Designing Search Based Adaptive Systems: A Quantitative Approach

Parisa Zoghi
York University
Toronto, ON
Canada
pzoghi@yorku.ca

Mark Shtern
York University
Toronto, ON
Canada
mark@cse.yorku.ca

Marin Litoiu
York University
Toronto, ON
Canada
mlitoiu@yorku.ca

ABSTRACT

Designing an adaptive system to meet its quality constraints in the face of environmental uncertainties 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 paper 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 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. We conducted several experiments to evaluate the different phases of our methodology.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

General Terms

Theory, Design

Keywords

adaptive systems, design, cloud computing, performance

1. INTRODUCTION

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 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, unexpected failures, etc.

Adaptive design such as autonomic computing [11][12][8] can help achieve the NFRs. This paper focuses on designing cloud-based web applications by creating the mechanisms for achieving the adaptation goals. We formalize our problem as a multi-criteria optimization problem and identify the variables that enable the optimization.

Our contribution is 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 [28]. We define control points as those artifacts in a system that modified at run-time cause controlled changes in the system; thus, affect 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 one is the ranking of control points in the early design phase, which enables prioritization of control points and an informed decision-making about which ones are essential to meet the stakeholders' objectives. The second one is that conflict resolution (between competing NFRs) is solved through the use of feedback loops at runtime rather than arbitrarily at the elicitation phase. Our final contribution is a case study that demonstrates the capability of our methodology in designing an adaptive application in cloud.

The remainder of this paper 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 discuss related work in Section 5 and conclude the paper in Section 6.

2. MOTIVATION

Autonomic computing has been suggested as a general guideline for designing adaptive systems [10]. Survey papers such as [5] and [23] discuss the challenges of designing and engineering such systems.

Brun et al. emphasize the importance of feedback loops in

[1] In cloud, performance goals are achieved through scalability. 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/`

SEAMS '14, June 2-3, 2014, Hyderabad, India

designing adaptive systems [4]. 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 [3]. 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 [3]. The authors present a conceptual model of how to identify different components of an adaptive system by answering a set of questions along 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 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 the adaptation goals [5] 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 identify 10 control points, elicit feedback loops for them, and finally 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 adequate 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.

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

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] architectural style [9].The software system can have multiple adaptation goals, which can be conflicting. For instance, adding more resources improves the performance goal, but negatively affects the cost goal.

In this work, we assume the adaptation (NFR) goals (such as cost minimization 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 elicitation process. We elicit control points in the system that can affect one or more adaptation goals. 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 rank 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 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 formulas (1) and (2), m_i is an application metric (e.g., response time, throughput, utilization, cost, etc.), and F_i and φ_i are real number functions.

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

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

For example, (1) the cost of running a system in the cloud should be minimized, or (2) the cost of the running an application should be less than 8 cents multiplied by the number of users, or (2) the response time of checkout should be less than 1 second.

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, O, defined as follows:

$$O = \sum_{i=0}^{j} F_i w_i + \sum_{i=0}^{l} (w_i P_i(\varphi_i - F_i)^2 (1 - sign(\varphi_i - F_i))) \tag{3}$$

The P_i is the cost of missing the objectives (violating SLOs), which is defined by the stakeholders.

[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/

Functions similar to (3) have been used in other contexts, such as in genetic algorithms optimization [27],[18]. Note that the first term in function O refers to goals that need to be minimized (1) while the second term refers to boundary goals. The sign function associated with the second terms makes sure the terms are 0 when the goals are exceeded.

The objective function indicates how well the overall goals are achieved; therefore, it should be as small as possible. 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 (3). The feedback loop can optimize the function (3) using well-known algorithms, such as Hill climbing [21].

Any change in control points affects the metrics m_1, m_2, $m_3, ..., m_n$, and as a result, the value of objective function O will be changed. The smallest value of the objective function O 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 influenced control points, and second, how to control them efficiently to make the objective function O small. The first problem is addressed in sections 3.2 and 3.3), and the second problem is discussed in section 3.4.

