Designing Adaptive Applications Deployed on Cloud Environments

PARISA ZOGHI, MARK SHTERN, MARIN LITOIU, and HAMOUN GHANBARI,
York University

Designing an adaptive system to meet its quality constraints in the face of environmental uncertainties can be a challenging task. In a cloud environment, a designer has to consider and evaluate different control points, that is, those variables that affect the quality of the software system. This article presents a methodology for designing adaptive systems in cloud environments. The proposed methodology consists of several phases that take high-level stakeholders' adaptation goals and transform them into lower-level MAPE-K loop control points. The MAPE-K loops are then activated at runtime using search-based algorithms. Our methodology includes the elicitation, ranking, and evaluation of control points, all meant to enable a runtime search-based adaptation. We conducted several experiments to evaluate the different phases of our methodology and to validate the runtime adaptation efficiency.

Categories and Subject Descriptors: D.2.8 [**Software Engineering**]: Metrics—*Complexity measures, performance measures*

General Terms: Theory, Design

Additional Key Words and Phrases: Adaptive systems, cloud computing, design, performance

ACM Reference Format:
Parisa Zoghi, Mark Shtern, Marin Litoiu, and Hamoun Ghanbari. 2016. Designing adaptive applications deployed on cloud environments. ACM Trans. Auton. Adapt. Syst. 10, 4, Article 25 (January 2016), 26 pages.
DOI: http://dx.doi.org/10.1145/2822896

1. INTRODUCTION

Web applications deployed in 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 nonfunctional 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 at least 20 minutes. Evidently, achieving NFRs in the dynamic cloud environments in a cost-effective manner remains a largely unresolved problem due to the changing nature of the operational environment, complex requirements, unexpected failures, and so forth.

[1]In the cloud, performance goals are achieved through scalability. We look at 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/.

This research was supported by the SAVI Strategic Research Network (Smart Applications on Virtual Infrastructure), funded by The Natural Sciences and Engineering Research Council of Canada (NSERC) and by the Connected Vehicles and Smart Transportation (CVST) funded Ontario Research Fund.
Authors' addresses: P. Zoghi, M. Shtern, M. Litoiu, and H. Ghanbari, York University, Toronto, ON, Canada; emails: pzoghi@ceraslabs.com, mshtern@yorku.ca, mlitoiu@yorku.ca, hamoun@cse.yorku.ca.

Adaptive design such as autonomic computing [Kephart and Chess 2003; Kramer and Magee 2007; Ganek and Corbi 2003] can help achieve the NFRs. This article focuses on designing cloud-based web applications by creating mechanisms for achieving the adaptation goals. We formalize the problem as a multi-criteria optimization and identify the variables that enable the optimization.

This article relates and builds on previous work [Zoghi et al. 2014], proposing a quantitative methodology for designing adaptive web-based applications to meet quality requirements. It consists of elicitation and ranking of adaptation operations. Adaptation operations are also known as control points [Vetter and Reed 2000]. We define control points as those artifacts in a system that are modified at runtime and cause controlled changes in the system, thus affecting the quality of the services offered by the system. To facilitate the elicitation of control points, we propose control-point models that map control points to NFRs. Our approach has two main advantages. The first is the ranking of control points in the early design phase, which enables prioritization of control points and informed decision making about which ones are essential to meet the stakeholders' objectives. The second advantageis that conflict resolution (between competing NFRs) is solved through the use of feedback loops at runtime rather than arbitrarily at the elicitation phase. Finally, we conduct two case studies in a simulated environment to demonstrate the capability of our methodology in designing an adaptive application in the cloud.

We extend our previous work as follows: (a) We add new control points to the control point catalogue. Our extended control point catalogue includes control points for web applications and cloud. This catalogue contains a collection of control points that are important for designing adaptive web applications running in the cloud. (b) We conduct a case study with conflicting adaptation goals, such as low response time and low cost. This case study demonstrates that the proposed methodology can be used for designing adaptive applications with conflicting goals. (c) We compare our search-based algorithm, derived from methodology, with a known mathematical optimization algorithm (SQP). The results show that our proposed search-based algorithm is efficient. This means that our search-based algorithm is capable of bringing an application to a desired state effectively, and automatically trade-offs conflicts between adaptation goals.

The remainder of this article is organized as follows. In Section 2, we present our research motivation. Section 3 discusses our methodology. In Section 4, we show how to use our methodology through a case study. We conduct a case study on performance and cost, and compare our result with a mathematical optimization algorithm in Section 5. We discuss related work and validity concerns in Sections 6 and 7, respectively. We conclude the paper in Section 8.

2. MOTIVATION

Autonomic computing has been suggested as a general guideline for designing adaptive systems [Huebscher and McCann 2008]. Survey papers such as Cheng et al. [2009] and Salehie and Tahvildari [2009] discuss the challenges of designing and engineering such systems.

Brun et al. [2009] emphasize the importance of feedback loops in designing adaptive systems. The authors consider feedback loops as first-class entities since they are the essential features in controlling and managing uncertainties in software systems. They assert that, without visible feedback loops, the impact of these feedback loops on overall system behavior could not be identified, hence the most important properties of self-adaptation would fail to be addressed.

Designing an adaptive system entails decisions such as how to monitor the system's environment, how to select and activate adaptations, and so on. Currently, this is done in an ad-hoc manner as pointed it out in Brun et al. [2013]. They 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 system by answering a set of questions along with five dimensions: identification, observation, representation, control, and adaptation mechanisms. For each dimension, they identify key 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 cloud-computing environments. Moreover, they all have applied adaptations in their research work, such as add/remove servers. Some have applied change instance–type adaptations, and a few have applied other types 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 the adaptation goals[5] for a multitier online shopping cart application running in the cloud as a low response time and low cloud resource cost. The participants were required to identify 10 control points, elicit feedback loops for them, and rank these 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 an adequate intuition 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.

Therefore, this study motivated us to develop a methodology to assist the designers with identifying and ranking control points, and eliciting feedback loops to achieve the objectives.

3. METHODOLOGY

In this section, we present our approach for creating adaptation loops for web-based applications. Our target is web applications that are similar to Znn.com[6] in high-level architecture [Garlan et al. 2004]. The software system can have multiple adaptation goals, which could conflict. For instance, adding more resources improves the performance goal, but negatively affects the cost goal.

In this work, we assume the adaptation goals (such as minimize cost and response time below a threshold) have been already identified by the application's stakeholders. Thus, our task is to design and implement the feedback (or adaptation) loops that achieve those goals.

Our proposed methodology encompasses four steps:

(1) *Define the Objective Function.* We define an objective function in order to indicate whether a system is in alignment with its adaptation goals. We discuss our objective function in Section 3.1.
(2) *Elicit Control Points.* We introduce control-point models, which act as aids in the elicitation process. We elicit control points in the system that can affect one or more

[4]http://www.ceraslabs.com.
[5]NFR goals that are achieved through adapting the software and infrastructure.
[6]http://www.hpi.uni-potsdam.de/giese/public/selfadapt/exemplars/.

adaptation goals. During the elicitation phase, the elicitor does not address conflicts between the contradictory adaptation goals because our search-based algorithm resolves those conflicts at runtime. Our elicitation process is defined in Section 3.2.

(3) *Rank Control Points*. We rank the elicited control points. In order to rank, we propose a process using pairwise comparison and direct ranking, as discussed in Section 3.3.

(4) *Implement a Search-Based Feedback Loop Controller*. We use the elicited control points to build a feedback loop. The feedback loop will implement the search-based algorithm to adapt the system to meet the stakeholders' high-level objectives. It will determine the control points to act on at any point in time in order to achieve the adaptation goals. Section 3.4 discuses our search-based algorithm.

3.1. Define the Objective Function

Assume that the stakeholders identified k adaptation goals (G_i). We can formulate the achievement of the adaptation goals as an optimization problem: a minimization function (see Equation (1)), a boundary function (see Equation (2)), or, in the case of multiple goals, a combination of both (see Equation (3)). In Equations (1) and (2), m_i is an application metric (e.g., response time, throughput, utilization, cost), and F_i and φ_i are real number functions.

$$min(F_i(m_1, m_2, ..m_n)) \tag{1}$$

$$F_i(m_1, m_2, \ldots, m_n) < \varphi_i(m_1, m_2, \ldots, m_n). \tag{2}$$

For instance, minimizing the cost of an application running in the cloud is an adaptation goal, which can be expressed as Equation (1). On the the other hand, Equation (2) captures goals such as response time should be less than 1s.

