Autonomic Resource Provisioning for Cloud-Based Software

Pooyan Jamshidi
IC4, School of Computing, Dublin City University, Ireland.
pooyan.jamshidi@computing.dcu.ie

Aakash Ahmad
Lero, School of Computing, Dublin City University, Ireland.
ahmad.aakash@computing.dcu.ie

Claus Pahl
IC4, School of Computing, Dublin City University, Ireland.
claus.pahl@computing.dcu.ie

ABSTRACT

Cloud elasticity provides a software system with the ability to maintain optimal user experience by automatically acquiring and releasing resources, while paying only for what has been consumed. The mechanism for automatically adding or removing resources on the fly is referred to as auto-scaling. The state-of-the-practice with respect to auto-scaling involves specifying threshold-based rules to implement elasticity policies for cloud-based applications. However, there are several shortcomings regarding this approach. Firstly, the elasticity rules must be specified precisely by quantitative values, which requires deep knowledge and expertise. Furthermore, existing approaches do not explicitly deal with uncertainty in cloud-based software, where noise and unexpected events are common. This paper exploits fuzzy logic to enable qualitative specification of elasticity rules for cloud-based software. In addition, this paper discusses a control theoretical approach using type-2 fuzzy logic systems to reason about elasticity under uncertainties. We conduct several experiments to demonstrate that cloud-based software enhanced with such elasticity controller can robustly handle unexpected spikes in the workload and provide acceptable user experience. This translates into increased profit for the cloud application owner.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Reliability and availability;

General Terms

Management, Measurement, Performance, Experimentation.

Keywords

Cloud Computing, Auto-scaling, Elasticity, Uncertainty.

1. INTRODUCTION

Cloud computing platforms are widely used by major IT companies and startups to remain competitive [1]. Even traditional enterprises are attempting to exploit the benefits of cloud platforms [1]. The appealing characteristics that cloud platforms can provide include high-availability and low cost of maintenance [2]. However, the main selling point of cloud platforms is elasticity - i.e., the customers should only pay for what they have utilized [3]. Elasticity is the core design principle of elastic software that convey three aspects [3] [4]: (1) *scalability*, the ability of the system to sustain workload fluctuations, (2) *cost efficiency*, acquiring only the required resources by releasing unutilized ones, (3) *time efficiency*, acquiring and releasing resources as soon as a request is made.

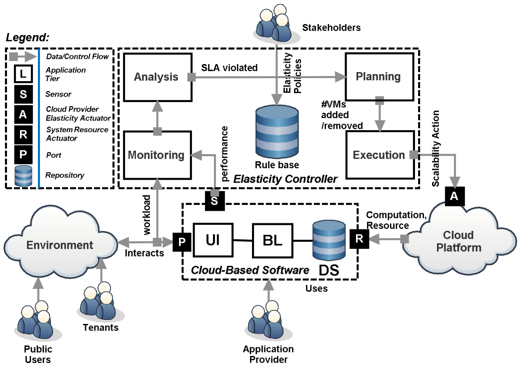

Figure 1. High-level view of elastic software.

Web-based software systems frequently experience load spikes. For example, one recent Facebook application experienced a 10 times increase in the number of users from 25,000 to 250,000 in just three days with up to 20,000 new registrations per hour in peak times [5]. Such typically business-critical systems must satisfy certain level of service level agreements (SLA), e.g., upper bounds on user perceived response time. Otherwise, unexpected loads cause a poor service level that frustrate end users. Amazon reported a loss of 245 million dollars for an increase of 100ms in response time [6]. To avoid such a situation and maintain service quality, the automated management of such applications is essential [7]. As a result, there has been a research and practice interest in automated resource provisioning for such applications [8].

The challenge of building elastic systems involves adjustment of resources along with load variations without the need for human interventions. Automated cloud-based scalability (i.e., auto-scaling) is one of the most recent advancements for dynamic resource provisioning [9] [10] [11]. To auto-scale an application, the state-of-the-practice involves specifying threshold-based rules to implement elasticity policies for cloud applications [10]. There remained several challenges that we intend to address in this work. Firstly, elasticity rules must be specified precisely by quantitative values. This requires expertise, which makes the accuracy of the policy subjective and prone to uncertainty. Furthermore, existing approaches make impractical assumptions about elastic systems and their environment. More specifically, they assume that stakeholders have a unified opinion about the thresholds in the rules. More importantly, they do not explicitly consider noises in the input data. However, these assumptions are barely valid in the cloud, where uncertainty in terms of noise and dynamic changes in the environment are frequent [12] [13]. The approaches that rely on such assumptions are not dependable [12] [13] [9].

The particular contribution of this paper is to develop an elasticity controller, called RobusT2Scale, which utilizes fuzzy logic to enable qualitative specifications of elasticity rules. Fuzzy logic systems (FLSs) [14] are known to enable manipulation of linguistic rules. This paper proposes an elasticity reasoning (encompasses *analysis* and *planning* in Figure 1) using type-2 FLSs [14].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SEAMS'14, June 2–3, 2014, Hyderabad, India.
Copyright 2014 ACM 978-1-4503-2864-7/14/06... $15.00.

Elasticity rule uncertainties occur due to the use of imprecise qualitative values. For example, a typical elasticity rule might be:

> "IF the workload is *high*, AND response-time is *slow*, THEN add two more VMs to the existing resources" (1.1)

In this situation, a type-2 FLS can provide an effective mechanism to represent the uncertainties in these *italicized* linguistic labels and the numerical manipulation of these rule to plan the scalability. We demonstrate that RobusT2Scale, via the fuzzy elasticity reasoning, is robust to several forms of changes in the environment, including unpredictable changes in requests and unpredictable degradations in response-time. A number of experimental results demonstrate the effectiveness of this approach in handling measurement noises when dealing with unexpected bursts in the requests. We demonstrate that our approach significantly outperforms two other provisioning policies, meeting response time obligations while greatly reducing the number of cloud resources.

The remainder of this paper is structured as follows. Section 2 motivates the research and provides an overview of the proposed solution. Section 3 reviews the background on mathematical foundations. Section 4 presents the proposed approach, including the details of the fuzzy elasticity controller. Section 5 reports the implementation details of RobusT2Scale followed by experimental evaluations in Section 6. The paper concludes with a review of related work and opportunities for future research.

2. CHALLENGES AND APPROACH

In this section, we use a running example to highlight the research challenges and to exemplify the proposed solution.

2.1 Motivating Example

Let us consider a multi-tenant software as a service (SaaS) that enables customers (tenants) to design, publish and collect the results of surveys [15]. Customers can manage their surveys by creating a subscription with the service and determining a location for their account and surveys. Public people can participate in the survey by completing the designed survey provided by the survey creator through a URL. Surveys usually run for a short period but may attract huge number of respondents. Because the nature of survey application includes sudden bursts in demand, it must be able to quickly grow or contract its deployment infrastructure.

To achieve the scalability requirements, the survey application is implemented as a cloud-based application. Figure 2 illustrates the high-level architectural view of the survey application. The architectural style of the application is a typical multi-tier cloud-based architecture, with every component running on cloud-based nodes. User interface (UI) runs on web nodes. Business logic (BL) runs on compute nodes and data storage (DS) runs on data nodes. A node might be a part of physical server (e.g., a virtual machine), a physical server or even the cluster of servers but we use node as a generic terms because the underlying resources are not relevant.

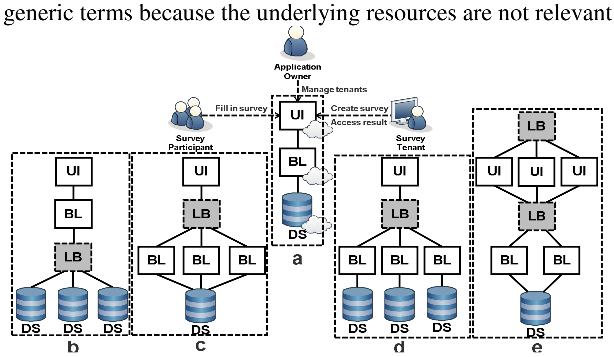

Figure 2. The survey application architecture.

2.2 Research Challenges

Let us imagine customers using the survey application continues to grow. The number of surveys with large number of respondents is increasing, leading to sudden spikes in application usage. In order to handle such bursts in usage, the resources for such application needs to be dynamically adjusted, see a number of possible configurations in Figure 2. In cloud platforms, auto-scaling enables the implementation of application's *elasticity policy* [10]. Similar to an adaptation policy [16], which governs how/when software components are added and removed from a software system, an elasticity policy [17] governs how/when resources are added and removed from a cloud-based application. In rule-based mechanisms, when the value of certain metrics, such as CPU utilization, exceed a predefined threshold, more nodes are added until the values have dropped to an acceptable level.