3.2 Eliciting Control Points

The process of eliciting control points resembles NFR elicitation. One such approach is the NFR Framework[6], in which higher level goals are decomposed into sub-goals using an AND/OR tree until they cannot be decomposed any further (operationalized). Creating catalogues to assist with eliciting NFRs have been used [7][6]. We propose a control point model to assist us with eliciting control points. Our proposed method draws from goal modelling methodology. Hence, 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 high level goal, 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 goals or sub-goals 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/sub-goals into smaller sub-goals. It also helps to reduce the risks associated with missing operations or misunderstanding adaptation goal/sub-goals.

One difference between our control point model and a goal model is that we decompose each adaptation goal separately and only elicit control points with regard to that goal. Another difference is the conflict resolution. A control point model is conflict-free since all operations/sub-goals should only contribute for meeting the top-level adaptation goal. We are not concerned with any goal conflicts in early stage of elicitation. Our control point model only addresses 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 (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 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 points model by asking a series of questions for Application, Cloud and Multi-cloud as follow:

- **Cloud.** What cloud capability can help achieve an adaptation goal? How does cloud agility/elasticity help an application to meet objectives? What services offered by the cloud provider can aid in meeting your adaptation objectives? For instance, imagine the adaptation goal is less than 1 second response time. Amazon Web Services (AWS) Elastic Beanstalk [7] is an Elastic Compute Cloud (EC2)[8] 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.
- **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? For instance, the adaptation goal is less than 1 second response time and the application will be deployed in 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 [26].
- **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 runtime behaviour? For instance, let's assume an 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.

As an example, consider low cloud cost as an adaptation goal to be achieved. First, we decompose the adaptation goal into a metric, the cost of runtime system. We

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

then think of what influences the cost in cloud, and decompose our adaptation goal into sub-goals such as Decommission of under-utilized resources, Consolidate Resources, or Find cheaper resource/service. Now that we have our sub-goals, we think of decomposing them further to reach control points. Looking at the decommission of under-utilized resources sub-goal, we can remove or change VM instance types to achieve lower runtime cloud cost. Hence, removing a web server or changing its size to a smaller one, are examples of our control points.

Figure 1 and 2 show our elicited control points for meeting low cloud cost and low response time adaptation goals in cloud and application respectively. Therefore, these control point models can be treated as catalogues for knowledge reuse.

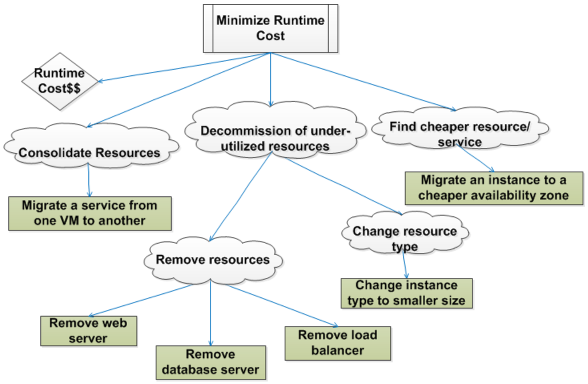

Minimize Runtime Cost
Runtime Cost$$
Consolidate Resources
Decommission of under-utilized resources
Find cheaper resource/ service
Migrate a service from one VM to another
Migrate an instance to a cheaper availability zone
Change resource type
Remove resources
Change instance type to smaller size
Remove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

