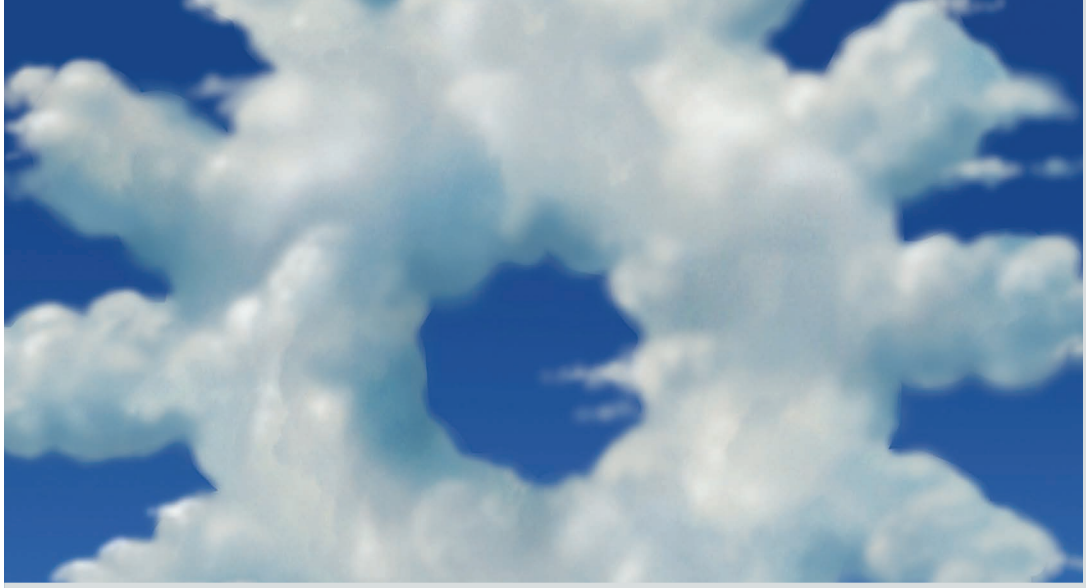

# Managing Uncertainty in Autonomic Cloud Elasticity Controllers

**Pooyan Jamshidi,** Imperial College London
**Claus Pahl,** Free University of Bozen-Bolzano
**Nabor C. Mendonça,** University of Fortaleza

*Following a control-theoretic approach, the authors integrate a fuzzy cloud controller with an online learning mechanism to achieve a framework that can cope with various sources of uncertainty in the cloud.*

The challenge of building autonomous elastic systems involves the adjustment of computing resources along with load variations without the need for human intervention. Automated cloud-based scalability (that is, *autoscaling*) is a recent advancement toward creating full-fledged elastic systems.[1] Autoscaling an application, typically in commercial solutions, involves specifying threshold-based rules to implement elasticity policies

 2325-6095/16/$33.00 

for acquiring and releasing cloud resources, such as virtual machines (VMs).[2,3] For example, a typical elasticity rule is

IF average CPU usage > 80% AND average memory usage > 50% THEN add 1 VM.

Thus, elasticity rules must be specified with precise quantitative thresholds. To determine such thresholds (for example, 80 percent), cloud users require expertise, which makes the accuracy of the elasticity rules subjective and prone to uncertainty. Furthermore, existing rule-based approaches often make unrealistic assumptions about elastic systems. For instance, they assume that stakeholders agree on the thresholds in the rules. In addition, they don't explicitly consider the presence of measurement inaccuracies in the input data.[4] However, such uncertainties in the cloud environment are frequent.[5,6] Sources of uncertainty in public cloud platforms include the amount of time required to start new VMs, which might be several minutes.[7] This could cause many autoscaling actions to be ineffective since cloud applications are exposed to rapidly changing workloads. In addition, sharing physical resources by different cloud users can lead to significant performance variations among VMs of equivalent capacity, requiring frequent tuning of the application's autoscaling thresholds.[8] (See the sidebar for a discussion of related work.)

In this article, we discuss the main sources of uncertainty and challenges for elasticity management using autonomic cloud controllers. We propose a control-theoretic approach[5] to implement a monitor, analyze, plan, execute, and knowledge (MAPE-K) adaptation loop[9] by designing a fuzzy logic controller. We extend this controller with an online learning loop to enable knowledge evolution in MAPE-K, proposing the MAPE-KE loop, which can cope with various types of uncertainty while ensuring application performance and cost.

## Uncertainty Sources and Challenges in Cloud Elasticity Management

Uncertainty emerges from various sources in elastic cloud systems, such as different interpretations and decisions in the scaling rule definition, internal decision making processes, or monitoring systems that produce partially unreliable and incomplete data.

### Uncertainty in Elasticity Policy Definition

The specification of elasticity policies needs a careful determination of lower and upper thresholds. This determination relies on a user's in-depth knowledge of system behavior over time and how resources are managed.[2] Therefore, the overall accuracy of elasticity policies remains subjective, making the effect of any scaling rule prone to uncertainty. In addition, unpredictable changes in the environment or on the application demand might require scaling rules to be continuously reevaluated and tuned, which is a nontrivial task even for expert users.

The main challenge here is to enable the specification of human-intuitive scaling rules to alleviate the expertise requirement from cloud users. An even bolder challenge is to eliminate human intervention altogether by having the controller define and fine-tune its own scaling rules.