Often, stakeholders assign different priorities to adaptation goals. Consequently, it is possible to define a weight (w_i) for each goal such as $\sum_{i=0}^{k} w_i = 1$. We can then combine all adaptation goals into one objective function, defined as follows:

$$min\left[\sum_{i=0}^{j} F_i w_i + \sum_{i=0}^{l}(w_i P_i(max(\varphi_i - F_i, 0))max(\varphi_i - F_i, 0))\right] \tag{3}$$

The P_i is the cost of missing the objectives (violating SLOs), which is defined by the stakeholders, and $k = j + l$, where j and l are number of minimization and boundary conditions, respectively.

Functions similar to Equation (3) have been used in other contexts, such as genetic algorithms optimization [Smith and Coit 1997; Michalewicz et al. 1996]. Note that the first term in Equation (3) refers to those goals that need to be minimized (Equation (1)) while the second term refers to the boundary goals. The sign function associated with the second term makes sure that the terms are 0 when the goals are exceeded.

The objective function indicates how well the overall goals are achieved; therefore, it should be minimized. The smaller value of the objective function means better alignment with the stakeholders' goals. The role of a runtime feedback loop is to minimize function (Equation (3)). The feedback loop can optimize Equation (3) using well-known algorithms, such as hill climbing [Russell and Norvig 1995].

Any change in control points affects the metrics $m_1, m_2, m_3, \ldots, m_n$; as a result, the value of Equation (3) will be changed. The smallest value of the Equation (3) indicates that the stakeholders' adaptation goals are either achieved or close to being fully achieved. Therefore, to achieve the adaptation goals, two problems need to be solved: first, how to discover the most influencing control points, and second, how to control them efficiently to make Equation (3) small. The first problem is addressed in Sections 3.2 and 3.3, and the second problem is discussed in Section 3.4.

We believe that stakeholders should be able to identify metrics using the framework presented in Villegas et al. [2011] for their adaptation goals. After identification of metrics, the stakeholders should be able to formalize the adaptation goals using Equations (1) and (2). This formalization helps us to convert an adaptation problem into an optimization problem. There are different ways to define objective functions and solve the optimization problems. We selected the objective function because of its efficiency when constraints are nonlinear [Ervin 1977].

3.2. Elicit Control Points

The process of eliciting control points resembles NFR elicitation. One such approach is the NFR Framework [Chung et al. 2000], in which higher-level goals are decomposed into subgoals using an AND/OR tree until they cannot be decomposed any further. In our approach, we propose control-point models to assist with identifying and eliciting control points. To elicit control points, one can build a control point model, as we discuss in Section 3.2.1 or reuse our catalogues (see Section 3.2.3). This section is allocated to control-point models, how to construct one, and their reusable functionality through catalogues.

3.2.1. Control-Point Model. Our proposed method draws from goal-modelling methodologies. Thus, 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 operations (leaf node) that is required for meeting the adaptation goal. This understanding gives the reason why operations are essential.
(3) The complex adaptation goal is decomposed into different subgoals.
(4) A subgoal is represented by a cloud.
(5) The operations are denoted by rectangles.
(6) (Optional) Adaptation goals or subgoals can have different metrics attached to them. (Metrics are denoted by diamonds).

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

One difference between our control-point model and a goal model is that we decompose each adaptation goal separately and elicit control points only with regard to that goal. Another difference is the conflict resolution process. A control point model is conflict-free since all operations/subgoals should contribute for meeting only the top-level adaptation goal. We are not concerned with any goal conflicts in this early stage of elicitation. Our control-point model addresses only the control points for one adaptation goal at a time, ignoring the effect of each control point on other adaptation goals (i.e., performance vs. cost). We resolve the conflicts at runtime in a feedback loop (see Section 3.4) by using the objective function that mitigates the conflicting goals (see Section 3.1). This built-in conflict resolution mechanism between different adaptation goals simplifies the elicitation of control points and makes it possible to model control points as a separate control-point model.

Applications deployed in the cloud also have the feature of having 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 follows:

—*Cloud*. What cloud capability can help to achieve an adaptation goal? 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, imagine the adaptation goal is less than 1s response time. Amazon Web Services (AWS) Elastic Beanstalk [7] is an Elastic Compute Cloud (EC2)[8] service that provides autoscaling capability for a web application. We can define a control point, which enables autoscaling using the AWS Elastic Beanstalk service.

—*Multi-cloud*. How will migration from one cloud to another one align the application with its adaptation goals? How does utilizing additional clouds help the application to meet the stakeholders' objectives? For instance, the adaptation goal is less than 1s response time and the application will be deployed in a private cloud. Utilization of a public cloud will enable massive scaling by bursting the application process into a public cloud. We can define a control point that will enable the offloading of nonsensitive service into a public cloud [Smit et al. 2012].

—*Application*. Which web application scenarios exhibit similar runtime behavior in relation to adaptation goal or subgoal? How can management systems control the runtime behavior of these scenarios? What configuration parameters (existing or defined) will control runtime behavior? For instance, let's assume that an adaptation goal is less than 1s 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.

3.2.2. Control-Point Model Construction. This section presents how to build a control-point model in order to elicit control points for an adaptation goal. Let us assume that the adaptation goal for an application is to minimize the runtime cost in a cloud. In order to construct a control-point model, the following steps are taken:

(1) We start the process with an adaptation goal as a high-level goal to be attained. Figure 1 shows the adaptation goal, Minimize Runtime Cost, for the application.
(2) We extract metric(s) that directly affect(s) the achievement of the adaptation goal, as shown in Figure 2. This will help us focus on operations that lower the runtime cost in a cloud.
(3) We start decomposing the high-level adaptation goal into subgoals by answering a series of questions such as how to increase the utilization of resources, how to reduce unnecessary resources, and how to substitute the required services with cheaper ones. Figure 3 depicts the decomposition of the adaptation goal by answering these questions by splitting them into three subgoals. For instance, multiple services that run on separate VMs can be redeployed on one VM in order to increase utilization. Thus, the adaptation goal can be decomposed into the increase utilization subgoal. We can also scale down the application by removing unnecessary/underutilized resources. Hence, we decompose the adaptation goal into the decommission of underutilized resources subgoal to save cost. Finally, we can find cheaper services/resources. For example, a micro instance in US East zone costs \$0.013 per hour, whereas the same instance costs \$0.027 per hour[9] elsewhere. Therefore, the adaptation goal can also be refined to the find cheaper resource/service subgoal.

[7]http://aws.amazon.com/elasticbeanstalk/.
[8]http://aws.amazon.com/ec2/.
[9]http://aws.amazon.com/ec2/pricing/.

Minimize Runtime Cost

Fig. 1. Application adaptation goal.

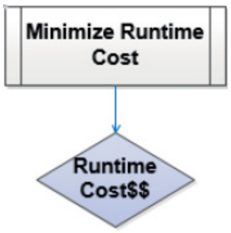


Fig. 2. Metric for the application adaptation goal.

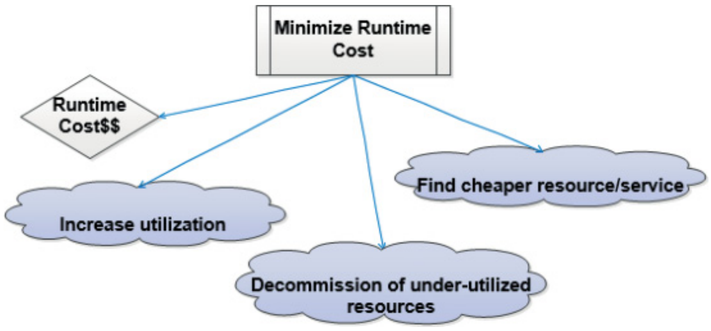

Minimize Runtime Cost
Runtime Cost$$
Increase utilization
Find cheaper resource/service
Decommission of under-utilized resources

Fig. 3. Sub-goals for Achieving the Adaptation Goal.

(4) We keep refining each subgoal until we can decompose them into operations, which are control points. For example, we start by refining the subgoal decommission of underutilized resources, shown in Figure 4, to the remove resources subgoal. We then decompose the remove resources subgoal to operations such as remove web server, remove database server, and remove load balance, which are our control points.
   In some cases, we do not need to refine subgoals. We can directly decompose a subgoal into operations. Figure 5 depicts how a subgoal is decomposed into an operation. Figure 6 shows the completed control-point model.