Figure 1: Control Points Model for Runtime Cloud Cost

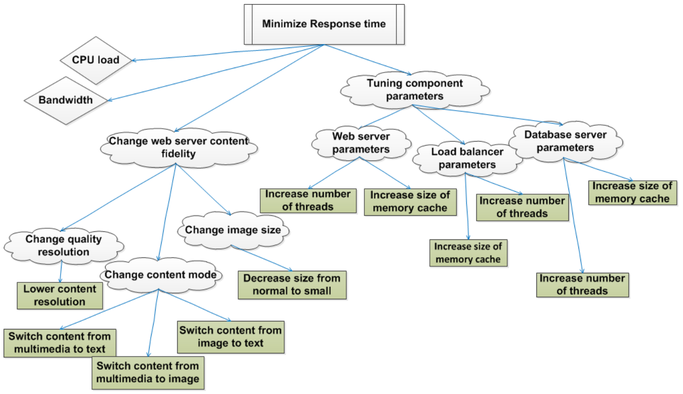

Minimize Response time
CPU load
Bandwidth
Tuning component parameters
Change web server content fidelity
Web server parameters
Load balancer parameters
Database server parameters
Increase number of threads
Increase size of memory cache
Increase number of threads
Increase size of memory cache
Change quality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

Figure 2: Control Points Model for Application Response Time

Table 1 shows the most important NFRs for web-applications that can be met through adaptation.

Applying our elicitation process, Table 2 shows a catalogue of possible control points for cloud environment. All the control points listed in Table 2 are applicable for web servers, load balancers, and database servers.

Table 1: NFRs and their corresponding Metrics

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

3.3 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 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. **Conduct pairwise comparisons.**

   We use pairwise comparisons as in the Analytic Hierarchy Process (AHP)[22], which compares two control points based on the higher level goal. To conduct pairwise comparisons, the designers are asked to choose between three options (A>B meaning A is preferred to B; A<B meaning B is 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 designer to perform comparisons. Each designer populates the upper triangle of a matrix[9] (see Figure 3) 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 each 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 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 the designers build a good mental model they then are ready for direct ranking. The results of direct ranks obtained from the designers are then aggregated.

3. **Aggregate the obtained ranks.**

   Finally, the final rank is obtained by aggregation of result as follows:

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

Table 2: Elicited Control Points for adaptation goals

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 availability zone/cheaper cloud provider		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		

(a) Rank Control Points Regarding a Goal. Count the number of times each control point was ranked 1^{st}, 2^{nd}, etc. The rank of the control point is the most agreed rank among all designers. The ranked control points are added to an ordered list, 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, now a decision can be made on how many control points to consider for implementation. This decision is not within the scope of the paper, we assume all are implemented.

3.4 Implement 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 the optimization algorithm that minimizes equation (3). There can be many ways to optimize function (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 X previous times. The second phase is to meet the selected adaptation goal by running the search-based adaptation algorithm (see Algorithm 1), which implements Hill climbing strategy. The optimization process is repeated until the stakeholders' high level objectives are met.

Now, we explain the details about the adaptation algorithm in the second phase. The adaptation algorithm (See 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.

Algorithm 1: Search based algorithm.

```
begin
    Select the control points contributes 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 $m0 - m1 >$ pre-define threshold then
                        is-improved = true
                        is-any-improvement=true
                    end
                    else
                        Rollback operation CP
                        is-improved = false
                    end
                end
                else
                    is-improved = false
                end
            until is-improved;
        end
    until is-any-improved;
end
```

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 goal is satisfied or no further improvements of objective function O can be found[10]. 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.

4. CASE STUDY: ADAPTIVE SHOPPING CART FOR CLOUD

This section presents a case study, the design of an adaptive shopping cart web application for cloud. 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 < 500 ms 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 4.2, we show how we validated the control point ranking. In 4.3, we discuss the efficiency of the search based adaptation process using the ranking.

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:

$$O = ((500ms - R)^2(1 - sign(500ms - R)))$$

4.1.2 *Elicit Control Points*

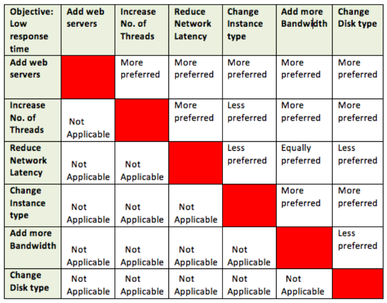
We selected control points from the catalogue for cloud environment (Table 2), and using control points model shown in Figure 2. 6 control points presented in Table 3, were selected for this study.

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 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 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 3 and Table 4 show a sample of the comparison matrix[11] and direct rank for a participant, Participant A. As shown in Table 4, Participant A 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 5.

