application tiers to disperse the workload among various web servers.

The presentation tier provides users with user interface through web browser enabling them to browse HTML pages, search for items, and make a purchase. The application tier carries out the transaction, and visits the database to retrieve information. The database tier is responsible for storing information. The *Shopping Cart* application uses request-reply protocols as communication type between the server and the client. The client sends a request to the server, which in return the server replies back to the client. The communication is synchronous, which means that the client upon request is waiting for a reply from the server.

Scenario: Imagine a sudden increase in the number of users yields a drastic increase in the response time. Rank the alternative adaptations identified on the next page with respect to their impact on bringing the response time under the control.

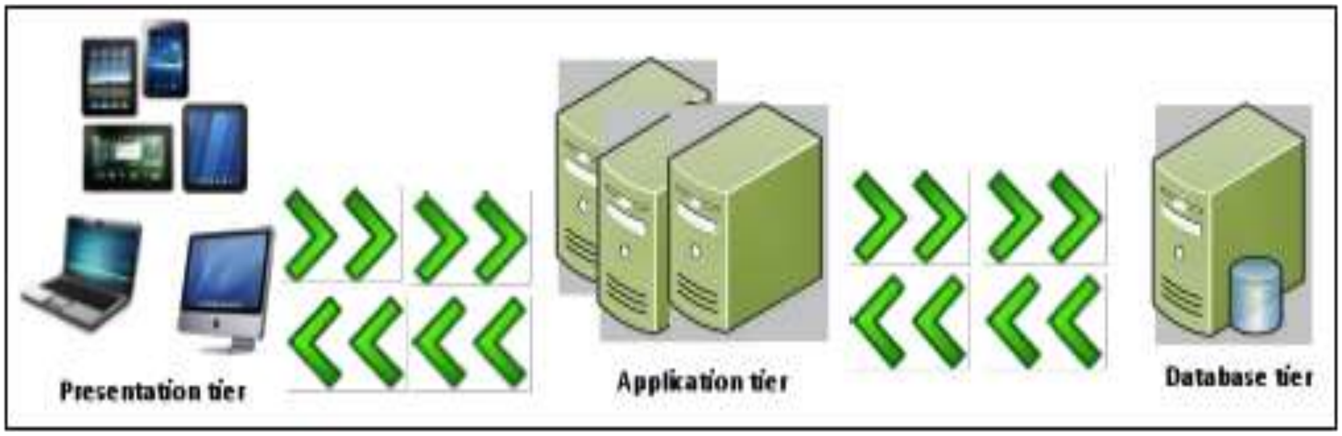


Figure 1. Conceptual architecture.

List of Adaptations:

The adaptations below are designed to bring down the shopping cart application's response time to the desired level (e.g., within 1 sec.).

NOTE: We assume that all adaptations will improve the response time.

Control Points	**Definition**
Add more Bandwidth	To add more bandwidth between web server and database server.
Change Instance type	To increase instance size for web server cluster.
Change Disk type	To increase disk size for web server cluster.
Reduce Network Latency	To decrease network latency between the web server and database server.
Increase No. of Threads	To increase number of threads in the web server cluster.
Add web servers	To add additional web servers to the application environment.

Instruction:

You will be provided with a criterion (i.e., low response time) to rank the alternative adaptations for the Shopping Cart application by performing two tasks: (1) Pair-wise comparison and (2) Direct ranking

In order to perform pair-wise comparison, task (1), you should give your preferences for each pair using the intensity preference measurement indicated in the table below.

Intensity Preference	**Definition**
Less preferred to	An alternative has lower impact on response time than another one.
Equally preferred to	Two alternatives are equally important with respect to the criteria.
More preferred to	An alternative has higher impact on the response time than another one.

Table 2. Intensity Preference Measurement

Example:

The example below demonstrates the pair-wise comparisons in choosing a cell phone. One might consider particular features such as screen size and memory when attempting to buy a cell phone.

To determine which cell phone is the best-suited candidate; one can express his preference by comparing each feature. As shown below, the decision maker indicates screen size is less preferred to memory in choosing a cell phone.

GOAL: Choosing a cell phone	**Intensity Preference**	**Feature**	**Definition**
Screen size	Less preferred to	Memory	Screen size is less favored over memory in choosing a cell phone.
Screen size	Equally preferred to	Physical Keyboard	Screen size and keyboard are equally important in choosing a cell phone.
Memory	More preferred to	Physical Keyboard	Memory is more preferred over physical keyboard in choosing a cell phone.

Table 3. Example of a pair-wise comparison, Choosing a cell phone

Task (1):

Pair-wise Comparison Table:

Please perform the pairwise comparisons by populating the upper triangle of the below table. Remember the goal is to bring down the response time to 1 sec.

Please specify your preference from the drop down menu, Intensity Preference, by clicking on each cell on the table.

Objective: Low response time	**Add web servers**	**Increase No. of Threads**	**Reduce Network Latency**	**Change Instance type**	**Add more Bandwidth**	**Change Disk type**
Add web servers	🟥					
Increase No. of Threads	Not Applicable	🟥				
Reduce Network Latency	Not Applicable	Not Applicable	🟥			
Change Instance type	Not Applicable	Not Applicable	Not Applicable	🟥		
Add more Bandwidth	Not Applicable	Not Applicable	Not Applicable	Not Applicable	🟥	
Change Disk type	Not Applicable	Not Applicable	Not Applicable	Not Applicable	Not Applicable	🟥

Table 4. Comparison Matrix.

Task (2):

<u>Direct Ranking:</u>

On the previous task, you performed pair-wise comparisons between adaptations based on achieving low response time (e.g., within 1 second). We would like to know what is the best ranking in a general situation for the Shopping Cart application to maintain its goal.

Please rank the following adaptations listed in Table 5 in order of importance. **Please indicate your preference by numbering from 1-6, where 1 is the most important adaptation and 6 is the least important adaptation. You can use a number more than one to indicate your indifference among adaptations.**

Adaptations	**Rank#**
Change Disk type	
Change Instance type	
Increase No.of Threads	
Add web servers	
Move web server closer to database server	
Add more bandwidth	

Table 5. Direct Rank Table.