Figure 7 shows a control-point model for meeting an application's low response time, which we developed by applying our elicitation process described earlier.

3.2.3. Control-Point Catalogue. Creating and using catalogues to assist with elicitation have been used in the literature [Cysneiros and Yu 2004; Chung et al. 2000; Cardoso et al. 2009]. Inspired by these works, we define a catalogue as a repository of control points, which can be reused or extended further. In order to create a catalogue of control

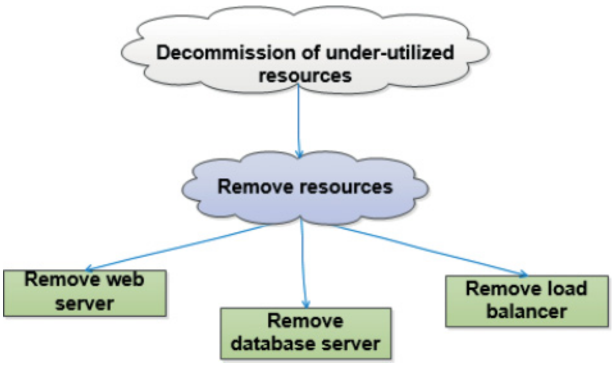


Fig. 4. (a) Decomposing the subgoals for the achieving adaptation goal.

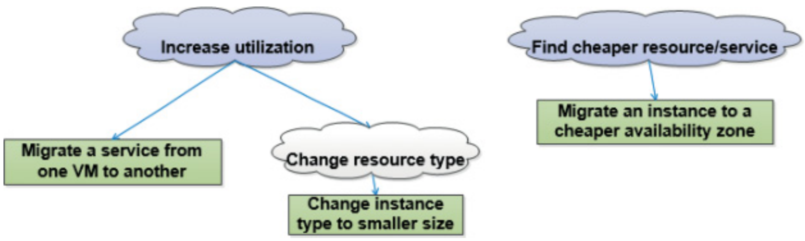

Increase utilization
Find cheaper resource/service
Migrate an instance to a cheaper availability zone
Migrate a service from one VM to another
Change resource type
Change instance type to smaller size

Fig. 5. (b) Decomposing the subgoals for achieving adaptation goal.

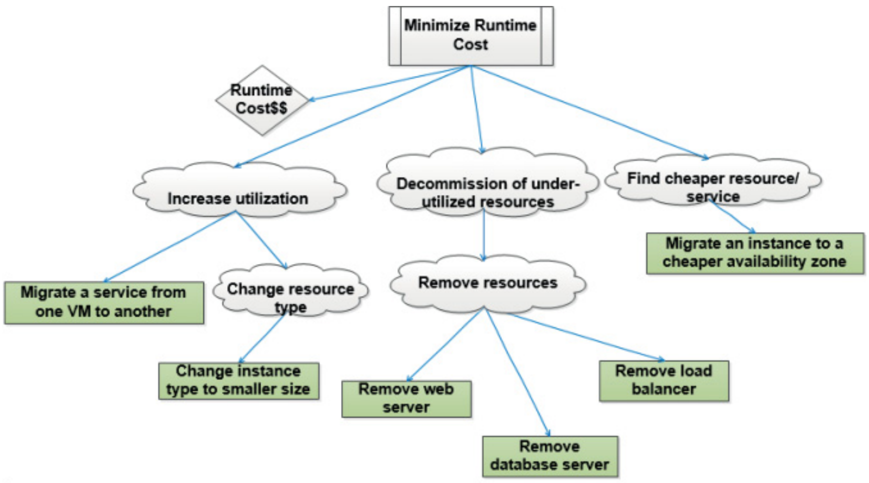

Minimize Runtime Cost
Runtime Cost$$
Increase utilization
Decommission of under-utilized resources
Find cheaper resource/ service
Migrate an instance to a cheaper availability zone
Migrate a service from one VM to another
Change resource type
Remove resources
Change instance type to smaller size
Remove web server
Remove load balancer
Remove database server

Fig. 6. Control-point model for runtime cloud cost.

points, we consider a list of NFRs that can be met through the adaptation process of an application, as depicted in Table I.

Table II displays a control-point catalogue with respect to different NFRs for applications and cloud environments. All control points listed in Table II are applicable to web servers, load balancers, and database servers.

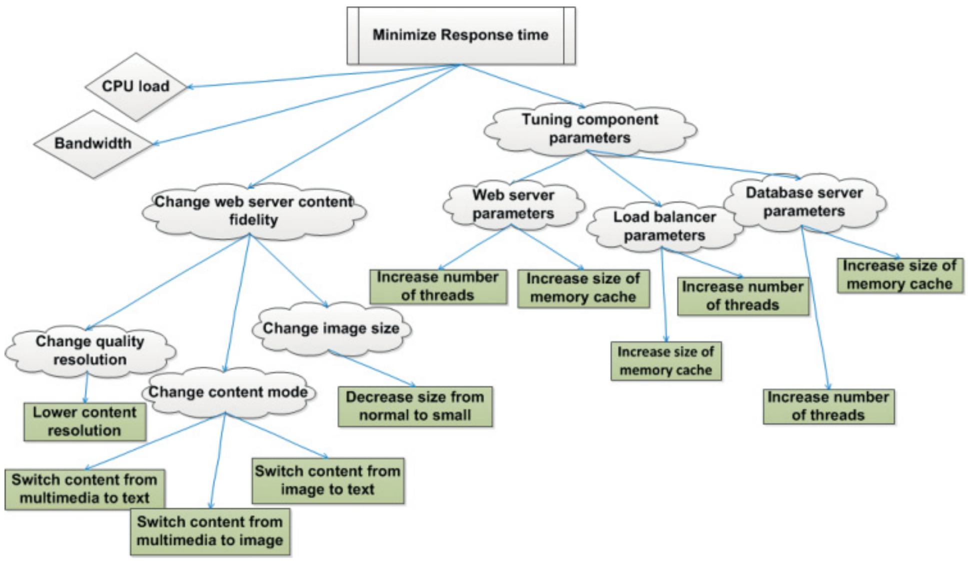

Minimize Response time
CPU load
Bandwidth
Tuning component parameters
Change web server content fidelity
Web server parameters
Load balancer parameters
Database server parameters
Increase number of threads
Increase size of memory cache
Increase number of threads
Increase size of memory cache
Increase size of memory cache
Increase number of threads
Change quality resolution
Change image size
Change content mode
Lower content resolution
Decrease size from normal to small
Switch content from multimedia to text
Switch content from image to text
Switch content from multimedia to image

Fig. 7. Control-point model for application response time.

Table I. NFRs and Their Corresponding Metrics

NFRs	Metrics
Performance	Response time, server utilization, throughput
Cost	Total number of machines, number of users
Security	User activity, credit card/IP velocity, suspicion level
Availability	Latency, abandon rate

3.3. Rank Control Points

This section explains 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 point with rank 1 means that 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) *Product pairwise comparisons.*
We use pairwise comparisons as in the Analytic Hierarchy Process (AHP) [Saaty 1980], which compares control points in pairs based on the higher-level goal. To conduct pairwise comparisons, the designers are asked to choose between three options: A>B, meaning that A is preferred to B; A<B, meaning that B is preferred to A; and A=B, meaning that 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] (see Figure 8) 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 ranking.*
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 each

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

Table II. Elicited Control Points for Adaptation Goals

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

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 a pairwise comparison prior to direct ranking is that the pairwise comparison helps designers to build a mental model about the influence of control points on adaptation goals. The consistency of pairwise comparison helps to measure the quality of the designers' mental model. After creating a good mental model by designers, they would do the direct ranking. The results of direct ranking obtained from designers are then aggregated.

(3) *Aggregate the obtained ranks.*
The final rank is obtained by aggregation of result, as follows:
   (a) Rank Control Points Regarding a Goal. Count the number of times each control point was ranked 1st, 2nd, and so on. The rank of the control point is the most agreed upon rank among all designers. The ranked control points are added to an ordered list, the CP-List. 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. Moreover, if two or more control points share the same rank and the same agreement among the designers, a random decision can be made.
   (b) Global Rank of Control Points. Knowing the weight of adaptation goals, w_i, and the rank for each control point alternative, R_{ij} (calculated in Step (a)), we calculate the final rank, R_j, by the following formula:

$$R_j = \sum_{i=0}^{k} w_i R_{ij}, \text{ where } k \text{ is the number of goals.}$$

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

3.4. Implementation of a Search-Based Feedback Loop Controller

Our adaptation strategy is implemented by a MAPE-k loop. The MAPE-k loop monitors application metrics associated with goals, and analyzes them. The MAPE-k loop actions are triggered periodically or when a substantial discrepancy is detected between the desired goals' metrics and the measured ones. The controller implements an algorithm that solves the optimization problem (see Equation (3)). There can be many ways to optimize the function (Equation (3)). Our optimization process has two phases. The first phase is to select the most important unsatisfied adaptation goal from the list, which was not already selected in previous X times. The second phase is to meet the selected adaptation goal by running the search-based adaptation algorithm (see Algorithm 1), which implements the hill-climbing strategy. The optimization process is repeated until the stakeholders' high-level objectives are met.