Table 3: Elicited Control Points and their definition

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 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



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 3: Pairwise Comparisons for Low Response Time

Table 4: 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

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

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

Table 5: 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%

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, we call them Designer Ranking, (see Table 5) with an objective ranking obtained through model-based simulation (we call this ranking 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, etc.). This is equivalent with 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 5) 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[15].

Table 6 shows how we made the changes in the control points to bring the performance back to the 500ms.

The **Min** and **Max** columns in Table 6 reflect the minimum and maximum boundaries for control points.The **Step** 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= 10 ms, or reached the control points limits, or reached the adaptation goal, response time (500 ms). 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 reasonable number of iterations.

Table 6: Control Points Limitations

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

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

After obtaining the data, we counted the number of times each control point ranked 1st, 2nd, etc.The rank of the control point is the most agreed rank across all experiments. Table 7 shows the result.

Table 7: 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%

Tables 5 and 7 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 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 search based feedback 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 and those are described in sections 4.3.1 and 4.3.2. The experimental result is discussed in 4.3.3.

4.3.1 Random Ranking

We expect the impact of ranking is clear when we use two opposite ranking of control points: Experimental/Designer Ranking, and the inverse of Experimental Ranking. Table 8 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 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 [17]. We mapped Meadows' Leverage Points to our control points. Table 9 displays the mapping result.

We mapped the "Increase Number of Threads" and "Add

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

Table 8: 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 9: Leverage Points Ranking

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

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

web servers" control points to "Power to add, change, evolve, or self-organize system structure" leverage point (point #4 in Meadows list) since adding new threads creates snew information flows. Adding servers changes the system structure by creating new information flow through the system. Hence, they both correspond to number 4 in 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 the 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 10, we summarize the applications' response time prior to adaptation process, across all experiments.

Table 10: Distribution of Response Time Deviation

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

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. The summary of the collected data is presented below.

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 4. 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 strategy is better than others.

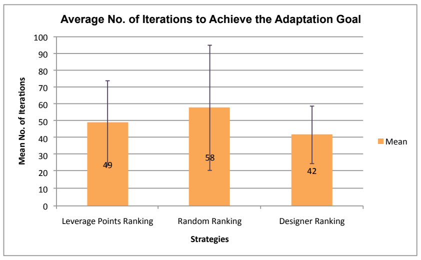


Figure 4: Average Number of Iterations Required to reach the adaptation goal

To assess the performance of Designer Ranking against Leverage Point Raking and Random Ranking, we calculated the frequency of the required iterations to meet adaptation goals. Figure 5 shows the distribution of the number of iterations required to achieve the adaptation goal. From Figure 5, 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 speculate the selection of 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 6 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 6, 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 order of control points. The Random Ranking starts with adding more bandwidth, thus, we do

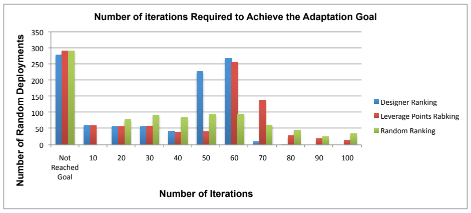


Figure 5: Histogram of Iterations Required to reach the adaptation goal

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.

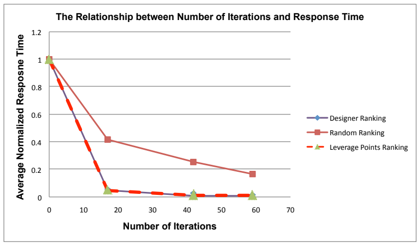


Figure 6: Effect of the Control Points on Feedback Loop

5. RELATED WORK

A key challenge posed by autonomic computing[8, 11] is to manage systems to handle uncertainty in their execution and environment. There are several proposals in the literature for designing adaptive systems. For instance, architectural approaches have been used in designing adaptive systems [12, 9, 19]. Requirements engineering approaches have been also suggested to support the adaptation [2, 20, 25]. Regarding the design phase, this is done in an ad hoc manner by developers/designers [3]. We used Brun et al. [3] approach as a blueprint for designing new adaptive systems. Our paper 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 cloud, and we use quantitative methods to guide the design.