Formally, each auto-scaling rule involves several parameters, which are defined by the stakeholders [18]: 1) an upper threshold for a metric m and a time value $thr_{U,m}, tv_U$, 2) a lower threshold for a metric m and a time value $thr_{L,m}, tv_L$, 3) the number of VMs to be allocated or released n 4) two cool-down periods cd_U, cd_L. More specifically, the rules have the following structure:

"**IF** $m \geq thr_{U,m}$ for tv_U seconds **THEN** $\#VM = \#VM + n$ **AND** cool down for cd_U s" (2.1)

"**IF** $m \leq thr_{L,m}$ for tv_L seconds **THEN** $\#VM = \#VM - n$ **AND** cool down for cd_L s" (2.2)

If the value measured for metric m reaches a threshold $thr_{U,m}$ for tv_U seconds, n nodes will be added. For instance, if the average CPU utilization of current VMs in the business logic tier is above 85% for 600 seconds, 2 new VMs will be added in that tier.

Threshold-based rules can control the amount of resources by performing auto-scaling actions to adapt resources based on the demand. This has been shown in previous research (e.g., [19] [20] [7] [21]), on public clouds (e.g., Amazon EC2 [22], Microsoft Azure [23]), open cloud providers (OpenNabula [22]) and even third party services (e.g., RightScale [22]). Although this approach is popular, there are several challenges associated with this:

- *Challenge* 1. *Parameters' value prediction ahead of time*. The process of acquiring and releasing resources is not instant. First, the auto-scaling controller needs to invoke the cloud platform to initiate the acquisition process. The VMs will then be spun up and then the application needs to be deployed on the new machines. During this time, which may take on average 10 minutes [24], the cloud application is vulnerable to workload increase and as a result provide user dissatisfaction. Section 5.1 describes our approach to predict inputs.
- *Challenge 2. Qualitative specification of thresholds*. The specification of the rules requires careful setting out of the lower and upper thresholds. This requires deep knowledge about the behavior of the system over time [18]. Therefore, the overall accuracy of the policies remains subjective, which makes the resource provisioning prone to uncertainty. Section 4.3 shows our solution to enable qualitative specification of thresholds.
- *Challenge 3. Robust control of uncertainty*. The measurement data corresponds to a distribution of values. For instance, a probe monitoring the response time of an application hosted in the cloud may return slightly different value every point in time. This variation could be associated to the sensory noise [7]. This results in the oscillations for resource allocations [18]. Sections 4.3 to 4.6 describe our solution to determine the required resources under the presence of uncertainty.

2.3 Solution Overview

The problem of application elasticity falls into the category of autonomic computing [25], where systems make use of autonomic managers implementing feedback control loops (cf. Figure 1). Figure 1 gives an overview of the solution space: a scalable cloud-based application hosted on nodes obtained from a provider based on a pay-as-you-go lease. In the example in Section 2.1, a cloud-based application that serves requests from a dynamic set of tenants and public clients is introduced. Since the users are sensitive to performance of the application, the owner is presumed to have a service level objective (SLO) to characterize an acceptable performance. If an application does not violate SLOs, users have good experience. The purpose of the "controlled elasticity" is to grow and shrink resources to meet the SLO efficiently under the dynamic workload and to minimize the incurred cost. This work only targets applications that can benefit from such elasticity.

We implemented a controller that runs on behalf of cloud-based software and drives actuators to acquire/release nodes based on application status and environmental conditions. In particular, this paper makes the following contributions:

1. Our approach integrates a time-series technique with a fuzzy logic controller to realize a hybrid auto-scaler, which we call RobusT2Scale. This allows us to determine the right capacity in response to changes. We demonstrate that RobusT2Scale can handle most well-known change patterns in workload.
2. RobusT2Scale enables qualitative imprecise thresholds (e.g., "high", "low") for specifying elasticity rules. To the best of our knowledge, RobusT2Scale is the first auto-scaler to exhibit such flexibility in rule specification.
3. RobusT2Scale is robust to noisy data, which are collected based on client-side application-level measurements.

3. BACKGROUND

A *type-2* (T2) fuzzy set [26] [27] is an extension of *type-1* (T1) fuzzy set. At a specific value x' (cf. Figure 3), there is an interval instead of a crisp value. This leads to the definition of a three dimensional *membership function* (MF), a T2 MF, which characterizes a T2 *fuzzy set* (FS) (**Definition 1**). Note all definitions in this paper are standard definitions in fuzzy theory that we borrowed from literature (e.g., [28] [29] [30]), also compare to Figure 3 for a better understanding of what definitions convey.

Definition 1. A ***T2 FS***, $\tilde{R}$, is characterized by a T2 MF $\mu_{\tilde{R}}(x,u)$

$$\tilde{R} = \{((x,u), \mu_{\tilde{R}}(x,u)) | \forall x \in X, \forall u \in J_x, \mu_{\tilde{R}}(x,u) \leq 1\} \tag{3.1}$$

When these values have the same weight, it leads to definition of an *interval type-2 fuzzy set* (IT2 FS), defined in **Definition 2**.

Definition 2. If $\mu_{\tilde{R}}(x,u) = 1$, $\tilde{R}$ is an ***interval T2 FS*** (**IT2 FS**).

Therefore, the MF of IT2 FS can be fully specified by the two T1 MFs (cf. **Definition 4**). The area between the two MFs (the grey region in Figure 3) characterizes the uncertainty.

Definition 3. The uncertainty in the membership function of an IT2-FS, $\tilde{R}$, is called ***footprint of uncertainty*** (***FOU***) of $\tilde{R}$, i.e.,

$$FOU(\tilde{R}) = \bigcup_{x \in X} J_x = \{(x,u) | \forall x \in X, \forall u \in J_x\} \tag{3.2}$$

Definition 4. The ***upper membership function*** (***UMF***) and ***lower membership function*** (***LMF***) of $\tilde{R}$ are two T1-MFs $\overline{\mu}_{\tilde{R}}(x), \underline{\mu}_{\tilde{R}}(x)$ respectively that bound the FOU.

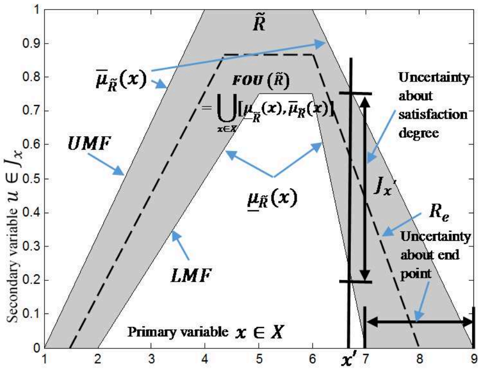

Figure 3. A type-2 fuzzy set based possibility distribution.

Definition 5. An ***embedded fuzzy set*** R_e is a T1 FS that is located inside the FOU of $\tilde{R}$.

4. ELASTICITY REASONING USING TYPE-2 FUZZY LOGIC SYSTEMS

In this section, we develop an IT2 FLS to enable the elasticity reasoning in cloud-based software, in which elasticity rules are based on a data collection from a group of technical stakeholders. As we discussed in Section 2.3, we chose to develop a fuzzy controller to give the stakeholders of cloud-based applications more flexibility to accommodate their thoughts in a qualitative manner.

4.1 Autonomous Control of Elasticity

As depicted in Figure 1, a cloud-based *elastic system* comprises three parts: 1) a cloud-based application, 2) a cloud platform, 3) an elasticity controller. The elasticity controller 1) Monitor the application and the environment. 2) Analyze the data and detect any violations. 3) Plan corrective actions in terms of adding resources or removing existing unutilized ones. 4) Execute the plan according to a specific platform. 5) Use or update a shared Knowledge. This is known as MAPE-K [25] loop named after its phases.

The monitoring is usually facilitated through the cloud platforms or third party solutions. For example, Amazon CloudWatch [22] provides monitoring for applications run on Amazon's cloud platform. The execution is facilitated through the cloud platform APIs and runtime configurability of the application. The elasticity reasoning process, P, is typically consisted of two steps: (i) processing a time-series runtime data collected through monitoring (see Section 5.1), and (ii) decision-making about the elasticity action (see Section 4.2). Once a specific situation $s \in S$ is detected, the reasoning mechanism chooses an action $ea \in EA$ delineated as:

$$P: S \rightarrow EA \tag{4.1}$$

The notion of the reasoner here generalizes a broader domain of analysis and planning altogether.

