



Design and evaluation of a biologically-inspired cloud elasticity framework

Amjad Ullah¹ · Jingpeng Li² · Amir Hussain³

Received: 29 January 2019 / Revised: 6 November 2019 / Accepted: 14 February 2020
© The Author(s) 2020

Abstract

The elasticity in cloud is essential to the effective management of computational resources as it enables readjustment at runtime to meet application demands. Over the years, researchers and practitioners have proposed many auto-scaling solutions using versatile techniques ranging from simple if-then-else based rules to sophisticated optimisation, control theory and machine learning based methods. However, despite an extensive range of existing elasticity research, the aim of implementing an efficient scaling technique that satisfies the actual demands is still a challenge to achieve. The existing methods suffer from issues like: (1) the lack of adaptability and static scaling behaviour whilst considering completely fixed approaches; (2) the burden of additional computational overhead, the inability to cope with the sudden changes in the workload behaviour and the preference of adaptability over reliability at runtime whilst considering the fully dynamic approaches; and (3) the lack of considering uncertainty aspects while designing auto-scaling solutions. In this paper, we aim to address these issues using a holistic biologically-inspired feedback switch controller. This method utilises multiple controllers and a switching mechanism, implemented using fuzzy system, that realises the selection of suitable controller at runtime. The fuzzy system also facilitates the design of qualitative elasticity rules. Furthermore, to improve the possibility of avoiding the oscillatory behaviour (a problem commonly associated with switch methodologies), this paper integrates a biologically-inspired computational model of action selection. Lastly, we identify seven different kinds of real workload patterns and utilise them to evaluate the performance of the proposed method against the state-of-the-art approaches. The obtained computational results demonstrate that the proposed method results in achieving better performance without incurring any additional cost in comparison to the state-of-the-art approaches.

Keywords Cloud elasticity · Dynamic resource provisioning · Fuzzy control system · Basal ganglia · Auto-scaling · Switched controller · Elastic feedback controller

1 Introduction

The pool of virtually unlimited on-demand computational resources, provided by cloud providers (CPs), and many attractive features of cloud computing, such as pay-as-you-go pricing and on-the-fly re-adjustment of hired computational resources (elasticity), is a perfect match to host web applications that are subject to fluctuating workload conditions [1, 2]. The cloud's elasticity allows applications to dynamically adjust the underlying computational resources in response to the changes observed in the environment, thus enabling application service providers (SPs) to meet application demands and pay only for the resources that are necessary [3].

✉ Amjad Ullah
a.ullah@westminster.ac.uk

Jingpeng Li
jli@cs.stir.ac.uk

Amir Hussain
a.hussain@napier.ac.uk

¹ Department of Computer Science, University of Westminster, London, UK

² Division of Computing Science and Mathematics, University of Stirling, Stirling, UK

³ School of Computing, Edinburgh Napier University, Edinburgh, UK

Over the years, researchers and practitioners have proposed many elastic methods using versatile techniques including but not limited to rule-based [4–8], control theory [9–13], fuzzy logic [14, 15], optimisation [16–18] and machine learning [19, 20]. However, despite a large range of existing elasticity research work, the aim of implementing an efficient scaling technique that satisfies the actual demands is still a challenge to achieve [21–23]. This is evident from the low utilisation, estimated as 8% to 20%, of the server capacity purchased by the SPs [24].

The existing research literature on cloud elasticity differs in various aspects, e.g. triggering behaviour (Reactive/Predictive/Hybrid), scope (CPs/SPs perspective), dependency on metrics (CPU utilisation/Response time, etc.), and the implementation technique (Control Theory/Machine learning/Rule-based, etc.). Despite such differences most of the existing methods can generally be grouped into *Fixed* or *Adaptive* categories based on their design and working mechanism to analyse their pros and cons as a whole [25].

The *Fixed* class refers to the family of all elastic methods that are designed off-line and remain fixed at runtime. On the other hand, the *Adaptive* class indicates methods that are equipped with an on-line learning capability that is responsible for adaptation at runtime in response to changes in the working environment. The *Fixed* approaches are simple, easy to design and better for systems with uniform workload behaviour, e.g. rule-based systems and fixed gain elastic feedback controllers. However, the performance severely affects systems with variable workloads due to lack of adaptability at runtime. In contrast, the *Adaptive* approaches are more flexible due to on-line learning capabilities and they perform better in scenarios with slowly varying workload behaviour. However, they are also criticised for their additional computational cost caused due to the online learning [26], long training delays, their associated risk of reducing the quality assurance of the resulted system and the impossibility of deriving a convergence or stability proof [25].