### Uncertainty in Dynamic Resource Provisioning

The process of acquiring and releasing virtual resources in the cloud isn't instantaneous. The elasticity controller needs to invoke cloud platform services to initiate the acquisition process and must wait until new VMs have been spun up to allow the application components to be deployed on newly provisioned resources, leading to a better quality of service. During this time, which often lasts several minutes,[7] the cloud application is vulnerable to workload increase. This makes resource provisioning also prone to uncertainty.

The main challenge here is to identify trends in the monitoring data to anticipate the need for further resources ahead of time. This would let the controller acquire resources proactively, expediting the execution of autoscaling actions with minimum impact on application performance.

### Uncertainty in Monitoring Data

The cloud controller needs to continuously monitor the state of the application as well as that of the cloud resources in which the application is deployed in order to react to possible load variations in a timely manner. Monitoring data usually corresponds to a distribution of values collected by measurement-specific probes

## RELATED WORK IN CLOUD AUTOSCALING

Cloud autoscaling is a hot research topic that has attracted the interest of researchers from both academia and industry.[1–3] Here, we consider some representative approaches that, like ours, focus on elasticity techniques to deal with unpredictable workloads and other types of uncertainties when dynamically allocating resources to cloud-based applications. A more comprehensive review of existing cloud autoscaling solutions and services is available elsewhere.[2]

Jia Rao and his colleagues adopt a multilayer approach to handle multi-objective requirements such as performance and power in dynamic resource allocation.[4] The lower layer focuses on each objective and exploits a fuzzy controller model proposed earlier.[5] The higher layer maintains a tradeoff between the multiple objectives by coordinating their respective controllers. Enda Barrett and his colleagues use reinforcement learning (RL) techniques to automanage cloud resources.[6] However, their solution is model based and only applicable for stable workloads, since they must recalculate the models when conditions change at runtime. Alessio Gambi and his colleagues design cloud controllers as black-box surrogate models of the evolving system and use machine learning techniques to predict system performance under different usage scenarios.[7] Finally, Harold Lim and his colleagues[8] and Ahmad Al-Shishtawy and Vladimir Vlassov[9] propose cloud controllers specifically tailored to cope with uncertainties raised at the cloud storage tier, such as actuator delays due to the need to rebalance data across storage nodes.

Unlike these approaches, our work offers a seamless knowledge evolution-based self-adaptation solution through fuzzy control and RL, taking the burden of defining adaptation rules from the users.

**References**


1. L.M. Vaquero, L. Rodero-Merino, and R. Buyya, "Dynamically Scaling Applications in the Cloud," *ACM SIGCOMM Computer Comm. Rev.*, vol. 41, no. 51, 2011, pp. 45–52.
2. T. Lorido-Botran, J. Miguel-Alonso, and J.A. Lozano, "A Review of Auto-scaling Techniques for Elastic Applications in Cloud Environments," *J. Grid Computing*, vol. 12, no. 4, 2014, pp. 559–592.
3. H.C. Lim et al., "Automated Control in Cloud Computing: Challenges and Opportunities," *Proc. Workshop Automated Control for Datacenters and Clouds* (ACDC), 2009, pp. 13–18.
4. J. Rao et al., "DynaQoS: Model-Free Self-Tuning Fuzzy Control of Virtualized Resources for QoS Provisioning," *Proc. Int'l Workshop Quality of Service* (IWQoS), 2011, pp. 1–9.
5. J. Xu et al., "Autonomic Resource Management in Virtualized Data Centers Using Fuzzy Logic-Based Approaches," *Cluster Computing*, vol. 11, no. 3, 2008, pp. 213–227.
6. E. Barrett, E. Howley and J. Duggan, "Applying Reinforcement Learning Towards Automating Resource Allocation and Application Scalability in the Cloud," *Concurrency and Computation: Practice and Experience*, vol. 25, no. 12, 2013, pp. 1656–1674.
7. A. Gambi, M. Pezze, and G. Toffetti, "Kriging-Based Self-Adaptive Cloud Controllers," *IEEE Trans. Services Computing*, preprint, 2014; doi:10.1109/TSC.2015.2389236.
8. H.C. Lim, S. Babu, and J.S. Chase, "Automated Control for Elastic Storage," *Proc. Int'l Conf. Autonomic Computing* (ICAC), 2010, pp 1–10.
9. A. Al-Shishtawy and V. Vlassov, "Elastman: Elasticity Manager for Elastic Key-Value Stores in the Cloud," *Proc. ACM Cloud and Autonomic Computing Conf.* (CAC), 2013, pp. 7:1–7:10.


or sensors, which aren't immune to measurement deviations (this can be associated with so-called *sensory noise*[1]). For instance, a probe monitoring the response time of an application hosted in the cloud might return slightly different values at different times. This sensory noise is another source of uncertainty, as it results in oscillations that can affect how the controller allocates resources to applications.[3]

The main challenge here is to build a robust controller that sustains sensory noise. This would avoid oscillations in resource allocation due to slight variations in the monitored values.