Now, we explain the details of the adaptation algorithm in the second phase. The adaptation algorithm (Algorithm 1) reduces the gap between the measured and desired goal's metrics 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 list of elicited control points, and `CP-Sorted-List` is the ordered list of control points aiming to achieve the adaptation goal. The previous and current values of objective function are stored in `m0` and `m1`, respectively.

ALGORITHM 1: Search-Based Algorithm

```
Select the control points contributing to goals G from CP-List and store them according to their
rank into CP-Sorted-List.
repeat
    is-any-improvement = false
    foreach CP from CP-Sorted-List do
        repeat
            Store the value of objective function (See Formula 3) into m0
            if CP can be executed AND Goal is NOT met then
                execute CP operation
                Store the value of objective function into m1
                if m0 - m1 > predefine threshold then
                    is-improved = true
                    is-any-improvement = true
                else
                    Rollback operation CP
                    is-improved = false
                end
            else
                is-improved = false
            end
        until is-improved;
    end
until is-any-improved;
```

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

Table III. Elicited Control Points and Their Definition

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 more web servers to the application environment

goal is satisfied or no further improvements of Equation (3) can be found[11]. Next, the adaptation algorithm selects the next control point and repeats the cycle. The algorithm stops if the input goal is satisfied or none of the control points can improve the objective function.

Note that the main distinction between our presented algorithm and a greedy algorithm is that our algorithm does not select the most effective control point in each step. Instead, it selects control points based on users' rank and applies the selected control point repeatedly until that control point can no longer be applied. Then, the next ranked control point will be selected and the process will be repeated.

4. CASE STUDY 1: ADAPTIVE SHOPPING CART FOR THE CLOUD

This section presents a case study, the design of an adaptive shopping cart web application for the cloud. Consider a shopping cart system, Online Shop, with a typical multitier 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 500ms as the application's adaptation goal.

In this case study, we evaluate our methodology by following all steps described in Section 3. We focus on two important issues: the validation of control point ranking and the validation of the search algorithm. In Section 4.2, we show how we validate the control point ranking. In Section 4.3, we discuss the efficiency of the search-based adaptation process using the ranking to meet the response time of less than 500ms.

4.1. Control Points for Online Shop

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

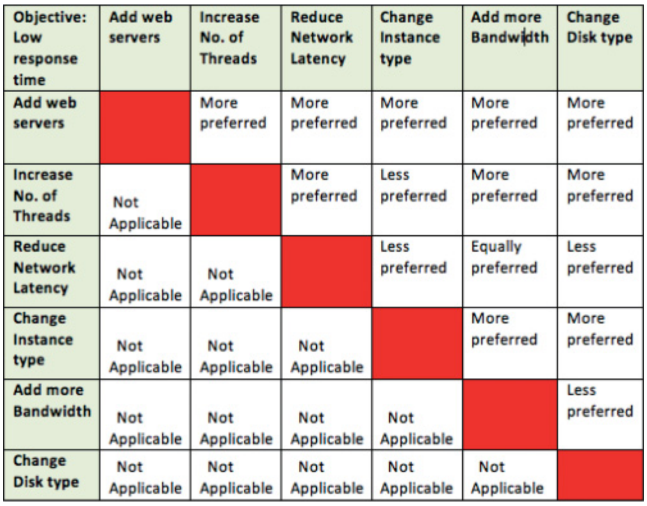
4.1.1. Define the Objective Function. The stakeholders' adaptation goal can be represented as the boundary function, such as $R < 500ms$, where R is the measured response time of the application. According to our presented methodology (see Section 3.1), the objective function should be defined as follows:

$$O = max(R - 500ms, 0)$$

4.1.2. Elicit Control Points. We selected control points from the catalogue for the cloud environment (Table II), and using the control points model shown in Figure 7. The 6 control points presented in Table III were selected for this study.

4.1.3. 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

[11]No further improvements of objective function means that the modification of selected control points has no further effect on the goal or it affects negatively other goals (e.g., adding servers improves performance but negatively affects the cost).



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	🟥

Fig. 8. Pairwise comparisons for low response time.

Table IV. Direct Rank for Participant A

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 V. Designer Ranking—Final Rank Result

Control Points	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%

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 response time. We asked the participants to rank the selected 6 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 one another. Figure 8 and Table IV show a sample of the comparison matrix[12] and direct rank for a participant, Participant A. Participant A prefers ``Add web servers'' more to ``increase No. of Threads'' as shown in Figure 8. Table IV shows that the same participant used rank number 1 for both add web servers and change instance type to show their equal preference. The aggregated result for all participants is shown in Table V.

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

Table VI. Control Points Limitations

Name	Min	Max	Step
No. of servers	1	50	1
No. of CPUs	1	4	1
No. of disks	1	4	1
Latency	10ms	500ms	10
Bandwidth	100Mb/ms	1 Gb/ms	x2
No. of threads	Request no./2	Request no.	10

Note: Request number varies from 100 to 10,000 requests.

After comparing the participants' individual rank with the aggregated final rank, we noticed that the majority of participants' ranks were different from the aggregated result. In Section 4.2, we show that the group ranking agrees with 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 individual designer decision.

4.2. Experiment 1: Validate Ranking Method

In this experiment, we set to compare the ranks obtained from the designers, which we call Designer Ranking (see Table V), with an objective ranking obtained through model-based simulation (we call this Experimental Ranking). 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, and so on). This is equivalent to 1000 different implementations or deployments of the Online Shop.

4.2.1. Model-Based Ranks. 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 V) on the 1000 simulated deployments in the 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)[13] 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 Litoiu and Barna [2013].

Table VI shows how we made the changes in the control points to bring the performance back to 500ms.

The Min and Max columns in Table VI reflect the minimum and maximum boundaries for control points.The Step column indicates the increment and decrement used for changing control points. For each experiment, we changed the control points one at a time until one of the stopping conditions is met: no improvement in n=3 successive iterations with a difference of less than threshold=10ms, or reached the control points limits, or reached the adaptation goal, response time less than 500ms. After conducting several experiments, we selected the stopping condition parameters, such as number of successful iterations and threshold such that a model may reach its adaptation goal in a reasonable number of iterations.

After obtaining the data, we counted the number of times each control point ranked 1st, 2nd, and so on. The rank of the control point is the most agreed upon rank across all experiments. Table VII shows the result.

[13]http://www.ceraslabs.com/technologies/opera.

Table VII. Experimental Ranking—Final Rank Result

Control Points	Rank No.	%	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 VIII. Random Ranking

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 IX. Leverage Points Ranking

Control Points	LP	EP
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

Tables V and VII 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.

4.3. Experiment 2: Evaluation of Search-Based Controller

This experiment evaluates whether the ranking of control points is important in the search-based feedback loop and if Designer Ranking performs well. Similar to the previous experiment, we adapted 1000 application deployments to meet the response time of less than 500ms using a search-based feedback loop with a different order of control points (the order of the control points is used by 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 4.3.1 and 4.3.2. The experimental result is discussed in Section 4.3.3.

4.3.1. Random Ranking. We expect that the impact of ranking is clear when we use two opposite rankings of control points: Experimental/Designer Ranking and the inverse of Experimental Ranking. Table VIII displays the inverse of Experimental Ranking, called Random Ranking.

4.3.2. Leverage Points Ranking. To develop an additional ranking of control points, we used the rank proposed for socioeconomically complex systems by Donella Meadows. Meadows identifies 12 points (known as leverage points) from the least effective to most effective, which can be seen as different ways to change a system [Meadows 1997]. We mapped Meadows's Leverage Points to our control points. Table IX displays

Table X. Distribution of Response Time Deviation

Minimum response time	693ms
Maximum response time	13min
Average response time	3min
Standard deviation	3min

the mapping result. LP label is the result of mapping control points to leverage points ranks, and EP label is the ranks we used in our experiments, as shown in Table IX.