In contrast to the families mentioned above, this paper advocates a fixed-adaptive (also referred to as *Hybrid* by Gambi et al. [25]) approach, a method commonly associated with the biologically-inspired multi-model switching and tuning (MMST) methods. Using such an approach, an elastic method follows a *Fixed* design principle, but also achieves certain level of adaptive behaviour at runtime. The review of existing state-of-the-art elasticity research (Section 6) indicates that such an approach for implementing cloud elasticity has not received much attention.

Another important factor identified in the existing elasticity literature is the importance of addressing the uncertainty related issues, e.g. impreciseness in domain knowledge and noise in monitoring data. Jamshidi et al

[14, 27] and Farokhi et al [28] stressed the importance of the uncertainty aspects required to be considered while designing elastic controllers. However, despite the importance, the implementation of uncertainty in the context of cloud elasticity has not yet been well received [28]. The methodology proposed in this paper is also a step forward in this direction.

This paper addresses the horizontal elasticity problem from a SP perspective and particularly focuses on contributing towards resolving the following issues in the existing elasticity literature: (1) The lack of adaptability and static scaling behaviour whilst considering completely fixed approaches; (2) The burden of additional computational overhead, the inability to cope with sudden changes in workload behaviour and preference of adaptability over reliability at runtime whilst considering the fully dynamic approaches; (3) The lack of considering uncertainty aspects while designing auto-scaling solutions; and (4) Lastly, the unavailability of solutions that facilitate qualitative elasticity rules to resolve the quantitative nature of the commonly used rule-based approaches. This paper investigates the synergy between the biologically-inspired multi-controller approach and fuzzy control system, to provide a holistic solution to address the aforementioned issues.

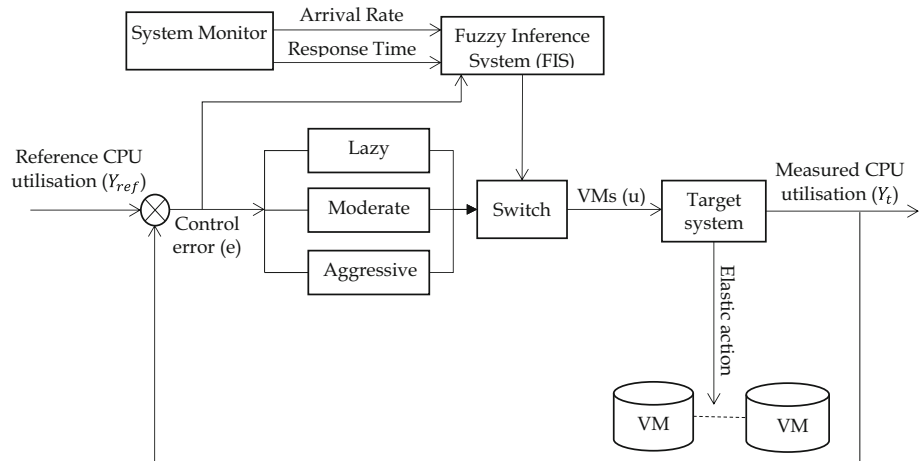
The rest of the paper is organised as follows. The next section provides the design of our proposed biologically-inspired cloud elasticity framework. The design, however consists of two different modes, termed as Hard switching and Soft switching. Therefore, each mode is explained in different section. Section 3 elaborate the customised settings used for experimentation and the results obtained in the case of Hard switching approach. Similarly, Sect. 5 presents the results obtained in the case of Soft switching approach. Section 6 comparatively summarizes the research undertaken in the field of cloud elasticity research. Lastly, Sect. 7 concludes this paper.

2 Biologically-inspired elasticity framework: hard switching

The proposed multi-controller with fuzzy switching framework consists of the use of an array of controllers, where each controller is particularly designed to achieve better performance in a different situation and the selection of a suitable controller is realised at runtime. The architectural diagram of the proposed control methodology can be seen in Fig. 1 which extends and builds on the classical feedback loop model.

The key idea behind the proposed framework is to divide the complexity of the overall system by constructing multiple fixed gain controllers, where each controller

Fig. 1 Hard switching framework



depicts a separate elastic policy that carries out scaling actions at different intensity level. The design of the proposed methodology (or any switched method in general) involves the following two key challenges: (1) how to partition the system among multiple controllers? (2) How to switch (or formulate) the final decision? Due to the lack of a standard approach for partitioning the system among sub controllers [29], this research realises the use of expert-oriented distribution of workload intensity into various categories such as low, moderate and high. For each category, a system model is constructed, based on which a controller is designed. The final decision is carried out by the selection of a suitable controller at runtime using an intelligent switching mechanism, implemented as a fuzzy control system, i.e. also formally called as Fuzzy Inference System (FIS) as represented in Fig. 1.