Table 1. Sample of expert responses for elasticity policies.

| Rule | Antecedents | | Consequent | | | | | Weighted average ($C_{avg}$) |
|---|---|---|---|---|---|---|---|---|
| | Workload | Response time | –2 | –1 | 0 | 1 | 2 | |
| 1 | Very low | Instantaneous | 7 | 2 | 1 | 0 | 0 | –1.6 |
| 7 | Low | fast | 2 | 7 | 1 | 0 | 0 | –1.1 |
| 13 | Medium | Medium | 0 | 0 | 5 | 4 | 1 | 0.6 |
| 19 | High | Slow | 0 | 0 | 1 | 7 | 2 | 11 |
| 25 | Very high | Very slow | 0 | 0 | 0 | 4 | 6 | 16 |

## A Fuzzy Control-Theoretic Approach for Robust Elastic Cloud Systems

We propose RobusT2Scale, a control-theoretic elasticity management approach based on fuzzy control (https://github.com/pooyanjamshidi/RobusT2Scale). It enables qualitative specifications of elasticity rules but also deals with noise and uncertainty arising from monitoring in cloud environments.[4,5] As our experimental results show, RobusT2Scale is robust to several forms of changes in the environment, including unpredictable changes in application demand as well as unpredictable degradations of system performance.

### Fuzzy Controller Design

A cloud-based elastic system has three parts: a cloud-based application, a cloud platform, and an elasticity controller. The elasticity controller implements the following tasks, which form the MAPE-K control loop[9]:

- *monitor* the application and its environment (that is, in control-theoretic terms disturbances such as workload);
- *analyze* the input data and detect any possible violation;
- *plan* corrective actions in terms of adding resources or removing existing unused ones;
- *execute* the plan according to a specific platform; and
- use or update shared *knowledge*.

The knowledge base consists of the scaling rules. We chose performance (response time) and workload (number of accesses) as the two input parameters for our controller.

**Enabling qualitative specification of scaling rules.** Our aim is to facilitate qualitative, deliberately imprecise specification of scaling rules. For example, an elasticity rule might be expressed qualitatively as follows:

IF workload is *high* AND response time is *slow*
THEN add 2 VMs

To this end, we rely on fuzzy logic systems (FLSs) to enable the manipulation of linguistic rules.[10] In particular, we use so-called type-2 FLS[10] to represent the uncertainties embedded in these linguistic labels, such as high and low, and the numerical manipulation of these rules to plan the scalability of cloud-based applications.

The scaling rule antecedents need to capture linguistic impreciseness. The linguistic variable representing the value of a workload is divided into five levels: *very low*, *low*, *medium*, *high*, and *very high*. Similarly, the linguistic variable representing the response time is also divided into five levels: *instantaneous*, *fast*, *medium*, *slow*, and *very slow*. The rule consequent is divided into the number of VM nodes that are added or removed. To design the fuzzy rules, we collected the required data by interviewing 10 cloud experts. We asked these experts to determine a consequent using an integer from [–2, 2]. As we expected, different experts chose different numbers of node instances for the same questions (see Table 1). Note that the final consequent associated with each rule is determined by the weighted average, denoted by $C_{avg}$, of the consequents given to that rule. We also asked the experts to locate an interval for each linguistic label for workload and response time in the range [0, 100]. For the labels, we received 10 different intervals. We then calculated the mean and deviations of the two ends (see Table 2).

**Defining membership functions.** The first step to fuzzy reasoning is to "fuzzify" the monitoring inputs for further processing. We map each linguistic concept into a type-2 fuzzy logic membership function (MF). We use trapezoidal MFs to represent the

**Table 2. Workload and response time quantifications.**

| Linguistic label | | Mean | | Standard deviation | |
|---|---|---|---|---|---|
| | | Start (a) | End (b) | Start ($\sigma_a$) | End ($\sigma_b$) |
| 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 |

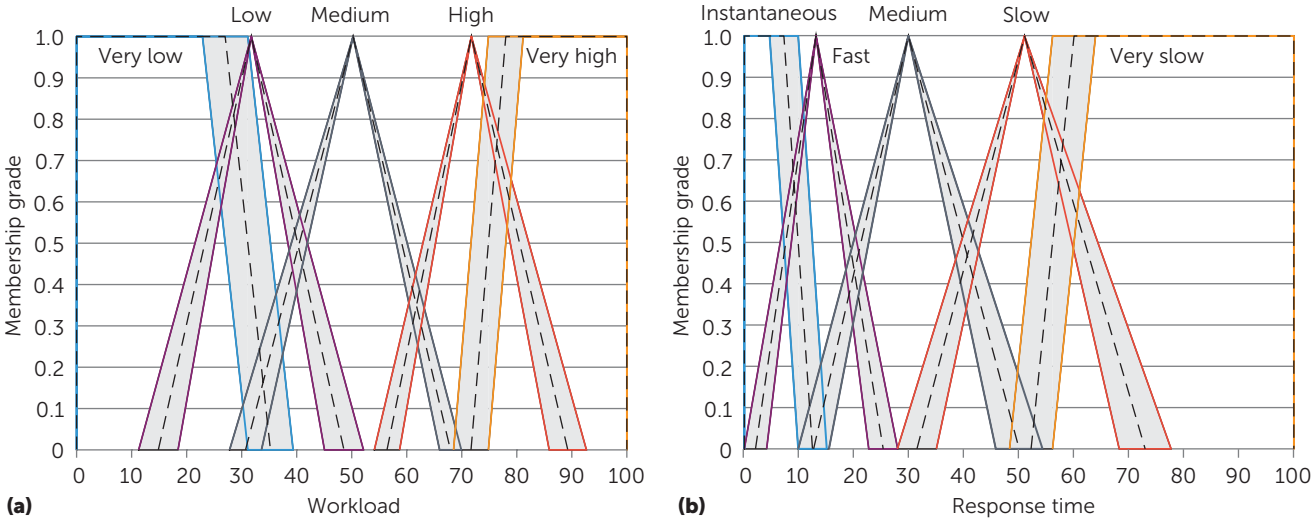


**FIGURE 1.** The type-2 membership functions (MFs) are derived from the user's data to define the antecedents of the autoscaling rules: (a) workload and (b) response time.