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's list). Adding new threads creates information flows and adding new web servers changes the system structure by creating new information flow through the system. Hence, they both correspond to #4 in the leverage points list.

We mapped the "reduce Network Latency" to "length of delays" leverage points (point #9) since it directly alters the delay. Adding more bandwidth is also related to delay as it is the speed through which the information is delivered. Both "change instance type" and "change disk type" are changed by modifying the parameter of the VM and storage, respectively. Therefore, we mapped them to the "constants, parameters, numbers" leverage point (point #12).

4.3.3. Experiment Result. In the experiments, for the 1000 random deployments, we applied the search-based 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 X, we summarize the applications' response time prior to adaptation process, across all experiments.

For each random deployment, we ran the optimization algorithm and captured the response time and the number of times a control point was iterated in order to meet the adaptation goal[14]. The summary of the collected data is presented later.

The average number of iterations required for the Designer Ranking, Leverage Points Ranking, and Random Ranking (strategies hereafter) to reach the adaptation goal is shown in Figure 9. The result shows that Designer Ranking required fewer iterations to achieve the adaptation goal. Also, Designer Ranking ended up with a smaller standard deviation. However, the difference between the average number of iterations is not significant enough to firmly conclude whether one strategy is better than others.

To assess the performance of Designer Ranking against Leverage Point Ranking and Random Ranking, we calculated the frequency of the required iterations to meet adaptation goals. Figure 10 shows the distribution of the number of iterations required to achieve the adaptation goal. From Figure 10, we noticed that all rankings yield approximately a similar number of failures (approximately 30%) to reach the response time of less than 500ms. We speculate that the selection of a different set of control points would lead to better results. We can make two observations: (1) All rankings eventually reached the adaptation goal, given enough time; and (2) the rankings affect the speed of achieving the adaptation goal. Figure 11 depicts the Observation (2). 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 11, the Designer Ranking

[14]Number of iteration refers to rise time (time needed for the system to get back to the desired state after deviations) in control theory.

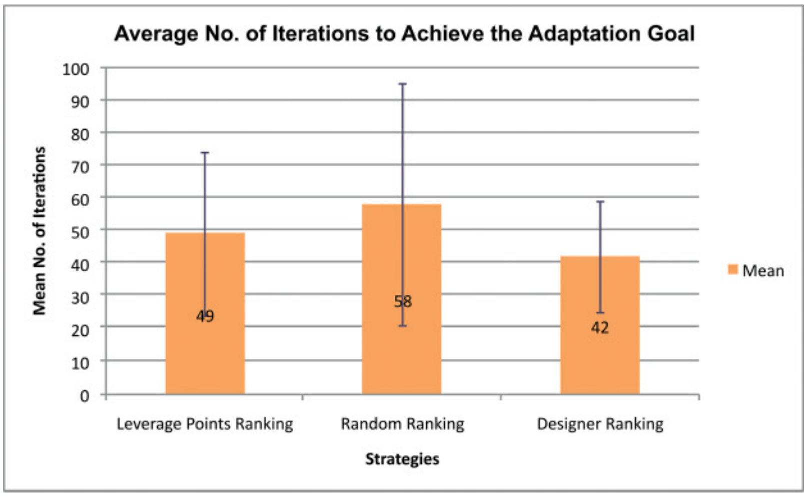


Fig. 9. Average number of iterations required to reach the adaptation goal.

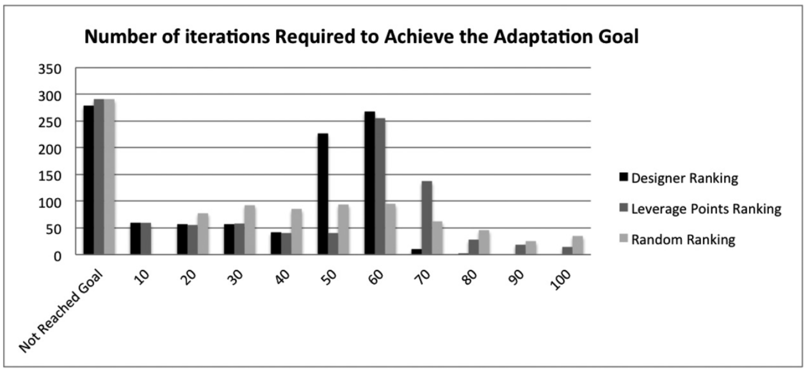


Fig. 10. Histogram of iterations required to reach the adaptation goal.

and Leverage Points Ranking have a steep slope and overlap, meaning that 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; as a result, it does not converge fast enough to reach the adaptation goal. This is due to the effect of the 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 of adding web servers as their first control point.

In this case study, we demonstrated that, when a group of designers used our methodology to rank control points, the result of ranking was similar with the objective ranking obtained from the model. These results show that our proposed methodology can help designers develop an adaptive system without using explicit application models.

5. CASE STUDY 2: ADAPTIVE SHOPPING CART WITH CONFLICTING ADAPTATION GOALS

In this section, we focus on a case study that includes stakeholders' conflicting adaptation goals. We consider two goals: G1: Minimize Resource Cost and G2: Response Time Less Than 500ms. Moreover, we set to compare the results of our search-based

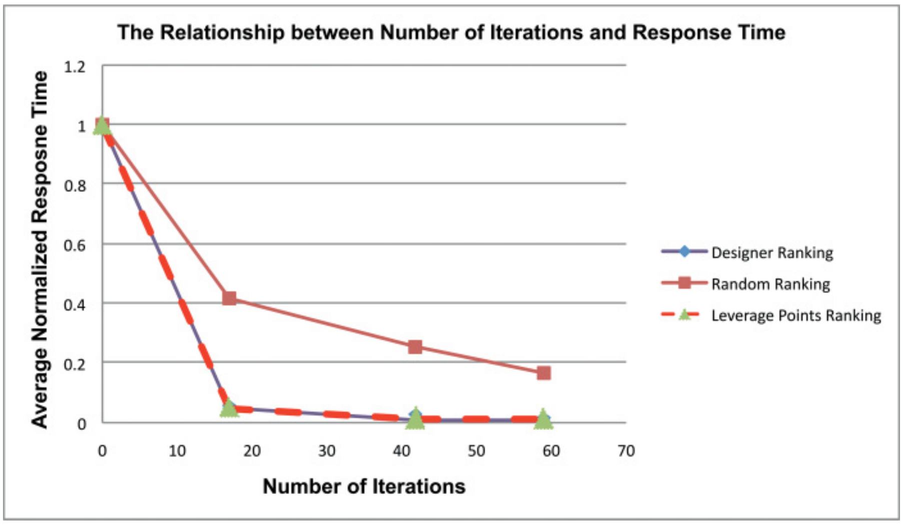


Fig. 11. Effect of the control points on feedback loop.

algorithm with an oracle solution in order to assess the quality of the search-based algorithm to meet the adaptation goals. We reviewed the pricing model from public cloud providers and noticed a correlation between the number of CPUs and the cost of an instance per hour. Moreover, for general purpose instances, the price of the CPU is often a dominant factor in determining the cost of an instance. Therefore, the cost of an instance is proportional to the number of CPUs. We also noticed that the cost of data transfer varies between different regions and across different cloud providers. Also, the cost of data transfer between instances in the same region is $0. In this article, we assumed that all instances are in the same region, a frequent practical case. In Section 5.1, we show how our methodology meets conflicting adaptation goals. We conduct a comparison between our search-based algorithm and oracle in Section 5.2 to show that the search-based algorithm produces relatively good results when adaptation goals are in conflict.

5.1. Cost and Performance of Adaptive Web Application in the Cloud

This experiment demonstrates how our proposed methodology assists with designing an adaptive application with conflicting adaptation goals such as G1: Minimize Resource Cost and G2: Response Time Less Than 500ms. We assume that the runtime cost is proportional to the number of CPUs used in each deployment due to a correlation between instance prices and number of CPUs (as, e.g., in Amazon EC2). According to our methodology, we express our adaptation goals as:

$$w_1 * max(R - 500ms, 0) + w_2 * f * \texttt{numberOfCPUs}, \tag{4}$$

where w_1 and w_2 are the weight factors for SLA misses cost and for resource cost, respectively; f is the price per CPU. We assume both cost terms have the same weight and we consider the price per CPU to be 1 (unit). Next, according to our methodology, we separately select control points for the adaptation goals. To achieve the goal G2: Response Time Less Than 500ms, we used the elicited control points from our previous experiment, shown in Table IV. To achieve the goal G1: Minimize Resource Cost, we selected the Remove Web Servers control point from Figure 6.