4.2 Overview of Elasticity Reasoning

The elasticity reasoning process, discussed in Section 4.1, is realized in this work using IT2-FLS. Figure 4 shows an elastic system within which the reasoning process is replaced with an IT2-FLS. The reference model that we borrowed is FORMS [31]. In this model, the *base-level* cloud-based software is under the control of *meta-level* auto-scaler. In this paper, we exemplify a SaaS (see Section 2.1), which is scaled by RobusT2Scale. In the meta-level, we realized the IBM MAPE-K [25]. Users use the functionalities via different devices, stakeholders specify policies and cloud platforms facilitate resource provisioning. In the remainder, we describe a method for designing the elasticity reasoning that operates at the heart of elasticity mechanism.

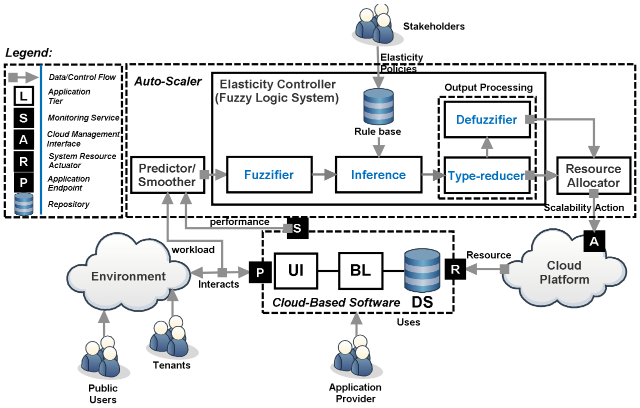

Figure 4. Overview of RobusT2Scale.

4.3 Extracting Elasticity Knowledge

In FLSs, the rule base and the membership functions associated with the variables in the rules are designed either by data collection from system behavior or by human experience [32]. In this work, human expertise have been considered to design the fuzzy sets and rules of the controller responsible for handling the elasticity reasoning. One of the reasons behind this choice was the inabilities of data-driven approaches to work under unforeseen situations. The most prominent capability of IT2-FLSs is the possibility of systematic collection of knowledge from different experts.

A fuzzy knowledge base (also called rule base as in Figure 4) holds the knowledge of how to best scale the target system in terms of a set of linguistic rules (e.g., rule (1.1)). In an if-then rule, the antecedent is composed of a number of sensed variables, and the consequent is composed of a number of control variables [29]. To construct a fuzzy knowledge base, the rules are systematically obtained from the stakeholders (e.g., architects or administrators). For instance, administrators employ subconsciously a set of if-then rules to manage the amount of resources a system needs to maintain acceptable level of user experience. Here, we present a technique for extracting elasticity knowledge from a group of experts. We also used the guidelines in [33] for data extraction.

In the running example, linguistic variable representing the value of workload were divided into five levels: *very low* (VL), *low* (L), *medium* (M), *high* (H), and *very high* (VH). Similarly, linguistic variable representing the value of response-time were divided into five levels: *instantaneous* (I), *fast* (F), *medium* (M), *slow* (S), *very slow* (VS). The consequent was divided into number of nodes that are added or removed. In this paper, for presentation purposes, we only consider five possible options from -2 to +2 nodes. To design the fuzzy rules, we collected the required data by performing a data collection among 10 experts in cloud computing. We used the following questions to extract knowledge from experts:

IF (the workload is *high*, AND the response time is *slow*), THEN (add/remove … node instances). (4.2)

These experts were asked to determine a consequent using an integer from $[-2,2]$. As we expected, different experts chose different number of node instances for the same questions. The questions and responds are summarized in Table 1. In order to reduce the threat of ordering effects, we reordered the questions.

We also asked the experts to locate an interval for each linguistic label for workload and response-time in $[0,100]$. For the labels, we received 10 different intervals from the 10 experts. We then calculated the mean and deviations of the two ends in Table 2.

Table 1. Questions for elasticity policies and expert responses.

Rule (l)	Antecedents		Consequent					c^{l}_{avg}
	Workload	Response-time	-2	-1	0	1	2	
1	Very low	Instantaneous	7	2	1	0	0	-1.6
2	Very low	Fast	5	4	1	0	0	-1.4
3	Very low	Medium	0	2	6	2	0	0
4	Very low	Slow	0	0	4	6	0	0.6
5	Very low	Very slow	0	0	0	6	4	1.4
6	Low	Instantaneous	5	3	2	0	0	-1.3
7	Low	Fast	2	7	1	0	0	-1.1
8	Low	Medium	0	1	5	3	1	0.4
9	Low	Slow	0	0	1	8	1	1
10	Low	Very slow	0	0	0	4	6	1.6
11	Medium	Instantaneous	6	4	0	0	0	-1.6
12	Medium	Fast	2	5	3	0	0	-0.9
13	Medium	Medium	0	0	5	4	1	0.6
14	Medium	Slow	0	0	1	7	2	1.1
15	Medium	Very slow	0	0	1	3	6	1.5
16	High	Instantaneous	8	2	0	0	0	-1.8
17	High	Fast	4	6	0	0	0	-1.4
18	High	Medium	0	1	5	3	1	0.4
19	High	Slow	0	0	1	7	2	1.1
20	High	Very slow	0	0	0	6	4	1.4
21	Very high	Instantaneous	9	1	0	0	0	-1.9
22	Very high	Fast	3	6	1	0	0	-1.2
23	Very high	Medium	0	1	4	4	1	0.5
24	Very high	Slow	0	0	1	8	1	1
25	Very high	Very slow	0	0	0	4	6	1.6

Table 2. Data regarding workload and response-time labels.

Linguistic		Means		Standard Deviations	
		Start (a)	End (b)	Start (σ_a)	End (σ_b)
Workload	Very low	0	27	0	8.23
	Low	22	41.5	7.15	7.09
	Medium	36.5	64	5.80	3.94
	High	61	82.5	4.59	6.77
	Very high	78	100	6.32	0
Response-time	Instantaneous	0	7.2	0	5.20
	Fast	6.1	20	4.07	5.27
	Medium	18.2	41.5	5.59	8.51
	Slow	38.5	63.5	7.09	9.44
	Very slow	60	100	7.82	0

4.4 Defining Membership Functions

Sensors measure the input values to the controller. Their conversion to fuzzy values is realized by MFs. In this section, we show how to derive appropriate MFs based on the data extracted in Section 4.3. We used the guidelines in [34] [35] in order to construct the MFs.

As illustrated in Figure 5 and Figure 6, we used trapezoidal MFs to represent "Very low" ("Instantaneous") and "Very high" ("Very slow"), and triangular MFs to represent "Low" ("Fast"), "Medium" and "High" ("Slow"). Let a and b with standard deviations σ_a and σ_b respectively be the mean values of the interval end-points of the linguistic labels (cf. Table 2). For "Low", "Medium" and "High" label, the triangular T1 MF is then constructed by connecting: $l = (a - \sigma_a, 0), m = ((a+b)/2, 1), r = (b + \sigma_b, 0)$. Accordingly, for "Very low" and "Very high" labels, the associated trapezoidal MFs can be constructed by connecting: $(a - \sigma_a, 0), (a, 1), (b, 1), (b + \sigma_b, 0)$, see dashed lines in Figure 5 and Figure 6. As it is indicated by the standard deviations in Table 2, there are uncertainties associated with the ends and the locations of the MFs. For instance, one may imagine a triangular T1 MF in:$l' = (a - 0.3 * \sigma_a, 0), m = ((a+b)/2, 1), r' = (b + 0.4 * \sigma_b, 0)$. These uncertainties cannot be captured by T1 fuzzy MFs. However, in IT2 MFs, the footprint of uncertainty (i.e., FOU in **Definition 3**) can be obtained by the UMF and LMF (**Definition 4**) for each linguistics. A blurring parameter $0 \le \alpha \le 1$ can determine the FOU (see Table 3).

Table 3. Locations of the main points of IT2 MFs.