outer concepts *very low* (*instantaneous*) and *very high* (*very slow*), and triangular MFs to represent *low* (*fast*), *medium*, and *high* (*slow*). If $a$ and $b$ with standard deviations $\sigma_a$ and $\sigma_b$, respectively, are the mean values of the interval end points of the labels (see Table 2), the MFs can be constructed based on the overall value distribution for the labels and the distribution of the respective means for each user. The result is a trapezoidal or triangular type-2 MF, defined by lower and upper type-1 MFs capturing the degree of uncertainty between the different interpretations of the different users, as Figure 1 shows.

**Fuzzy controller workflow.** The designed fuzzy controller works as follows (see Figure 2):

- The monitoring inputs comprising the workload ($w$) as well as the response time ($rt$) are first fuzzified.
- The fuzzified inputs activate the inference engine to produce type-2 fuzzy sets.
- Decisions made by fuzzy inference are in the form of fuzzy values, which can't be directly used. The fuzzy values are type-reduced, which combines the output sets and calculates the set center via the MF's centroid (dotted line).

- The type-reduced fuzzy sets are type-1 fuzzy sets that need to be defuzzified to determine the scaling actions (*sa*) in terms of the number of VM nodes that need to be added or removed.
- These are then fed to the resource allocator (actuator) to enact the change.

Figure 3 illustrates the control surface generated by the type-2 fuzzy sets computed by applying different combinations of workload and response time values. This provides a smoother resources management under workload and performance variations compared to type-1 solutions. More details on designing the RobusT2Scale fuzzy controller can be found elsewhere.[4]

**Enabling workload and performance prediction.** To allow the controller to anticipate drastic changes in demand and application behavior, and also to alleviate the VM provisioning latency, we combined the fuzzy control mechanism with exponential smoothing prediction. Exponential smoothing is a technique applied to time series data, either to produce smoothed data for presentation or to make forecasts about future system behavior. We use two types for workload and performance prediction:

- When an application starts running, we employ a time series forecasting technique to estimate the future workload. We use double exponential smoothing because it can smooth the inputs and predict trends in historical data.
- For estimating response time, we use single exponential smoothing because, for oscillatory response times, we don't need to predict the trend, but only require a smoothed value.

Both exponential smoothing techniques put a weight on the historical data by a series of exponentially decreasing factors, making recently observed data more relevant.

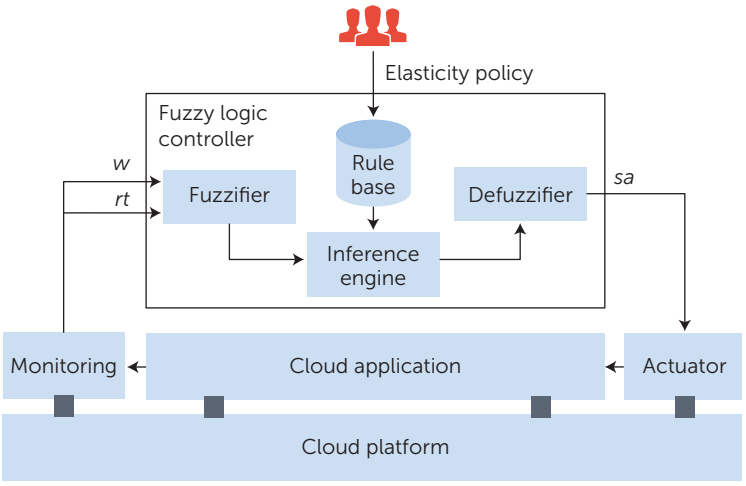


FIGURE 2. RobusT2Scale realizes the MAPE-K control loop by integrating Monitoring, Fuzzy Reasoning and Cloud Actuator components in a cohesive architecture.

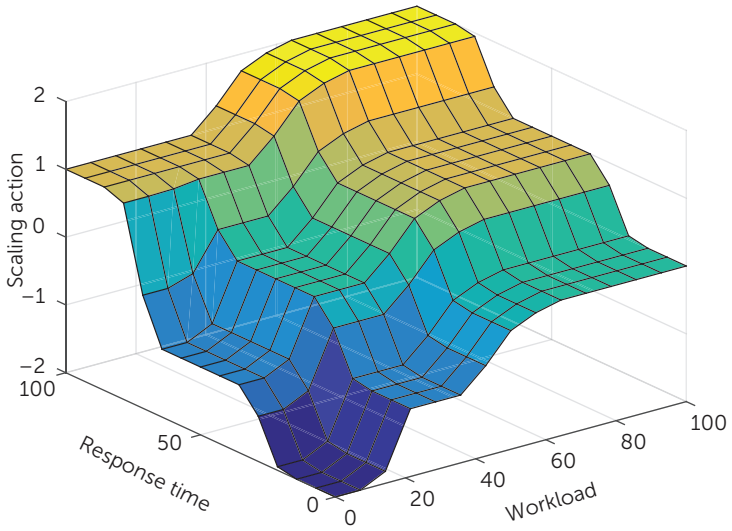


FIGURE 3. Control surface of RobusT2Scale, the type-2 fuzzy controller for elasticity reasoning, for different combinations of workload and response time.