Similar to previous experiments, we adapted 1000 random deployments. In order to meet the adaptation goals, we run our search-based algorithm in two settings: (1) with

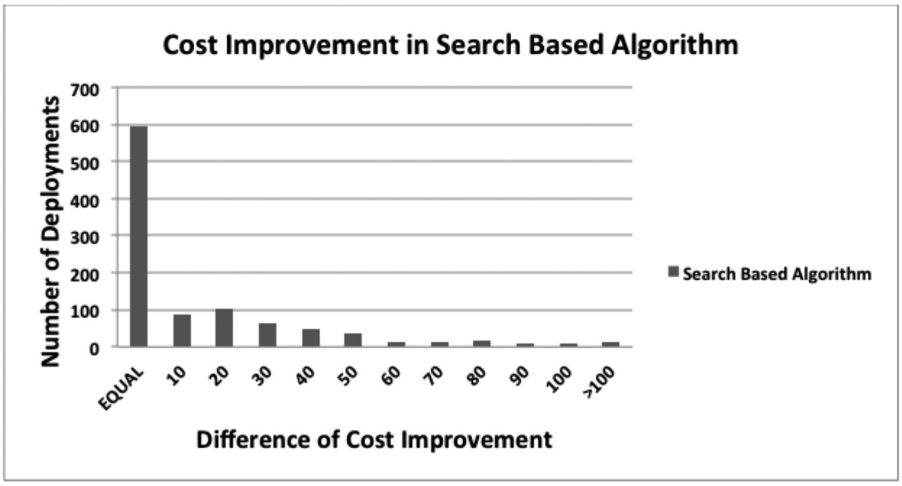


Fig. 12. Cost improvement in search-based algorithm.

Equation (4) capturing ONLY goal G2 (first term of Equation (4)) and the corresponding set of control points; (2) with Equation (4) including both goals, G1 and G2 (both terms of Equation (4)), and their corresponding sets of control points. The summary of the result is shown in Figure 12. In both runs, the search-based algorithm was able to meet the adaptation goals for the same random deployments, and in 60% of cases (denoted in Figure 12 as EQUAL), it reached the same cost. The search-based algorithm using the objective function from Equation (4) and an additional control point, Remove Web Servers (setting 2, earlier), achieved a cost reduction of at least 20% in 30% of cases. Moreover, in about 100 deployments, the improvement in total cost was more than 20% (third bar from left in Figure 12). The case study illustrates that, over a large set of experiments and deployments, the proposed search algorithm achieves two conflicting goals: G1: Minimize Resource Cost and G2: Response Time Less Than 500ms.

5.2. Evaluating the Search-Based Algorithm

The previous case study showed that our search algorithm can accommodate and deal with conflicting goals. However, we do not know how efficient the algorithm is. In this section, we address the efficiency of the search-based algorithm to meet its adaptation goals. Since our search algorithm is a heuristic, we wanted to compare it with a known efficient optimization algorithm, available in the literature. We call this efficient algorithm an oracle. To achieve this, we compared the solution obtained through the search-based algorithm from Section 3.4 with the oracle solution. The oracle should discover the best system's topology from any initial deployment, which aligns with the stakeholders' goals. (Note that this type of optimization is NP-complete, therefore, the oracle solution is not optimal but among the best we can get). In Section 5.2.1, the oracle is described.

For this experiment, we used the same settings as in Section 5.1: two conflicting goals, G1: Minimize Resource Cost and G2: Response Time Less Than 500ms; the objective function represented by Equation (4); and control points identical with those defined in Section 5.1. We generated 1000 random deployments and used the oracle to find the cheapest topology to meet the response time of less than 500ms. In parallel, we applied our search-based algorithm for all deployments and calculated the cost of each deployment. We then compared the values of the objective functions produced by the oracle with that produced by the search-based algorithm. Both approaches used

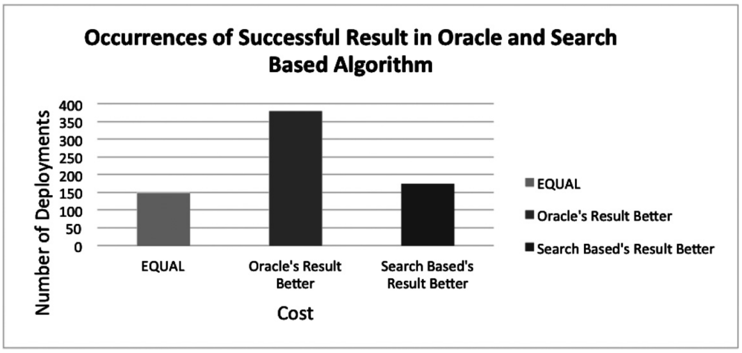


Fig. 13. Occurrences of successful result in the oracle and search-based algorithm.

the same control points to achieve the objectives. In this experiment, we assume that cost is calculated based on the number of CPUs used in a topology, similar to the previous experiment. The result of our comparison between the oracle and search-based algorithm is presented in Section 5.2.2.

5.2.1. Oracle. The oracle discovers the cheapest topology to meet the response time objective by applying a mathematical optimization method. Our mathematical optimization method utilizes the Sequential Quadratic Programming (SQP) [Boggs and Tolle 1995] algorithm. SQP is an iterative method for nonlinear optimization. We utilized many mathematical optimization methods, and the SQP algorithm was able to find a topology for most cases in order to achieve the response time of less than 500ms.

We formalize the SQP optimization problem by defining an objective function:

$$\texttt{numberOfServers} * \texttt{number of CPUs in instanceType}$$

with constraints $R < 500$, where R is response time. We calculated R using OPERA, where $R =$ OPERA (instance type, disk type, number of web servers, number of threads, network latency, and bandwidth). For every argument, we used the same boundary as defined in Table VI. Note that the `numberOfServers * number of CPUs for instance type` is equal to `numberOfCPUs`, which is the cost of the topology in the cloud. In other words, our objective function minimizes the cost of the topology in the cloud.

To improve the efficiency of our optimization method, we reduced the dimension of the search space by considering only instance type and disk type as parameters. Since we assigned four possible values (i.e., small, medium, large, and xlarge) for instance type and disk type, we solved 16 optimization for all combinations of instance type and disk type. We then selected the cheapest topology among the 16 solutions as our final oracle result. The next section describes the result obtained from the cost comparison of the search-based algorithm and oracle.

5.2.2. Comparison of the Results between Oracle and Search-Based Algorithm. We generated 1000 random deployments and calculated the cost of the best possible solution obtained by both algorithms. As shown in Figure 13, both algorithms achieved the same cost in 148 cases. Oracle discovered better topologies in 379 cases, and the search-based algorithm was able to outperform oracle in 174 cases.

Let us focus on cases in which both algorithms were able to find a satisfactory solution. In other words, they both found a topology for which response time of less than 500ms was met. The detailed comparison between the oracle and search-based algorithm is presented in Figure 14.

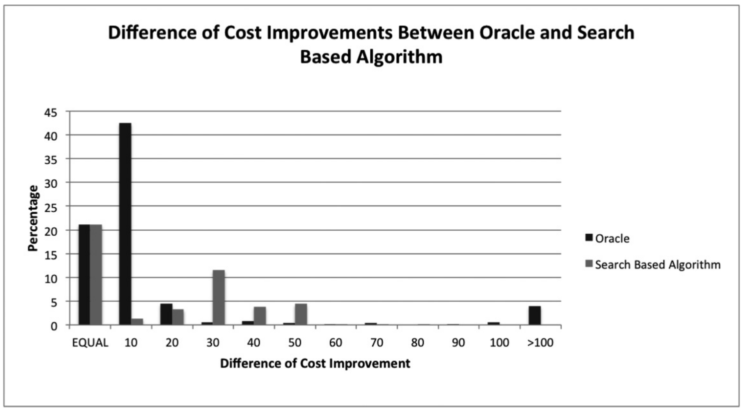


Fig. 14. Comparison of cost improvement between the oracle and search-based algorithm.