Triangular	Trapezoidal
$l_{UMF} = (a - (1+\alpha) * \sigma_a, 0)$ $m_{UMF} = ((a+b)/2, 1)$ $r_{UMF} = (b + (1+\alpha) * \sigma_b, 0)$ $l_{LMF} = (a - (1-\alpha) * \sigma_a, 0)$ $m_{LMF} = ((a+b)/2, 1)$ $r_{LMF} = (b + (1-\alpha) * \sigma_b, 0)$	$ll_{UMF} = (a - (1+\alpha) * \sigma_a, 0)$ $ul_{UMF} = (a - \alpha\sigma_a, 1)$ $ur_{UMF} = (b + \alpha\sigma_b, 1)$ $lr_{UMF} = (b + (1+\alpha) * \sigma_b, 0)$ $ll_{LMF} = (a - (1-\alpha) * \sigma_a, 0)$ $ul_{LMF} = (a + \alpha\sigma_a, 1)$ $ur_{LMF} = (b - \alpha\sigma_b, 1)$ $lr_{LMF} = (b + (1-\alpha) * \sigma_b, 0)$

Here, we use $\alpha = 0.5$. Parameter $\alpha = 0$ reduces IT2 MFs to a T1 MFs, while parameter $\alpha = 1$ makes FSs with the widest FOUs.

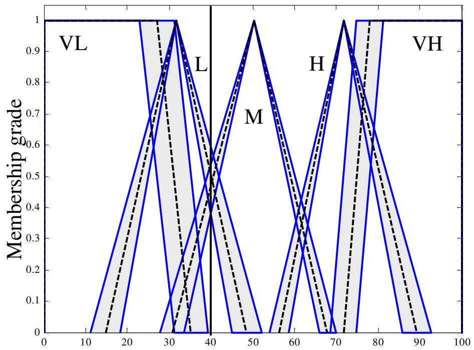

Figure 5. IT2 MFs of the workload labels.

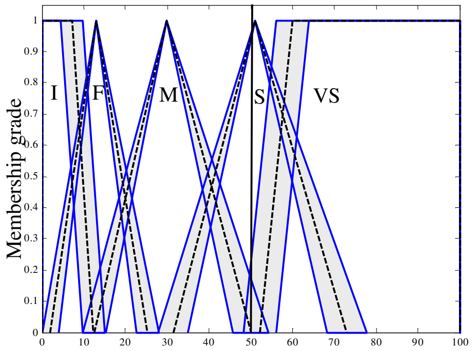

Figure 6. IT2 MFs of the response-time labels.

4.5 Basics of the Fuzzy Elasticity Controller

Having constructed the IT2 FLS with the MFs and the set of rules, the controller can then start controlling the elasticity reasoning on behalf of stakeholders. The designed controller works as the following (see Figure 4): (1) the inputs comprising the workload as well as the response time are first fuzzified. (2) Then the fuzzified input activates the inference engine to produce output IT2 FSs. (3) Decisions made by fuzzy inference are in the form of fuzzy values, which cannot be directly used. The outputs are then processed by a type-reducer, which combines the output sets and then calculate the center-of-set (**Definition 7**). (4) The type reduced FSs are T1 fuzzy sets that needs to be defuzzified to determine the nodes. (5) It then fed to the resource allocator to enact the change.

First, we must specify how the numeric inputs $u_i \in U_i$ are converted to fuzzy sets (a process called "fuzzification" [28]) so that they can be used by the FLS. In this paper, we use singleton:

$$\mu_{\tilde{R}_i} = \begin{cases} 1 & x = u_i \\ 0 & otherwise \end{cases} \qquad (4.3)$$

For defuzzification, we use the notion of centroid [36].

Definition 6. The ***centroid*** of a IT2 FS $\tilde{R}$ is the union of the centroids of all its embedded T1 fuzzy sets R_e (**Definition 5**):

$$C_{\tilde{R}} \equiv \bigcup_{\forall R_e} c(R_e) = [c_l(\tilde{R}), c_r(\tilde{R})] \qquad (4.4)$$

The type-reducer that we use here is center-of-sets [36].

Definition 7. The ***center-of-set type reduction*** is computed as:

$$Y_{cos} = \bigcup_{\substack{f^l \in F^l \\ y^l \in C_{\tilde{G}^l}}} \frac{\sum_{l=1}^{N} f^l \times y^l}{\sum_{l=1}^{N} f^l} = [y_l, y_r] \qquad (4.5)$$

, where $f^l \in F^l$ is the firing degree of rule l and $y^l \in C_{\tilde{G}^l}$ is the centroid of the IT2 FS $\tilde{G}^l$ (cf. **Definition 6**).

$c_l(\tilde{R}), c_r(\tilde{R})$ and y_l, y_r are computed by the KM algorithm [36].

4.6 Elasticity Reasoning as the Key Process

The rules in this work are in the form of multi-input single-output. Because the preferences of stakeholders may not be similar, many elasticity rules in the mind of stakeholders may be conflicting, i.e. rules with the same antecedent but different consequent values. In this step, rules with the same if part are combined into a single rule. For each response that we received from the stakeholders, we have:

$$R^l\!: IF\ x_1\ is\ F_1^l\ and \dots and\ x_p\ is\ F_p^l, THEN\ y\ is\ y^{(t_u^l)} \qquad (4.6)$$

, where t_u^l is the index for the responses. In order to combine these conflicting rules, we used the average of all the responses for each rule and used this as the centroid of the rule consequent. Note that the rule consequents are IT2 FSs, however, when the type reduction in **Definition 1** is used, these IT2 FSs are replaced by their centroids, so we represent them as intervals $[\underline{y}^n, \overline{y}^n]$ or crisp values when $\underline{y}^n = \overline{y}^n$. This leads to rules with the following form:

R^l: IF (the workload (x_1) is $\tilde{F}_{i_1}$, AND the response-time (x_2) is $\tilde{G}_{i_2}$), THEN (add/remove c_{avg}^l instances). (4.7)

$$c_{avg}^l = \frac{\sum_{u=1}^{N_l} w_u^l \times C}{\sum_{u=1}^{N_l} w_u^l} \qquad (4.8)$$

, here C is the value of associated consequent, i.e., an integer between $[-2,2]$, and w_u^l is the weight associated with uth consequent of the lth rule (cf. Table 1). Therefore, each c_{avg}^l (see Table 1) can be computed with the Equation (4.8). For instance, c_{avg}^{12}, which is associated to rule number 12 is calculated as:

$$c_{avg}^{12} = \frac{2 \times -2 + 5 \times -1 + 3 \times 0 + 0 \times 1 + 0 \times 2}{2 + 5 + 3 + 0 + 0} = -0.9 \qquad (4.9)$$

In an example, we now discuss the details of the elasticity reasoning process according to Figure 4. Let us imagine the normalized values regarding the workload and response-time are $x_1 = 40$ $x_2 = 50$ respectively, see the solid lines in Figure 5 and Figure 6. For $x_1 = 40$, two IT2 FSs regarding the linguistics $\tilde{F}_2 = Low$ and $\tilde{F}_3 = Medium$ with the degrees $[0.3797, 0.5954]$ and $[0.3844, 0.5434]$ are fired. Similarly, for $x_2 = 50$, three IT2 FSs regarding the linguistics $\tilde{G}_3 = Medium$, $\tilde{G}_4 = Slow$, and $\tilde{G}_5 = Very\ slow$ with the firing degrees $[0, 0.1749]$, $[0.9377, 0.9568]$ and $[0, 0.2212]$ are fired. Intuitively, the lower and upper values of the intervals can be computed by finding the y-intercept of the solid lines in the figures respectively with the LMF and the UMF of the crossed FSs. As a result, six rules are fired: $R^8\!: (\tilde{F}_2, \tilde{G}_3), R^9\!: (\tilde{F}_2, \tilde{G}_4), R^{10}\!: (\tilde{F}_2, \tilde{G}_5), R^{13}\!: (\tilde{F}_3, \tilde{G}_3), R^{14}\!: (\tilde{F}_3, \tilde{G}_4), R^{15}\!: (\tilde{F}_3, \tilde{G}_5)$, see Table 1. The firing intervals are computed using meet operation [27]. For instance, the firing interval associated to the rule R^9 is:

$$\begin{aligned} \underline{f}^9 &= \underline{\mu}_{\tilde{F}_2^9}(x_1') \otimes \underline{\mu}_{\tilde{G}_4^9}(x_2') = 0.3797 \times 0.9377 = 0.3560 \\ \overline{f}^9 &= \overline{\mu}_{\tilde{F}_2^9}(x_1') \otimes \overline{\mu}_{\tilde{G}_4^9}(x_2') = 0.5954 \times 0.9568 = 0.5697 \end{aligned} \qquad (4.10)$$

The output can be obtained using the center-of-set (**Definition 7**):

$$\begin{aligned} Y_l(40,50) &= [y_l(40,50), y_r(40,50)] \\ &= [0.9296, 1.1809] \end{aligned} \qquad (4.11)$$

The defuzzified output can be calculated:

$$Y(40,50) = \frac{0.9296 + 1.1809}{2} = 1.0553 \quad (4.12)$$

Similarly, we can compute $Y(x_1, x_2)$ for all the possible normalized values of the input parameters (i.e., $x_1 \in [0,100]$, $x_2 \in [0,100]$). The resulting hyper-surface $Y(x_1, x_2)$ is shown in Figure 7. Note that $Y(x_1, x_2) \subseteq [-2,2]$ for any (x_1, x_2).