## Experimental Evaluation

We conducted several experimental studies in the Microsoft Azure cloud to evaluate the proposed fuzzy elasticity controller in terms of its prediction accuracy, its effectiveness in guaranteeing service-level objectives (SLOs), and its robustness against measurement noise. To this end, we developed a stateless, dynamically scalable cloud application framework based on our fuzzy controller, called ElasticBench (https://github.com/pooyanjamshidi/ElasticBench). ElasticBench includes a workload generator to simulate different workload patterns, allowing us to test and train the controller before actual execution. To build a generic workload generator, we developed a service to generate Fibonacci numbers. A delay is embedded in the process of calculating Fibonacci numbers to simulate a process that takes a reasonably long period. Note that calculating Fibonacci numbers is an $O(N)$ task, making it a good candidate for demonstrating different application types by embedding different delays, since our platform can generate requests with

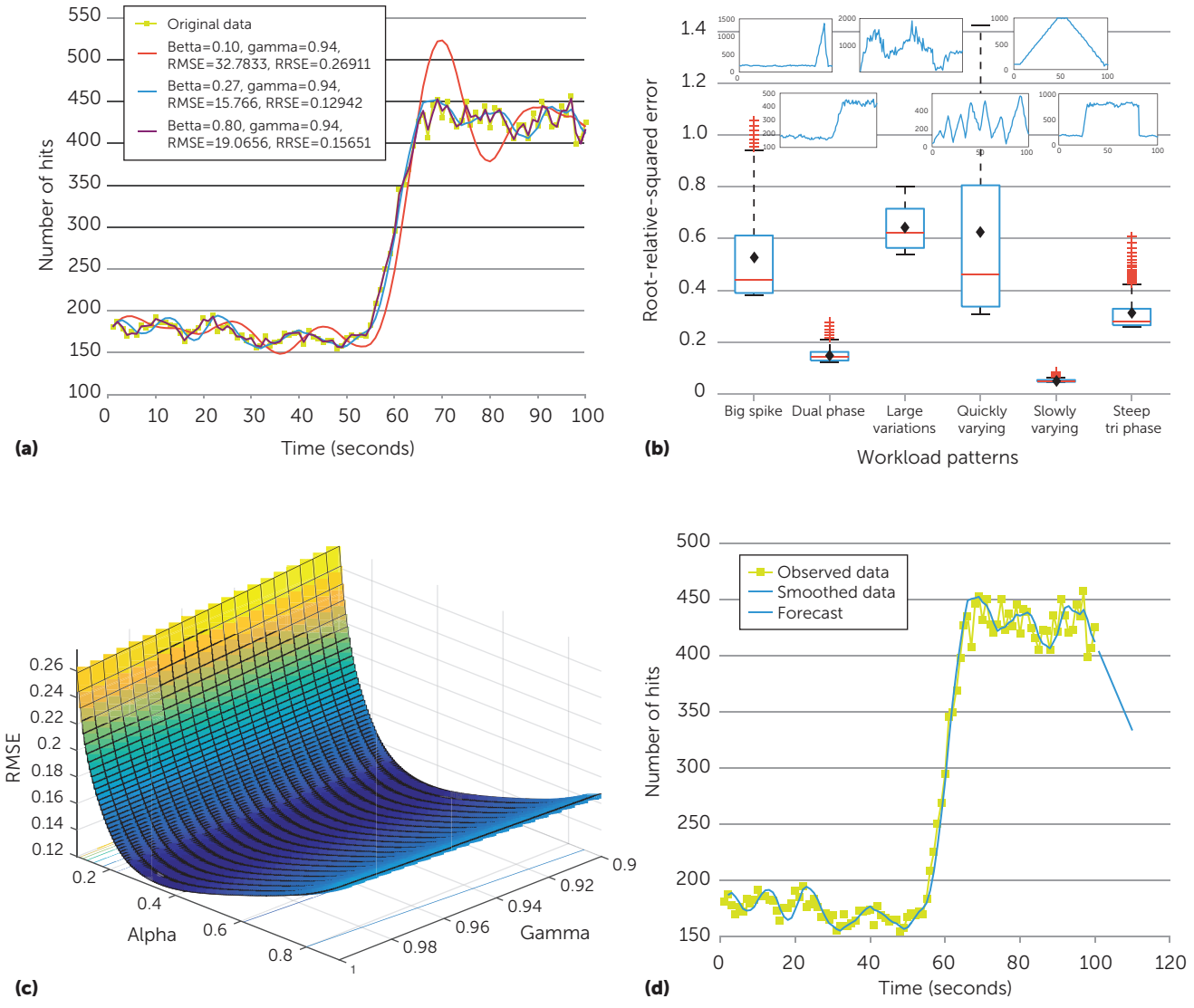


**FIGURE 4.** Accuracy evaluation: (a) estimation versus actual workload, (b) estimation error, (c) prediction error, and (d) workload prediction. (RMSE: root-mean-square error; RRSE: root-relative-squared error)

varying patterns (as Figure 4 shows, different values of Fibonacci numbers are generated according to the level of requests).