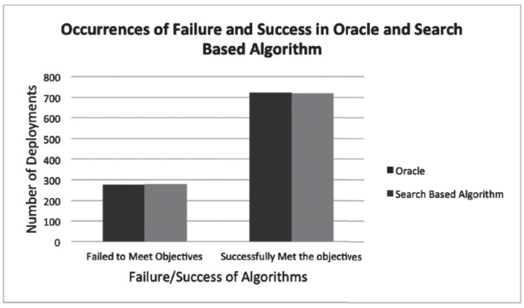


Fig. 15. Success and failure occurrences in algorithms.

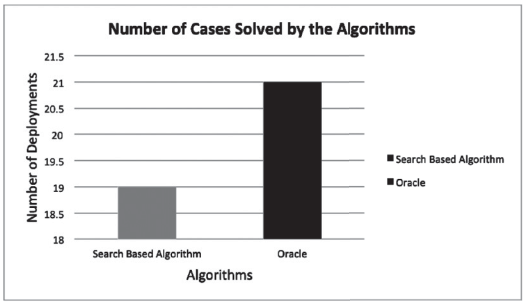


Fig. 16. Number of cases solved by algorithms.

In approximately 21%, both the oracle and search-based algorithm found a topology with the same cost. In approximately 42% of cases, the oracle solution was better than the search-based algorithm up to only 10%. In approximately 4% of cases, the oracle was better than the search-based algorithm up to 20%. Surprisingly, in 25% of cases, the search-based algorithm outperforms the oracle. In only 8% of cases, the oracle outperformed our search-based algorithm. As can be clearly seen, in the majority of cases, our search-based algorithm provides a satisfactory result. We also noticed that the search-based algorithm required significantly fewer numbers of iterations to reach the adaptation goals.

There were cases in which both algorithms failed to find a solution that satisfied both goals. In Figure 15, we show how many times each algorithm failed to reach the response time. In addition, we show in Figure 16 how many times one algorithm failed whereas another was able to find a solution. As a result, only in 3% of cases, the search-based algorithm failed to find a solution while the oracle was able to. Interestingly, the search-based algorithm was able to find a solution in approximately 3% of cases in which the oracle failed to find one. This is explained by the fact that the the oracle is not optimal either.

These experiments, run across many deployments, allow us to conclude that the adaptation method based on the search-based algorithm yields promising results. Moreover, in 63% of cases, our search-based algorithm using the designers' ranking reached the adaptation goal (response time less than 500ms) and the application runtime cost is less than 10% of the cost obtained by the oracle (analytical model). This shows the effectiveness of the search-based algorithm using designers' rank.

6. RELATED WORK

A key challenge posed by autonomic computing [Ganek and Corbi 2003; Kephart and Chess 2003] is to manage systems to handle uncertainty in their execution and environment. There are several proposals in the literature for designing adaptive systems [Cheng et al. 2009; Castañeda et al. 2014]. Architectural approaches have been extensively used in designing adaptive systems [Kramer and Magee 2007; Garlan et al. 2004; Peyman et al. 1999; Menasce et al. 2011; Tamura et al. 2013]. For instance, Troubitsyna and Javed [2014] propose an approach to design and develop a fault-tolerance adaptive system. Requirements engineering approaches have been also suggested to support the adaptation [Baresi and Pasquale 2010; Qureshi et al. 2010; Silva Souza et al. 2011; Souza et al. 2012].

Silva Souza et al. [2011] 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 considered, as well as the situations in which the system is adapted. The authors provide a 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.

Furthermore, the authors extended their work in Silva Souza et al. [2012] to introduce evolution requirements. They 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.

Recently, Angelopoulos et al. [2014] proposed an adaptive approach to deal with multiple requirement failures at runtime. The authors use AHP to prioritize requirements in order to select which failing requirement should be given higher priority to be fixed. In our work, we involve designers in ranking control points in order to tune the overall state of the system through the objective function.

Regarding the design phase, this is done in an ad hoc manner by developers/designers [Brun et al. 2013]. We used the Brun et al. [2013] approach as a blueprint for designing new adaptive systems. Our article focuses on the control points as being the main artifacts that drive the design of feedback loops. Therefore, our proposed method enhances other methodologies. Moreover, we focus on NFR goals in the context of the cloud, and use quantitative methods to guide the design.

Andersson et al. [2009] have proposed modeling dimensions that describe various aspects of the adaptation, which allow engineers to identify properties of self-adaptation and select the proper solution. Their study aims to identify and compare important aspects of self-adaptive systems. Thus, this classification of modeling dimensions needs to be considered when modeling an adaptive system. The authors categorize the points of variation, modeling dimensions, into four groups. The first group, *Goals*, is associated with a system's goals. The second group, *Change*, is associated with the cause of change in a system. The third group, *Mechanism*, deals with mechanisms to achieve change in a system. The fourth group, *Effects*, deals with the effects of adaptation on a system. Within each group, the authors have identified several dimensions, which focus on specific parts of the system that are pertinent to self-adaptation. While Andersson et al. [2009] identify the challenges in modelling adaptive systems, our work is a concrete study of the dimensions and a methodology to guide the design.

Using multiple control points to adapt the system has been attempted previously [Scandurra et al. 2012; Litoiu et al. 2005]. For example, Litoiu et al. [2005] 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 nonfunctional requirements. They present a general architecture of how these loops can work at different levels of granularity. These works are beneficial, although they do not focus on gathering nonfunctional adaptive requirements or ranking them. Neither do they consider strategies for combining multiple control points.

Implementing controllers as optimization algorithms is common in the cloud environment; there are many researchers addressing this issue. Li et al. [2009, 2011] have investigated dynamic resource allocation to meet application service level agreements. However, our article does not focus on the optimization algorithm itself, but on the methodology to build the objective function to optimize. At the same time, we achieve the optimization through many control points.

Villegas et al. [2011] present a framework for evaluating quality-driven self-adaptive systems by analyzing the self-adaptive approaches in the literature and the adaptation properties used in control theory. The authors map the adaptation properties, such as robustness to quality attributes (e.g., availability). Metrics are then used to evaluate the quality attributes. For example, response time is used as a metric to assess the performance of a system. Their framework is complementary to our approach, as it provides guidelines for the stakeholders to map metrics to adaptation goals, which is essential in our methodology for defining the objective function (see Section 3).

7. VALIDITY THREATS

We conducted our experiments using a simulation rather than running a real application in the cloud. The main reason that we used simulation was to test our methodology on large numbers of applications, which was not practical with real-life applications. We simulated 1000 different deployments and workloads. We used OPERA, a Layered Queuing Modeling tool, to model applications running in the cloud environment. With only 1000 experiments conducted in a simulated environment using 6 control points, there is room for additional validation (e.g., to validate our approach on a large number of control points and conflicting adaptations).

The remove web servers control point that we used for achieving the minimum resource cost for the cloud was efficient enough due to define a cost as a linear function from a number of CPUs.

8. CONCLUSIONS AND FUTURE WORK

In this articlde, we proposed a quantitative method to design adaptive web applications deployed in cloud environments. The goal of the adaptation is to optimize an objective function defined over NFR goals.

The main contribution of the article is the introduction of control points as the first class adaptation elements. We showed the building process of control point models for clouds, multiclouds, and applications to facilitate the elicitation process. We explained the ranking process of the control points.

We then built an adaptation strategy based on MAPE-k loops centered around a search-based method. The proposed search-based algorithm uses the most effective control points to achieve the adaptation goals. 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 objective function. We then select the next control point and repeat the cycle until the high-level objectives are met. The objective function indicates whether an application is going in the right direction when acting on the control points. The smaller the value of objective function, the better the compliance with all objective goals.

Our experimental evaluation showed that we are able to attain reasonable adaptation results using our methodology. In the future, we would like to extend our methodology to include multiple feedback loops and prioritize them. We also plan for further experiments with other types of optimization algorithms. Since we got satisfactory results, in the future, we are planning to apply this methodology to manage real-life applications.

REFERENCES

Jesper Andersson, Rogerio De Lemos, Sam Malek, and Danny Weyns. 2009. Modeling dimensions of self-adaptive software systems. In *Software Engineering for Self-Adaptive Systems*, B. H. C. Cheng et al. (Eds.), Lecture Notes in Computer Science, Vol. 5525, Springer, Berlin, 27–47.

Konstantinos Angelopoulos, Vítor E. Silva Souza, and John Mylopoulos. 2014. Dealing with multiple failures in zanshin: A control-theoretic approach. In *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. ACM, New York, NY, 165–174.