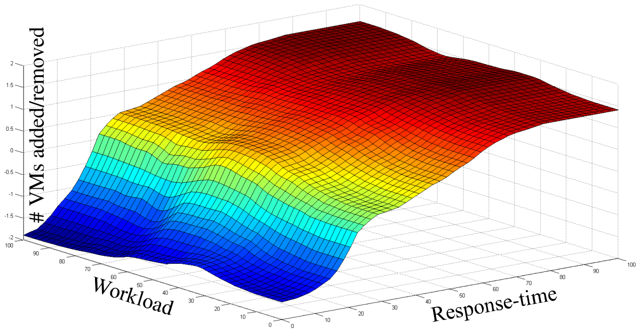

Figure 7. Output of the IT2 FLS for elasticity reasoning.

4.7 The Benefits of Using Type-2 over Type-1

As discussed in Section 4.1, elasticity reasoning is the process of finding a solution for a decision-making problem - choosing an appropriate number of nodes given environmental and system state. As shown in Section 4.6, the output of IT2 FLS is a boundary rather than a hard-threshold as in T1 FLS [29] [35]. Therefore, the decision for nodes can be more flexible providing a boundary. For instance, if the system requires a high performance, the decision can be made based on the upper boundary, i.e. $\lceil y_r(40,50) = 1.1809 \rceil$. As a result, two VMs will be added. If the system requires saving cost, the decision can be made based on the lower boundary, i.e. $\lfloor y_l(40,50) = 0.9296 \rfloor$. No new nodes would then be added. In addition, if the system needs to achieve a compromise in user experience and cost, the decision can be made based on any value in the boundary. This flexibility and the ability to handle conflicting rules (see Section 4.6) are the key benefits of T2 FLSs over T1 counterparts that motivated us to choose it for elasticity reasoning.

5. REALIZING THE AUTO-SCALER

In Section 4, we described the details of the elasticity reasoning that acts as the heart of RobusT2Scale for making the scaling decisions. In this section, we describe the other modules involves in RobusT2Scale as depicted in Figure 4. First, we describe the prediction module for reasoning input preparations, and then we detail the resource allocator as the actuator of RobusT2Scale. Finally, we describe the details of the integration of these modules.

5.1 Parameter Prediction and Smoothing

In historical data corresponding to workload measurements, there are typically high variability, which makes resource allocation at small time-scales unfeasible. As we discussed in Section 2.2 (see challenge 1), the startup time of the VMs are not instant and among the cloud providers, it varies between 60 to 600 seconds [24] but the workload contains many short duration spikes. On the other hand, the elasticity is only effective if node instances can be ready to use when they are needed to serve the workload. Instead of making decisions based on short duration spikes, the elasticity controller needs to identify workload variations that will persist for long enough periods in order to launch or terminate VMs. The term workload refers to a list of user requests and their arrival timestamp.

When an application starts running, a time-series forecasting technique is employed to estimate the workload at some future point in time. We use *double exponential smoothing* [37] because this model has the capability to smooth the inputs and predict the trend in historical data. This model takes the number of requests for application services at runtime and predict the future workload. On the other hand, for estimating response-time, we use *single exponential smoothing* [37] because for the oscillatory response-time, we do not need to predict the trend but a smoothed value.

Both the exponential smoothing techniques weight the history of the workload data by a series of exponentially decreasing factors. An exponential factor close to one gives a large weight to the first samples and rapidly makes old samples negligible. The specific formula for single exponential smoothing is:

$$s_t = \theta x_t + (1 - \theta) s_{t-1}, t > 0; s_0 = x_0 \quad (5.1)$$

Correspondingly, the formula for double exponential smoothing is:

$$\begin{aligned} s_t &= \beta x_t + (1 - \beta)(s_{t-1} + b_{t-1}) \\ b_t &= \gamma(s_t - s_{t-1}) + (1 - \gamma) b_{t-1};\ 0 < \theta, \beta, \gamma < 1 \end{aligned} \quad (5.2)$$

, where x_t the raw data sequence and s_t is the output of the techniques and θ, β, γ are the smoothing factors. Note the number of data points here depends on the control loop intervals and the frequency of the performance counters retrievals in each loop.

5.2 Resource Allocation

The resource allocator (See Figure 4) communicates with the cloud management services to acquire or release node instances as indicated by the controller (see Sections 4.6). However, the when and how to apply the changes in the resources is determined by the resource allocator. In order to regulate such policies, we implemented specific features in resource allocator module.

In some cloud providers, the cost for VMs are calculated on *hourly basis* (e.g., Amazon EC2) and in some providers it is *proportional* to the exact time between acquiring the machine to the time it has been released (e.g., Microsoft Azure). For this reason, we implemented two different termination policies. One policy terminates a node instance as soon as it has been decided and the other only terminates a node instance if it has been running just below a multiple number of hours. In the meantime, these instances contribute to processing the workload, thus providing some extra capacity to handle short load spikes at no extra cost. However, in this work, we evaluated RobusT2Scale on Azure, thus we only make use of the first policy. Note Azure offers both platform as a service (PaaS) as well as infrastructure as a service (IaaS), but in the context of this work, we only employed PaaS services.

Another feature is the *cool-down period* (also called *inertia* or *calm* period [18]). The cool-down period prevents the resource allocator of making any changes to the system deployment on the cloud for certain amount of time. The motivation behind this is to avoid frequent creation and releasing of instances when the workload exhibits high variability, as this would have a negative impact on the cost [18]. We implemented this feature by putting a delay in each control loop. As a result, the reactive controller is only able to change the deployment if sufficient time between scaling action has passed and if the new instances' state is 'operational'.

We also enforce some *constraints* to the number of node instances in order to avoid excessive cost or to jeopardize user experience. Note the constraint rules always takes precedence over reactive rules that are used in the fuzzy elasticity reasoner, to ensure that the reactive rules cannot continue to add new node instances above a maximum threshold or remove instances below a minimum level.

5.3 Implementation Details of RobusT2Scale

We have described the behavior of the three key components of the RobusT2Scale, i.e., fuzzy elasticity controller, workload predictor and cloud resource allocator, in detail in Section 4 and 5.1 and 5.2 respectively. We now present the details of the coordination between these components to realize RobusT2Scale as depicted in Figure 8. We implemented the predictor and elasticity reasoner in Matlab 2013a and the resource allocator in C#.NET. We also integrated all modules with a coordinator controller. Since in the resource allocator and the coordinator we make use of RESTful API calls to the cloud platform, the choice of programming language is just a matter of preference.

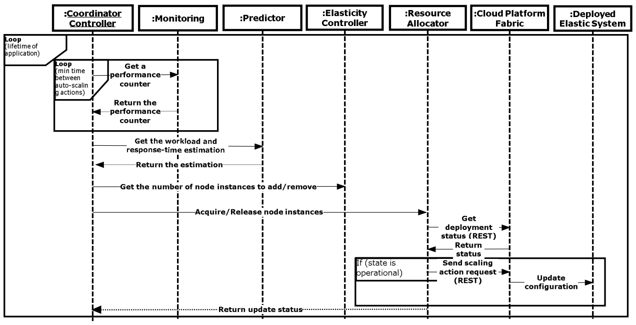
:Coordinator Controller
:Monitoring
:Predictor
:Elasticity Controller
:Resource Allocator
:Cloud Platform Fabric
:Deployed Elastic System
Loop (lifetime of application)
Loop (min time between auto-scalin g actions)
Get a performance counter
Return the performance counter
Get the workload and response-time estimation
Return the estimation
Get the number of node instances to add/remove
Acquire/Release node instances
Get deployment status (REST)
Return status
If (state is operational)
Send scaling action request (REST)
Update configuration
Return update status

Figure 8. Coordination between RobusT2Scale's components.