We use two types of Azure service roles to implement ElasticBench: *web role* and *worker role*. Both roles run on Azure A1 (small) instances. The requests issued from the load generator are received by the web role, which puts a message on a task assignment queue. The worker role instances continuously check the queue, and a background process (to calculate the Fibonacci number) is immediately started based on the content of the message in the queue. The worker roles communicate with the Azure storage service to acquire the data required for processing (for example, previously calculated Fibonacci numbers). Once the controller decides to scale-in (out) the resources, ElasticBench increases (decreases) the number of worker roles accordingly. Further details about our experimental setup are available elsewhere.[4]

**Prediction accuracy.** To evaluate the accuracy of the controller's exponential smoothing estimation technique, we simulated different workloads and measured the error of estimation by root-relative-squared error (RRSE).

Figure 4a shows sample data and different estimations achieved by changing the model's parameters, resulting in different levels of prediction accuracy. We also evaluated the prediction techniques' accuracy for six different workload patterns

Table 3. Comparison of the effectiveness of RobusT2Scale.

| Approach | Criteria | Workload patterns | | | | | |
|---|---|---|---|---|---|---|---|
| | | Big spike | Dual phase | Large variations | Quickly varying | Slowly varying | Steep tri phase |
| RobusT2Scale | *rt*_95 | 973 ms | 537 ms | 509 ms | 451 ms | 423 ms | 498 ms |
| | *vm* | 3.2 | 3.8 | 5.1 | 5.3 | 3.7 | 3.9 |
| Overprovisioning | *rt*_95 | 354 ms | 411 ms | 395 ms | 446 ms | 371 ms | 491 ms |
| | *vm* | 6 | 6 | 6 | 6 | 6 | 6 |
| Underprovisioning | *rt*_95 | 1,465 ms | 1,832 ms | 1,789 ms | 1,594 ms | 1,898 ms | 2,194 ms |
| | *vm* | 2 | 2 | 2 | 2 | 2 | 2 |

(*big spike*, *dual phase*, *large variations*, *quickly varying*, *slowly varying*, and *steep tri phase*), as Figure 4b shows.[4] For three patterns (*slowly varying*, *dual phase*, and *steep tri phase*), the relative error and variations are quite low. The *large variation* pattern shows the large error mean, while the *big spike* and *quickly varying* patterns present the largest variations. Overall, we can conclude that our prediction solution is sufficiently accurate for common workload situations. Figure 4c shows that the prediction accuracy will change with different values of double exponential smoothing hyper-parameters. Figure 4d provides a longer-term prediction based on the current observation. It shows that double exponential smoothing detected the ascending workload trend correctly.

**Effectiveness.** Scalability is concerned with variances that are large enough to warrant a scaling action. As a benchmark for measuring the controller's effectiveness, we consider

- the 95th percentile of the response time (*rt*_95), which represents our performance SLO; and
- the weighted average number of node instances acquired over time (*vm*), which determines the cost.

These criteria cover the three main aspects of elasticity: scalability, cost, and time efficiency. The goal is to meet the response time SLO (we assume *rt*_95 = 600 ms) while keeping *vm* as low as possible.

Given these criteria and goals, we compared our approach with two common basic provisioning approaches, overprovisioning (*vm* = 6) and underprovisioning (*vm* = 2), using the same six workload patterns we used for the accuracy evaluation. Table 3 shows the results.

In comparison with the overprovisioning approach, RobusT2Scale acquired fewer nodes, saving a factor of two in cost. As the table shows, RobusT2Scale didn't violate the response time SLO in workload patterns except for the *big spike* pattern. The overprovisioning approach satisfied the SLO for all patterns, but only by imposing twice the cost for *big spike* (for the other patterns, the difference is less significant) compared to RobusT2Scale. The underprovisioning approach never met the SLO for any pattern.

**Robustness against noise.** RobusT2Scale is resilient against different input noise types, one of which is the estimation error.

In our experiments, we observed that the worst estimation error happens for *large variations* and *quickly varying* patterns and is less than 10 percent of the actual workload. As a result, we injected white noise to the input measurement data with a 10 percent amplitude. We ran root-mean-square error (RMSE) measurements for each level of blurring. For each measurement, we used 10,000 data items as input. 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 for a controller with a higher blurring. A higher blurring leads to a larger footprint of uncertainty (FOU), which is representative of the supporting levels of uncertainty. Since an overly wide FOU can result in performance degradations,[8] the controller designer should take into account the level of uncertainty that the controller can support.

These observations provide evidence that RobusT2Scale is also robust against input noise.