Luciano Baresi and Liliana Pasquale. 2010. Live goals for adaptive service compositions. In *Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS'10)*. ACM, New York, NY, 114–123. DOI:http://dx.doi.org/10.1145/1808984.1808997

Paul T. Boggs and Jon W. Tolle. 1995. Sequential quadratic programming. *Acta Numerica* 4, 1–51.

Yuriy Brun, Ron Desmarais, Kurt Geihs, Marin Litoiu, Antonia Lopes, Mary Shaw, and Michael Smit. 2013. A design space for self-adaptive systems. In *Software Engineering for Self-Adaptive Systems II*, R. de Lemos et al. (Eds.), Lecture Notes in Computer Science, Vol. 7475. Springer, Berlin, 33–50. DOI:http://dx.doi.org/10.1007/978-3-642-35813-5_2

Yuriy Brun,Giovanna Di Marzo Serugendo, Cristina Gacek, et al. 2009. In Engineering Self-Adaptive Systems Through Feedback Loops. In *Software Engineering for Self-Adaptive Systems*, B. H. C. Cheng et al. (Eds.), Lecture Notes in Computer Science, Vol. 5525, Springer, Berlin, 48–70. DOI:http://dx.doi.org/10.1007/978-3-642-02161-9_3

Evellin C. S. Cardoso, João Paulo A. Almeida, Giancarlo Guizzardi, and Renata S. S. Guizzardi. 2009. Eliciting goals for business process models with non-functional requirements catalogues. In *Enterprise, Business-Process and Information Systems Modeling*, T. Halpin et al. (Eds.), Lecture Notes in Business Information Processing, Vol. 29, Springer, Berlin, 33–45.

Lorena Castañeda, Norha M. Villegas, and Hausi A. Müller. 2014. Self-adaptive applications: On the development of personalized web-tasking systems. In *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. ACM, New York, NY, 49–54.

Betty H. C. Cheng, Rogerio De Lemos, Holger Giese, Paola Inverardi, Jeff Magee, Jesper Andersson, Basil Becker, Nelly Bencomo, Yuriy Brun, Bojan Cukic, and others. 2009. Software engineering for self-adaptive systems: A research roadmap. In *Software Engineering for Self-Adaptive Systems*, Springer, 1–26.

Betty H. Cheng, Rogerio de Lemos, Holger Giese et al. 2009. Software engineering for self-adaptive systems. Software Engineering for Self-Adaptive Systems: A Research Roadmap. In *Self-Adaptive Systems*, B. H. C. Cheng et al. (Eds.), Lecture Notes in Computer Science, Springer, Berlin, 1–26. DOI:http://dx.doi.org/10.1007/978-3-642-02161-9_1

Lawrence Chung, B. Nixon, E. Yu, and J. Mylopoulos. 2000. Non-functional requirements. *Software Engineering.*

Luiz Marcio Cysneiros and Eric Yu. 2004. Non-functional requirements elicitation. In J. C. S. Prado Leite et al. (Eds.), Kluwer Academic Publishers, The Netherlands, 115–138.

A. G. Ganek and T. A. Corbi. 2003. The dawning of the autonomic computing era. *IBM Systems Journal* 42, 1, 5–18. DOI:http://dx.doi.org/10.1147/sj.421.0005

D. Garlan, Shang-Wen Cheng, An-Cheng Huang, B. Schmerl, and P. Steenkiste. 2004. Rainbow: Architecture-based self-adaptation with reusable infrastructure. *Computer* 37, 10, 46–54. DOI:http://dx.doi.org/10.1109/MC.2004.175

Markus C. Huebscher and Julie A. McCann. 2008. A survey of autonomic computing-degrees, models, and applications. *ACM Computing Surveys* 40, 3, 7.

J. O. Kephart and D. M. Chess. 2003. The vision of autonomic computing. *Computer* 36, 1, 41–50. DOI:http://dx.doi.org/10.1109/MC.2003.1160055

J. Kramer and J. Magee. 2007. Self-managed systems: An architectural challenge. In *Future of Software Engineering, 2007 (FOSE'07)*. 259–268. DOI:http://dx.doi.org/10.1109/FOSE.2007.19

Jim Li, John Chinneck, Murray Woodside, Marin Litoiu, and Gabriel Iszlai. 2009. Performance model driven QoS guarantees and optimization in clouds. In *Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*. IEEE Computer Society, 15–22.

Jim Zw Li, Murray Woodside, John Chinneck, and Marin Litoiu. 2011. CloudOpt: Multi-goal optimization of application deployments across a cloud. In *Proceedings of the 7th International Conference on Network and Services Management (CNSM'11)*. International Federation for Information Processing, 162–170. http://dl.acm.org.ezproxy.library.yorku.ca/citation.cfm?id=2147671.2147697

Marin Litoiu and Cornel Barna. 2013. A performance evaluation framework for web applications. *Journal of Software: Evolution and Process* 25, 8, 871–890. DOI:http://dx.doi.org/10.1002/smr.1563

Marin Litoiu, Murray Woodside, and Tao Zheng. 2005. Hierarchical model-based autonomic control of software systems. In *ACM SIGSOFT Software Engineering Notes*, Vol. 30. ACM, New York, NY, 1–7.

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

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

Zbigniew Michalewicz, Dipankar Dasgupta, Rodolphe G. Le Riche, and Marc Schoenauer. 1996. Evolutionary algorithms for constrained engineering problems. *Computers & Industrial Engineering* 30, 4, 851–870.

Peyman Oreizy, Michael M. Gorlick, Richard N. Taylor, et al. 1999. An architecture-based approach to self-adaptive software. *IEEE Intelligent Systems and Their Applications* 14, 3, 54–62.

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

Ervin Y. Rodin. 1977. Nonlinear Programming Analysis and Methods: by Mordecal Avriel. Prentice-Hall, New York (July 1976) 512 pp. (1977).

Stuart Russell and Peter Norvig. 1995. *Artificial intelligence: A modern approach.* Prentice-Hall, Englewood Cliffs, NJ.

T. L. Saaty. 1980. *The Analytic Hierarchy Process*. McGraw-Hill International, Maidenhead, UK.

Mazeiar Salehie and Ladan Tahvildari. 2009. Self-adaptive software: Landscape and research challenges. *ACM Transactions on Autonomous and Adaptive Systems* 4, 2, Article 14, 42 pages. DOI:http://dx.doi.org/10.1145/1516533.1516538

Patrizia Scandurra, Claudia Raibulet, Pasqualina Potena, Raffaela Mirandola, and Rafael Capilla. 2012. A layered coordination framework for optimizing resource allocation in adapting cloud-based applications. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing (SAC'12)*. ACM, New York, NY, 471–472. DOI:http://dx.doi.org/10.1145/2245276.2245366

Vítor E. Silva Souza, Alexei Lapouchnian, William N. Robinson, and John Mylopoulos. 2011. Awareness requirements for adaptive systems. In *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. ACM, New York, NY, 60–69.

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

Alice E. Smith and David W. Coit. 1997. Constraint handling techniques-penalty functions. In *Handbook of Evolutionary Computation*. Oxford University Press and Institute of Physics Publishing, New York, NY.

Vıtor E. Silva Souza, Alexei Lapouchnian, and John Mylopoulos. 2012. (Requirement) evolution requirements for adaptive systems. In *Proceedings of the 2012 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS'12)*. IEEE, 155–164.

Gabriel Tamura, Norha M. Villegas, Hausi A. Müller, Laurence Duchien, and Lionel Seinturier. 2013. Improving context-awareness in self-adaptation using the DYNAMICO reference model. In *Proceedings of the 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS'13)*. IEEE Press, Piscataway, NJ, 153–162. http://dl.acm.org/citation.cfm?id=2487336.2487361

Elena Troubitsyna and Kashif Javed. 2014. Towards systematic design of adaptive fault tolerant systems. In *Proceedings of the 6th International Conference on Adaptive and Self-Adaptive Systems and Applications (ADAPTIVE'14)*. 15–21.

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

Norha M. Villegas, Hausi A. Müller, Gabriel Tamura, Laurence Duchien, and Rubby Casallas. 2011. A framework for evaluating quality-driven self-adaptive software systems. In *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS'11)*. ACM, New York, NY, 80–89. DOI: http://dx.doi.org/10.1145/1988008.1988020

Parisa Zoghi, Mark Shtern, and Marin Litoiu. 2014. 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, New York, NY, 7–16. DOI: http://dx.doi.org/10.1145/2593929.2593935

Received October 2014; revised June 2015; accepted September 2015