As depicted in Figure 8, for collecting the monitoring data, we used the performance counters in Windows Azure. We configured the system under test to retrieve the performance counters periodically each 1000 ms. The predictor then calculates the number of hits and response-time in each control loop and feed to the elasticity controller. The elasticity controller then calculate the number of new nodes and feed it to the resource allocator to decide when and how to acquire resources. The allocator then sends a REST request to Windows Azure management fabric. The fabric then changes the status of the application from 'running' to 'transitioning' until the new instances are up and running. However, meanwhile load balancer can route request to existing nodes. When the deployment status restores back to 'running', the allocator send another request in the next control loop otherwise new requests will be ignored.

6. EXPERIMENTAL EVALUATIONS

In this section, we present a number of experimental studies on RobusT2Scale to answer the following research questions:

- **RQ1**. *What is the accuracy of the employed estimation techniques and does the error of estimation vary across different workloads*?
- **RQ2**. *Is it effective for guaranteeing SLAs and minimizing cost*?
- **RQ3**. *Is it robust against measurement noises*?

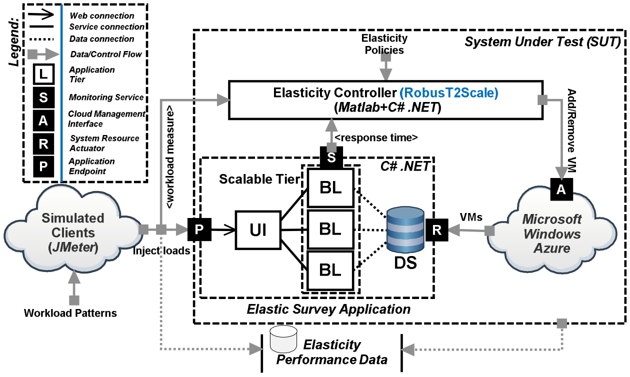

Figure 9. Overview of our experimental setting.

6.1 Experimental Setting

The architecture of our experimental setup is depicted in Figure 9. The client side is JMeter, which generate workload based on our predefined patterns. In our case, the server side is the System Under Test (SUT), which is a three tier cloud-based application controlled by RobusT2Scale. Here we defined test cases in which the number of users and their usage vary according to time-dependent patterns. A workload generated in this manner hits the SUT and triggers its controller. The controller ensure that the application remains elastic. Here we followed the guidelines of cloud testing, e.g. [13].

Typically, scalability is concerned with variances that are large enough to warrant a scaling action. In this work, we injected different patterns of workloads, most of which are drawn from real world workloads (e.g. [38], similar patterns are also used in [39]), to explore the platform's elasticity behavior for a range of demand patterns. In our measurements, we use a set of six different workloads – see Figure 11. Across time, some workloads show recurring cycles of growth and decrease, such as an hourly news cycle. Others have a single burst, such as during a special event. Further, we scale the duration of the traces to 1 hour. We evaluate RobusT2Scale against the full set of workloads (see Table 5). For the experiments, we also considered some other treatments. Since JMeter consumes significant amount of resources, we ran an instance on a dedicated machine. We ran the SUT on Azure VMs. VMs were located in the same availability zone in Ireland; see the deployment details in Table 4.

Table 4. Deployment details of our experimental setting.

Experimental Deployment Units	Clients	Elastic Application			Elasticity Controller
	JMeter	**UI**	**BL (Scalable)**	**DS**	**RobusT2Scale**
Specification	Desktop, Intel Core i7 CPU, 2.8GHz, 12 GB	**1** Small (A1) Azure VM	**2-6** Small (A1) Azure VMs	**1** Small (A1) Azure VM	**1** Small (A1) Azure VM

6.2 Workload Estimation Accuracy (RQ1)

In order to evaluate the accuracy of the adopted estimation technique, we simulated different workloads and measured the error of estimation by root relative squared error (RRSE). Figure 10 shows a sample data and different estimations by changing the parameters of the model. It is evident that the estimations with different parameters results in different level of prediction accuracy. For this sample, the estimation with $\beta = 0.27, \gamma = 0.94$ is more accurate than the other two estimations.

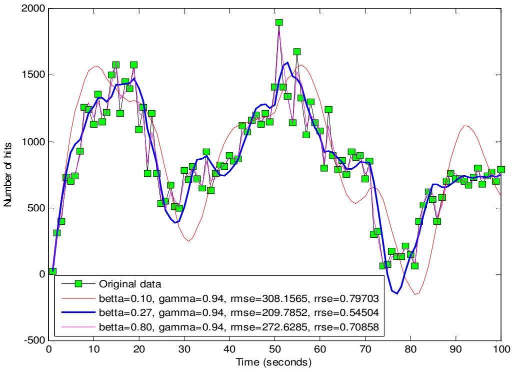

Figure 10. Predicted vs. actual workload.

We also evaluated the accuracy of the prediction techniques for different workload patterns. As it is depicted in Figure 11, for different patterns (i.e., big spike, etc.), the estimator shows different estimation errors. For three patterns, i.e., 'slowly varying', 'dual phase', 'steep tri phase', the relative error and variations are quite low. The 'large variation' shows the large mean of error and 'big spike' and 'quickly varying' demonstrate the largest variations.

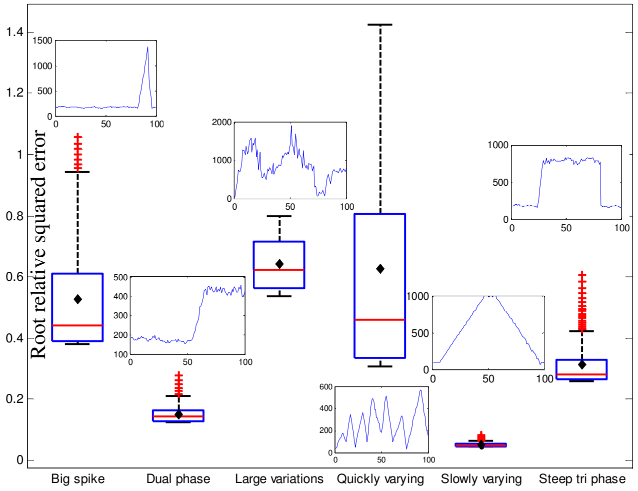

Figure 11. Estimation errors w.r.t. workload patterns.

6.3 Effectiveness of RobusT2Scale (RQ2)

As a benchmark for measuring the effectiveness of RobusT2Scale, we consider (1) 95th percentile of response time (rt_{95}), which represent our SLA and (2) the weighted average number of node instances acquired over time ($\overline{vm}$), which determines the cost of ownership. These criteria cover the three main aspects of elasticity comprising scalability, cost and time efficiency. The goal is to meet the response time SLA, here we assume $rt_{95} = 600ms$, while keeping $\overline{vm}$ as low as possible. The drop in $\overline{vm}$ represents the potential capacity to be released back to the cloud to save on costs. To evaluate the effectiveness of RobusT2Scale, we compared our approach with two provisioning policies: *over-provisioning* (here is 6) and *under-provisioning* (here is 2). A summary of the results is shown in Table 5. In comparison with over provisioning policy, RobusT2Scale has acquired less nodes, saving as much as a factor of two in cost. In comparison with under provisioning policy, RobusT2Scale is significantly better in terms of rt_{95}, giving a cloud-based application a better chance to guarantee the SLAs.

Table 5. Comparison of the effectiveness of RobusT2Scale.

SUT	Criteria	Big spike	Dual phase	Large variations	Quickly varying	Slowly varying	Steep tri phase
with RobusT2Scale	$rt_{95\%}$	973ms	537ms	509ms	451ms	423ms	498ms
	$\overline{vm}$	3.2	3.8	5.1	5.3	3.7	3.9
with overprovisioning	$rt_{95\%}$	354ms	411ms	395ms	446ms	371ms	491ms
	$\overline{vm}$	6	6	6	6	6	6
with under provisioning	$rt_{95\%}$	1465ms	1832ms	1789ms	1594ms	1898ms	2194ms
	$\overline{vm}$	2	2	2	2	2	2

As seen in Table 5, the SUT with RobusT2Scale has not violated the response time SLA in all patterns of workloads except for the "big spike". The SUT with the overprovisioning has satisfied the SLA for all the patterns, however, by imposing a cost of up to a double amount (for 'big spike', but for the other patterns the difference is less) of what has been imposed by RobusT2Scale. The SLA is never met for the SUT with the under provisioning.

6.4 Robustness of RobusT2Scale (RQ3)

In Section 6.2, we showed that the utilized estimation approach, i.e. double exponential smoothing, contains unavoidable errors. In this paper, we have claimed that the RobusT2Scale are resilient against input noises, one of which is the estimation error. In this section, we provide some experimental evidences to support this claim.