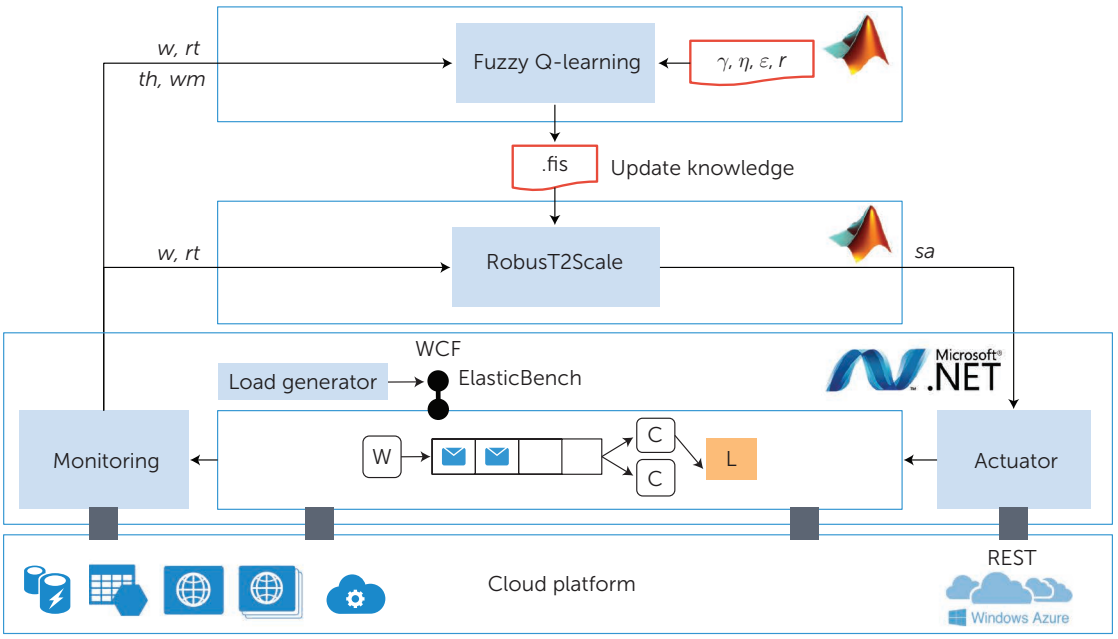


FIGURE 5. Fuzzy Q-learning for knowledge evolution (FQL4KE) integrates RobusT2Scale with an online learning mechanism into a cohesive architecture to enable on-the-fly autoscaling knowledge evolution.

## MAPE-KE: Enhancing MAPE-K with Knowledge Evolution

The scaling strategies we've considered so far rely on a set of rules, known as *elasticity policies*, to scale up/down the required resources depending on the application usage. However, from the cloud provider's perspective, applications running in their infrastructure are basically black boxes, making it difficult to accurately devise optimal or preemptive measures against under- or overprovisioning. RobusT2Scale facilitates elasticity policy elicitation through linguistic variables. However, this solution still requires input from users. To make entirely user-independent cloud controllers, we integrate the MAPE-K adaptation loop that was primarily implemented in RobusT2Scale (see Figure 2) with an online learning loop for knowledge evolution (KE). The MAPE-KE loop provides the basis for designing robust self-learning cloud controllers. We implemented one such controller, *fuzzy Q-learning for knowledge evolution* (FQL4KE; https://github.com/pooyanjamshidi/Fuzzy-Q-Learning), as an extension to RobusT2Scale that autonomously scales the cloud infrastructure without having to rely on user input for defining scaling rules (see Figure 5).

### Reinforcement Learning

FQL4KE implements a fuzzy rule-based reinforcement learning (RL) algorithm that learns and modifies fuzzy rules at runtime.[6,11] The updated fuzzy rules are fed into the controller's knowledge base. The implication is that the controller no longer needs to rely on knowledge provided by users. This complements existing rule-based cloud controllers, enabling users to specify system goals by simply adjusting some weights instead of specifying complex elasticity rules.

Our approach is based on a particular RL technique that combines fuzzy control and fuzzy Q-learning (FQL) to bring human expertise to continuous evolution machinery.[6] Q-learning is particularly suitable for situations in which it's possible to learn from interaction with the environment (that is, a control-theoretic setting[5]), where the learning is performed via a reward mechanism. The combination of fuzzy control and FQL results in a powerful self-adaptive mechanism where the fuzzy control facilitates reasoning at a higher level of abstraction (that is, human reasoning) and the Q-learning allows the controller to adapt/adjust the knowledge on the fly to provide more accurate decisions. RL is generally an appropriate fit for this problem because the workloads for cloud-based applications are unpredictable and obtaining actual training data that's representative of all runtime situations is challenging. However, unlike other supervised learning approaches (such as neural networks), RL doesn't require a training dataset. In addition, due

Table 4. Comparison of the effectiveness of FQL4KE, RobusT2Scale, and Azure autoscaling under different workloads.

| Approach | Criteria | Workload patterns | | | | | |
|---|---|---|---|---|---|---|---|
| | | Big spike | Dual phase | Large variations | Quickly varying | Slowly varying | Steep tri phase |
| FQL4KE | *rt_95* | 1,212 ms | 548 ms | 991 ms | 1,319 ms | 512 ms | 561 ms |
| | *vm* | 2.2 | 3.6 | 4.3 | 4.4 | 3.6 | 3.4 |
| RobusT2Scale | *rt_95* | 1,339 ms | 729 ms | 1,233 ms | 1,341 ms | 567 ms | 512 ms |
| | *vm* | 3.2 | 3.8 | 5.1 | 5.3 | 3.7 | 3.9 |
| Azure autoscaling | *rt_95* | 1,409 ms | 712 ms | 1,341 ms | 1,431 ms | 1,101 ms | 1,412 ms |
| | *vm* | 3.3 | 4 | 5.5 | 5.4 | 3.7 | 4 |

to workload unpredictability and the complexity of the cloud-based application, providers don't have sufficiently accurate and complete knowledge to take proper scaling actions in every situation.