Andersson et al. [1] have proposed modeling dimensions that describe various aspects of the adaptation, which allow for engineers to identify properties of self-adaptation and select 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. The authors categorize the points of variation, modeling dimensions, into four groups. Moreover, they state that each dimension focuses on a specific part of the system that is pertinent to self-adaptation. The first dimension, *Goals*, is associated with the systems' goals. The second dimension, *Change*, is associated with the cause of change. The third dimension, *Mechanism*, deals with mechanisms to achieve change, and the fourth dimension, *Effects*, deals with the effects of self-adaptability on the system. 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.

Using multiple control points to adapt the system has been attempted previously [24, 16]. For example, Litoiu et al. proposed a hierarchical model-based adaptation for tuning parameters in service-oriented architecture applications [16]. Their architecture consists of 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 gathering non-functional adaptive requirements or ranking them. Neither do they consider strategies for combining multiple control points.

Implementing controllers as optimization algorithms is common in cloud environment and there are many researchers addressing this issue. Litoiu et al. have investigated dynamic resource allocation to meet application service level agreements [13, 14]. However, our paper does not focus on 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.

6. CONCLUSIONS AND FUTURE WORK

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

The main contribution of the paper is the introduction of control points as the first class adaptation elements. We showed how we build control point models for clouds, multi clouds and applications to facilitate the elicitation process. We then explained how to rank the control points.

We then built an adaptation strategy based on MAPE-k loops centred 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 to 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 were 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.

We conducted our experiments using a simulation rather than running a real application on 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 cloud environment. Since we got satisfactory results, in the future, we are planning to apply this methodology to manage real life applications.

Acknowledgements

This research was supported by the SAVI Strategic Research Network (Smart Applications on Virtual Infrastructure), funded by NSERC (The Natural Sciences and Engineering Research Council of Canada) and by Connected Vehicles and Smart Transportation(CVST) funded Ontario Research

Fund. Lastly, thanks to Bradley Simmons for his helpful discussions and assistance during the writing of this document.

7. REFERENCES

[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*, SEAMS '10, pages 114–123. ACM, 2010.

[3] 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*, volume 7475 of *Lecture Notes in Computer Science*, pages 33–50. Springer Berlin Heidelberg, 2013.

[4] Brun, Yuriy et al. In *Software Engineering for Self-Adaptive Systems*, chapter Engineering Self-Adaptive Systems Through Feedback Loops, pages 48–70. Springer-Verlag, 2009.

[5] Cheng, Betty H. et al. Software engineering for self-adaptive systems. chapter Software Engineering for Self-Adaptive Systems: A Research Roadmap, pages 1–26. Springer-Verlag, 2009.

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

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

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

[9] D. Garlan, Shang-Wen Cheng, An-Cheng Huang, B. Schmerl, and P. Steenkiste. Rainbow: architecture-based self-adaptation with reusable infrastructure. *Computer*, 37(10):46–54, 2004.

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

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

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

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

[14] Jim Zw Li, Murray Woodside, John Chinneck, and Marin Litoiu. Cloudopt: Multi-goal optimization of application deployments across a cloud. In *Proceedings of the 7th International Conference on Network and Services Management*, CNSM '11, pages 162–170. International Federation for Information Processing, 2011.

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

[16] 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.

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

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

[19] Oreizy, Peyman et al. An architecture-based approach to self-adaptive software. *Intelligent Systems and Their Applications, IEEE*, 14(3):54–62, 1999.

[20] 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.

[21] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2009.

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

[23] Mazeiar Salehie and Ladan Tahvildari. Self-adaptive software: Landscape and research challenges. *ACM Trans. Auton. Adapt. Syst.*, 4(2):14:1–14:42, May 2009.

[24] 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*, SAC '12, pages 471–472. ACM, 2012.

[25] 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.

[26] 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.

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

[28] 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.