In Section 6.2, we observed that the worst estimation error happens for 'large variation' and 'quickly varying' patterns and is less than 10% of the actual workload. As a result, we injected a white noise to the input measurement data (i.e., x_1, see Section 4.6) with an amplitude of 10%. We ran RMSE measurements for each levels of blurring, and for each measurement, we used 10,000 data items as input. Figure 12 shows RMSE values for the four different blurring values. We observed two interesting points. First, the error of control output produced by the elasticity controller is less than 0.1 for the blurring levels. Second, the error of control output is decreasing when we designed the controller with a higher blurring. As we discussed in Section 4.4, a higher blurring leads to a bigger FOU, which is a representative for the supporting levels of uncertainty (see **Definition 3**). Therefore, designer should make a choice in terms of the level of uncertainty that the controller can support. Note in some circumstances an overly wide FOU results in a performance degradations [33]. These observations provide enough evidence that RobusT2Scale is robust against input noise. This achievement is one of the important benefits of using IT2 FLS rather than T1 FLS for elasticity reasoning in cloud-based software, where uncertainty in terms of noise and events are prevalent [13].

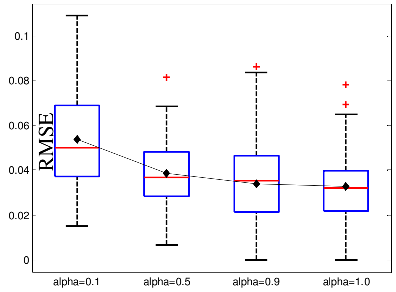

Figure 12. RMSEs of the controller with different blurrings.

6.5 Discussions and Threats to Validity

In the remainder of this section, we discuss the results, limitations, threats to validity of the findings, some insights and future work.

Independent elasticity controller. RobusT2Scale is independent from both the underlying cloud platform and specific cloud-based application (see Section 5.3), and allows cloud service providers (e.g., SaaS application owners) to easily integrate it with their own application. RobusT2Scale can be deployed either in the cloud, preferably in a separate node from the application, or on premise. Note that in this paper, we described a methodology to build such a controller based on data collection from users.

Few design parameters. The design of RobusT2Scale includes only few parameters: blurring parameter α (see Section 4.4), smoothing factors β, θ, γ (see Section 5.1), and cool-down period (see Section 5.2). Fortunately, all of the parameters only need to be determined once for an application to integrate with RobusT2Scale.

No need for offline training. The prediction techniques can be used without offline training because it takes advantage of online incremental learning. This reduces the upfront efforts required for configuring RobusT2Scale for building elastic applications. This along with the fact that RobusT2Scale requires few design parameters increases the chance for the adoption of this approach.

Rule explosion and computational complexity. Rule explosion is not a relevant concern in our approach to elasticity due to limited number of reasoning parameters as opposed to rule-based reasoning in self-adaptive software [16]. However, IT2 FLSs are effective in elimination of the rule explosion in comparison with T1 FLSs [29]. More importantly, the ability of the FOU to represent uncertainties enables the designer to cover input-output domains with fewer FSs leading to the reduction of the rules comparing with T1 FLS [33].

Limited workload patterns. In Section 6, we considered six different workload patterns. However, in production environments, workload may change in many unpredictable ways [40]. Evaluating RobusT2Scale under all scenarios is beyond the scope of this paper.

Evaluations with different application type. Our experimental evaluation is limited to a multi-tier application. Moreover, RobusT2Scale assumes that nodes are stateless. Fortunately, such architectural patterns have been promoted [23].

Evaluations with different cloud providers. Although the fuzzy reasoner and the adopted prediction techniques are independent of specific cloud providers, we tested RobusT2Scale only on Microsoft Azure. As a future work, we plan to extend our resource allocator with platforms such as Amazon EC2 and OpenStack [22].

Concentrating only on business tier. In this paper, we only focus on the business tier of the cloud-based application. However, we cannot claim that RobusT2Scale can be readily adopted to auto-scale the other tiers. Although in [18], authors argue that the auto-scaling approaches can be used in other tiers, in [8], authors argue that there are some very specific issues related to specific tiers.

7. RELATED WORK

In general, auto-scaling approaches can be characterized into three categories. *Reactive* techniques adjust the required resources based on demand. *Proactive* techniques anticipate the amount of resources. *Hybrid* approaches blends reactive and proactive techniques. For a more detail review, refer to [22] [41] [18] [12].

Reactive auto-scaling. Reactive techniques are popular in research and practice [41]. For instance, commercial solutions offered by several public cloud providers (e.g., Amazon and Microsoft), cloud platforms (e.g., OpenNabula) as well as third party tools (e.g., RightScale) utilize reactive rule-based methods. Threshold-based rules are popular (e.g., [19] [20] [7] [21]). However, it requires an extra effort for specifying metrics and parameters. Among the parameters, the upper and lower thresholds are the key. A complement to reactive rules is RightScale's auto-scaling algorithm [22]. It is a voting process whereby, if a majority of the nodes agrees that they need to scale up or down, that action is agreed. The main drawback of reactive approaches is their inability to anticipate the unexpected changes. This usually incurs higher cost. As opposed to existing work, in our approach, the thresholds can be set qualitatively. As a result, stakeholders do not need to set precise thresholds, which needs deep knowledge of the workload patterns and cause oscillations in the resources if specified incorrectly [18].

Proactive auto-scaling. In turn, the proactive approaches uses analytical models to anticipate workloads. Among others, time-series analysis, queuing models, machine learning, and control theory are popular techniques. Time-series analysis (e.g., [11]) predict future values of a parameter, based on its historical data. However, the prediction accuracy highly depends on the number observations and the interval [18]. Reinforcement learning (e.g., [42]) enable learning elasticity policies from observations. However, it requires long learning, which is only applicable for stable workloads. Queuing theory (e.g., [43]) imposes restrictive assumptions and as they are intended for stationary scenarios, the models need to be recalculated when the conditions change [18]. Finally, controllers (e.g., [8] [7]) are able to maintain the output at the desired level, depending on the input. However, setting a wrong gain parameter may cause oscillations. In some other approaches like [39], future workload prediction is relegated due to dynamics.

Hybrid auto-scaling. Hybrid approaches combine reactive and proactive techniques to determine when to acquire resources over short and long time scales respectively (e.g., [43]). In this category, some approaches use predictive techniques for releasing resources and reactive techniques for acquiring resources (e.g., [44]). RobusT2Scale is considered as a hybrid approach as we combine both proactive time-series analysis and reactive fuzzy controller.

Vertical vs. horizontal scaling. In practice, existing solutions for auto-scaling enable horizontal scaling, i.e., acquiring or realizing node instances, while vertical scaling, i.e., increasing computing power of node instances, is not considered. It has been attributed to impossibility of changing the size of nodes at the hypervisors level [22] [9]. RobusT2Scale enables horizontal scaling.

White box vs. black box approaches. Unlike our approach, some elasticity controllers are based on black-box surrogate models of the system that evolve over time, see [12]. These controllers use machine learning to predict system performance under different usage. In the training phase, the controller correlates the configurations with monitoring variables, and builds a model for each dimension of the system's behavior. In the control phase, the controller uses these models to manage resources while continuously learning from system behavior.

Fuzzy control solutions. Xu et al. [45] proposed an elasticity controller that applies fuzzy logic techniques to learn the relationship between workload, resources and performance and then manages the resource allocation based on the learned fuzzy rules. Similar type of fuzzy controller has been offered in [46] with adaptive output amplification and flexible rule selection. However, both approaches are based on T1 FLS and have not addressed the challenges posed by uncertainty.

Control-theoretical solutions. There are some approaches based on control theory (e.g., [19] [47]) that can enhance cloud-based applications with the capability to adjust their resources based on changing environmental conditions. These approaches typically synthesize an elasticity controller to automatically decide when to activate some optional features. The benefit of such approach is that they allow guaranteeing some specific desirable properties. Although such controllers are resilient against stationary noises but they are proved to be robust against non-stationary uncertainties.

Concluding remarks. The literature on auto-scaling is abundant. However, our approach has three distinguishing benefits. Firstly, it enables qualitative rule specification through a well-defined methodology. It can also handle measurement noises and robustly adjust resources. Additionally, conflicting rules can be handled within our approach. This can prepare tradeoffs to decide an appropriate action. To the best of our knowledge, this is the first work based on type-2 fuzzy controllers for the problem of dynamic resource provisioning in the cloud.