### Experimental Results

To experimentally demonstrate our self-learning controller approach's applicability, we integrated FQL4KE into ElasticBench and conducted further studies in the Microsoft Azure cloud. Our experimental results (summarized in Table 4, with details available elsewhere[6,12]) build confidence in FQL4KE's effectiveness and efficiency:

- FQL4KE performs better than Azure's native autoscaling service and better than or similarly to RobusT2Scale for all six workloads investigated, in terms of both response time and the average number of node instances acquired.
- FQL4KE can learn to acquire resources for dynamic cloud systems. Our results show that the control surface (see Figure 3 for RobusT2Scale) converges starting from a constant plane at point zero over time to an effective one, with an acceptable 10 percent runtime overhead on monitoring and actuation.
- FQL4KE is flexible enough to allow the operator to set different elasticity strategies. We considered different autoscaling strategies with respect to different environmental condition that might arise. FQL4KE is effective in terms of learning optimal policies and updating them at runtime.

Overall, our evaluation confirms the main advantages of the FQL4KE self-learning approach. First, FQL4KE's self-adaptive and self-learning capabilities make it robust to highly dynamic workload intensity. In addition, FQL4KE is model-independent. The variations in the performance of the deployed applications and the unpredictability of dynamic workloads don't impact the proposed approach's effectiveness. FQL4KE is also capable of automatically constructing the control rules and keeping control parameters updated through fast online learning. It executes resource allocation and learns to improve its performance simultaneously. Finally, unlike supervised techniques that learn from training data, FQL4KE doesn't require offline training, which saves significant amounts of time and effort.

What emerges from this discussion is that we can understand cloud controllers as control theory-based tools. Our solutions use fuzzy control theory combined with machine learning to specifically deal with uncertainty in its various forms. We are currently working on different machine learning mechanisms that enable knowledge evolution in FQL4KE, a MAPE-KE framework that we put forward in this work. We are also working on different realizations of FQL4KE on different cloud platforms such as OpenStack and Amazon EC2.

### References


1. L.M. Vaquero, L. Rodero-Merino, and R. Buyya, "Dynamically Scaling Applications in the Cloud," *ACM SIGCOMM Computer Comm. Rev.*, vol. 41, no. 51, 2011, pp. 45–52.
2. H. Ghanbari et al., "Exploring Alternative Approaches to Implement an Elasticity Policy," *Proc. IEEE Int'l Conf. Cloud Computing* (ICCC), 2011, pp. 716–723.

3. T. Lorido-Botran, J. Miguel-Alonso, and J.A. Lozano, "A Review of Auto-scaling Techniques for Elastic Applications in Cloud Environments," *J. Grid Computing*, vol. 12, no. 4, 2014, pp. 559–592.
4. P. Jamshidi, A. Ahmad, and C. Pahl, "Autonomic Resource Provisioning for Cloud-Based Software," *Proc. Int'l Symp. Software Eng. Adaptive and Self-Managing Systems* (SEAMS), 2014, pp. 95–104.
5. A. Filieri et al. "Software Engineering Meets Control Theory," *Proc. Int'l Symp. Software Eng. Adaptive and Self-Managing Systems* (SEAMS), 2015, pp. 71–82.
6. P. Jamshidi et al., "Self-Learning Cloud Controllers: Fuzzy Q-Learning for Knowledge Evolution," *Proc. IEEE Int'l Conf. Cloud and Autonomic Computing* (ICCAC), 2015, pp. 208–211.
7. M. Mao and M. Humphrey, "A Performance Study on the VM Startup Time in the Cloud," *Proc. Int'l Conf. Cloud Computing* (CLOUD), 2012, pp. 423–430.
8. H.C. Lim et al., "Automated Control in Cloud Computing: Challenges and Opportunities," *Proc. Workshop Automated Control for Datacenters and Clouds* (ACDC), 2009, pp. 13–18.
9. J.O. Kephart and D.M. Chess, "The Vision of Autonomic Computing," *Computer*, vol. 36, no. 1, 2003, pp. 41–50.
10. J.M. Mendel, "Type-2 Fuzzy Sets and Systems: An Overview," *Computational Intelligence Magazine*, vol. 2, no. 1, 2007, pp. 20–29.
11. E. Barrett, E. Howley and J. Duggan, "Applying Reinforcement Learning Towards Automating Resource Allocation and Application Scalability in the Cloud," *Concurrency and Computation: Practice and Experience*, vol. 25, no. 12, 2013, pp. 1656–1674.
12. P. Jamshidi et al., "Fuzzy Self-Learning Controllers for Elasticity Management in Dynamic Cloud Architectures," *Proc. Int'l ACM Sigsoft Conf. Quality of Software Architectures* (QoSA 16), 2016, pp. 70–79.



**POOYAN JAMSHIDI** *is a postdoctoral research associate in Imperial College London's Department of Computing. His interests include self-adaptive software, in particular, applying statistical machine learning and control theory to enable self-organizing behaviors in distributed systems for processing big data. Jamshidi received a PhD in computing from Dublin City University. He's a member of IEEE, ACM SIGSOFT, and the Standard Performance Evaluation Corporation. Contact him at p.jamshidi@imperial.ac.uk.*

**CLAUS PAHL** *is an associate professor at the Free University of Bozen-Bolzano, Italy. His research interests include software engineering in service and cloud computing, specifically migration and scalability concerns. Pahl has a PhD in computing from the University of Dortmund. Contact him at claus.pahl@unibz.it.*

**NABOR C. MENDONÇA** *is a full professor at the University of Fortaleza, Brazil. His research interests include software engineering and distributed systems, with a focus on the development and evaluation of cloud computing technologies. Mendonça has a PhD in computing from Imperial College London. He's a member of IEEE, ACM, and the Brazilian Computer Society. Contact him at nabor@unifor.br.*