8. CONCLUSION AND FUTURE WORK

This paper tackled the problem of dynamic allocation of resources for cloud-based applications facing unpredictable workloads to decrease cost of ownership without violating SLAs. We proposed a hybrid elasticity controller to adjust the required resources when the application is running. The notable novelty of our approach is to enable qualitative specification of elasticity rules. A secondary benefit is that the elasticity controller can handle conflicting rules. The proposed controller is also robust against noisy measurements. We envision some future work as 1) integration of RobusT2Scale with comprehensive set of platforms, 2) systematic comparison with other auto-scaling approaches in controlled experiments.

9. ACKNOWLEDGMENTS

The research work described in this paper was partly supported by the Irish Centre for Cloud Computing and Commerce (IC4), a national technology center funded by Enterprise Ireland, and, in part, by Science Foundation Ireland grant 10/CE/I1855 to Lero.

10. REFERENCES

[1] M. Armbrust and e. al., "A view of cloud computing," *Communications of the ACM,* vol. 53, no. 4, pp. 50-58, 2010.

[2] P. Jamshidi, A. Ahmad and C. Pahl, "Cloud Migration Research: A Systematic Review," *IEEE Transactions on Cloud Computing,* 2013.

[3] N. R. Herbst, S. Kounev and R. Reussner, "Elasticity in Cloud Computing: What It Is, and What It Is Not," in *ICAC*, 2013.

[4] S. Islam, K. Lee, A. Fekete and A. Liu, "How a consumer can measure elasticity for cloud platforms," in *ICPE*, 2012.

[5] "Animoto's Facebook Scale-Up," [Online]. Available: http://tinyurl.com/qdk7om4.

[6] G. Linden, "Make Data Useful," Amazon, 2009.

[7] H. C. Lim, S. Babu, J. S. Chase and S. S. Parekh, "Automated control in cloud computing: challenges and opportunities," in *ACDC*, 2009.

[8] H. C. Lim, S. Babu and J. S. Chase, "Automated control for elastic storage," in *ICAC*, 2010.

[9] L. M. Vaquero, L. Rodero-Merino and R. Buyya, "Dynamically scaling applications in the cloud," *Computer Communication Review,* 2011.

[10] H. Ghanbari, B. Simmons, M. Litoiu, C. Barna and G. Iszlai, "Optimal autoscaling in a IaaS cloud," in *ICAC*, 2012.

[11] Z. Shen, S. Subbiah, X. Gu and J. Wilkes, "Cloudscale: elastic resource scaling for multi-tenant cloud systems," in *SCC*, 2011.

[12] A. Gambi, G. Toffetti and M. Pezzè, "Assurance of self-adaptive controllers for the cloud," in *Assurances for Self-Adaptive Systems*, 2013.

[13] A. Gambi, W. Hummer, H.-L. Truong and S. Dustdar, "Testing Elastic Computing Systems," *Internet Computing,* 2013.

[14] N. N. Karnik, J. M. Mendel and Q. Liang, "Type-2 fuzzy logic systems," *IEEE Transactions on Fuzzy Systems,,* vol. 7, no. 6, pp. 643-658, 1999.

[15] D. Betts, Developing Multi-tenant Applications for the Cloud, Microsoft, 2012.

[16] D. Garlan and e. al., "Rainbow: Architecture-based self-adaptation with reusable infrastructure," *Computer,* vol. 37, no. 10, pp. 46-54, 2004.

[17] H. Ghanbari, B. Simmons, M. Litoiu and G. Iszlai, "Exploring alternative approaches to implement an elasticity policy," in *ICCC*, 2011.

[18] T. Lorido-Botrán, J. Miguel-Alonso and J. A. Lozano, "Auto-scaling Techniques for Elastic Applications in Cloud Environments," University of Basque Country, Tech. Rep. EHU-KAT-IK-09-12, 2012.

[19] X. Dutreilh, N. Rivierre, A. Moreau, J. Malenfant and I. Truck, "From data center resource allocation to control theory and back," in *ICCC*, 2010.

[20] M. Maurer, I. Brandic and R. Sakellariou, "Enacting SLAs in clouds using rules," in *Euro-Par*, 2011.

[21] P. Marshall, K. Keahey and T. Freeman, "Elastic site: Using clouds to elastically extend site resources," in *ICCCGC*, 2010.

[22] E. Caron, L. Rodero-Merino, F. Desprez and A. Muresan, "Auto-scaling, load balancing and monitoring in commercial and open-source clouds," 2012.

[23] B. Wilder, Cloud Architecture Patterns: Using Microsoft Azure, O'Reilly, 2012.

[24] M. Mao and M. Humphrey, "A performance study on the vm startup time in the cloud," in *CLOUD*, 2012.

[25] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer,* vol. 36, no. 1, p. 41–50, 2003.

[26] L. A. Zadeh, "The concept of a linguistic variable and its application to approximate reasoning—I," *IS,* 1975.

[27] J. M. Mendel, "Type-2 fuzzy sets and systems: an overview," *Computational Intelligence Magazine,* vol. 2, no. 1, 2007.

[28] J. M. Mendel, R. I. John and F. Liu, "Interval type-2 fuzzy logic systems made simple," *IEEE TFS,* 2006.

[29] D. Wu, "On the fundamental differences between Type-1 and interval Type-2 fuzzy logic controllers," *IEEE Transactions on Fuzzy Systems,* 2012.

[30] J. M. Mendel, H. Hagras and R. I. John, "Standard background material about interval type-2 fuzzy logic systems that can be used by all authors," *CIS,* 2010.

[31] D. Weyns, S. Malek and J. Andersson, "FORMS: a formal reference model for self-adaptation," in *ICAC*, 2010.

[32] Q. Liang and J. M. Mendel, "Interval type-2 fuzzy logic systems: theory and design," *IEEE TFS,* vol. 8, no. 5, pp. 535-550, 2000.

[33] J. M. Mendel, Uncertain rule-based fuzzy logic system: introduction and new directions, Prentice Hall, 2001.

[34] J. M. Mendel, "Computing with words, when words can mean different things to different people," in *ICSC Sympusium Fuzzy Logic Application*, 1999.

[35] Q. Liang, N. N. Karnik and J. M. Mendel, "Connection admission control in ATM networks using survey-based type-2 fuzzy logic systems," *Transactions on Applications and Systems, Man, and Cybernetics,* 2000.

[36] N. N. Karnik and J. M. Mendel, "Centroid of a type-2 fuzzy set," *Information Sciences,* vol. 132, no. 1, pp. 195-220, 2001.

[37] P. S. Kalekar, "Time series forecasting using Holt-Winters exponential smoothing," Kanwal Rekhi School of Information Technology, 2004.

[38] "Anonymized access logs," National Laboratory for Applied Network Research, 2001. [Online]. Available: ftp://ftp.ircache.net/Traces/.

[39] A. Gandhi, M. Harchol and R. Raghunathan, "Autoscale: Dynamic, robust capacity management for multi-tier data centers," *TOCS,* 2012.

[40] P. Bodik, A. Fox, M. J. Franklin, M. I. Jordan and a. D. A. Patterson, "Characterizing, modeling, and generating workload spikes for stateful services," in *SCC*, 2010.

[41] G. Galante and L. C. E. d. Bona, "A survey on cloud computing elasticity," in *UCC*, 2012.

[42] E. Barrett, E. Howley and J. Duggan, "Applying reinforcement learning towards automating resource allocation and application scalability in the cloud," *Concurrency and Computation: Practice and Experience,* 2012.

[43] B. Urgaonkar, P. Shenoy, A. Chandra, P. Goyal and T. Wood, "Agile dynamic provisioning of multi-tier internet applications," *ACM TAAS,* 2008.

[44] A. Ali-Eldin, J. Tordsson and E. Elmroth, "An adaptive hybrid elasticity controller for cloud infrastructures," in *NOMS*, 2012.

[45] J. Xu, M. Zhao, J. Fortes, R. Carpenter and M. Yousif, "On the use of fuzzy modeling in virtualized data center management," in *ICAC*, 2007.

[46] J. Rao, Y. Wei, J. Gong and C. Z. Xu, "DynaQoS: model-free self-tuning fuzzy control of virtualized resources for QoS provisioning," in *IWQoS*, 2011.

[47] C. Klein, M. Maggio, K. E. Årzén and F. Hernández-Rodriguez, "Brownout: Building More Robust Cloud Applications," in *ICSE*, 2014.