The proposed control method is responsible for the readjustment of the number of Virtual Machines (VMs) to maintain the average *CPU utilisation* of hired VMs running at that time. The proposed methodology incorporates three *Fixed gain* controllers termed *Lazy*, *Moderate* and *Aggressive*. In theory, the number of controllers depends on the adaptation and application scenario. Increasing the number of controllers facilitates more fine-grained control over cloud resources, however, it also increases the design complexity of the elastic method. Each controller depicts a different elasticity policy, and theoretically they can be implemented using any suitable technique.

The incorporation of fixed controllers with switching ability enables the adaptive behaviour of the system to respond appropriately to the needs of the system in case of changes in workload without the need of any on-line learning algorithm. Each of the controllers is designed to react differently in the various situation. In this case, as their name specifies, they indicate three different scenarios, i.e. to perform scaling action at slower, moderate and aggressive intensity level. The selection of one of this

policy depends on the behaviour of the system at that point in time. The behaviour of the system can be identified using the latest status of the following three aspects including application performance, workload arrivals, and resource utilisation. These aspects are represented as *Response time*, *Arrival rate* and *Control error* respectively in Fig. 1.

The *System Monitor* component of the proposed methodology is responsible for obtaining the latest status of the three parameters mentioned above. These measurements (as shown in Fig. 1) are provided to the FIS. The FIS then decides using the collection of elastic fuzzy rules (Sect. 2.2), what level of intensity is needed for the readjustment of resources (VMs) to meet the desired performance objective (explained in Sect. 2.2.2). The output of the FIS is one of the employed controller that is responsible for making scaling decisions.

2.1 Feedback control

The design and development of the feedback control part of the methodology follows the process flow proposed by Antonio et al. [30]. This process flow consists of the following steps: Defining the goal of control methodology, Identification of control input and devising of system's model and finally, the development, deployment and evaluation of the control methodology. The details of the control system goal, control input, system model and control design in the prospect of our proposed methodology are provided in the following subsections. Whereas, the deployment and evaluation are discussed in Sect. 3.

2.1.1 Goal of control methodology

The goal of the control methodology is to adjust the number of VMs (will also be referred as *Cluster size* in the rest of the paper) at runtime in response to changes in

workload to maintain the *CPU utilisation* of all VMs at a desired reference value. The use of CPU Utilisation is considered here assuming that the performance of the underlying test scenario is influenced by CPU utilisation only. In scenarios, where application performance relies on other metrics (e.g. memory consumption, etc), the CPU Utilisation should be replaced accordingly.

In the context of control system, based on the above-mentioned goal description, CPU utilisation becomes the *Measured output* of the system and we have to identify the *Reference input*, i.e. the target CPU utilisation that results in achieving the desired performance level. For any given application scenario, the desired performance level is the acceptable level of performance, i.e. the mean response time (mRT), that the application owner desires to maintain for their application. In this paper, for the evaluation of the proposed method, we consider the value ($mRT \leq 0.6$ s) as the desired performance measurement. Hence, the scaling mechanism will make changes to the system resources such that the performance of the application achieves the $mRT \leq 0.6$ s. However, response time is an application level metric. Therefore, we need to identify the corresponding CPU utilisation level, where the system will be able to maintain the application $mRT \leq 0.6$ s.

The key reasons for using CPU utilisation as the system output are the following: (1) The CPU utilisation is directly obtained from the CPUs provided monitoring Application Programming Interface (API). Hence it does not require application level monitoring efforts. (2) It is a system specific metric and no runtime relation identification between application metric, e.g. *Response time*, is required. Hence it does not involve additional overhead at runtime. (3) More importantly with respect to our methodology, we have already catered application level metric (i.e. Response time) for decision-making. Thus using CPU utilisation as another metric strengthens the decision-making mechanism by taking into account the system's resource utilisation perspective. Hence, the proposed methodology becomes hybrid in contrast to most of the existing methods that either rely on application [14, 31] or system level metrics [11, 32, 33].

The measurement for *Reference CPU utilisation* can be obtained using system identification (SID) experiments by establishing a relationship between VM CPU utilisation versus performance. This experiment and all other such SID experiments are conducted using an extended version of a well-known cloud simulation tool named CloudSim [34].

The SID experiment records the measurement of CPU utilisation and mRT against several workloads that differentiate regarding the number of incoming requests ranging from 50 requests per minute (rpm) to 950 rpm. Each measurement of CPU utilisation and mRT against the

specified rpm is obtained from sub-experiment, where the corresponding number of rpm are sent for 30 minutes to the system, which consists of one VM. The arrival time of job requests in a minute and the service time of each request is randomly assigned. This whole experiment is repeated 100 times and the average for each measurement is recorded. The obtained results are presented in Fig. 2.

It is evident from Fig. 2 that the increase in the number of rpm makes the mRT slower. The dashed line in Fig. 2 represents the desired performance measurement, and we are interested in the maximum rpm measurement for which the obtained performance is less than the desired target, i.e. ($mRT \leq 0.6$ s). This criterion is satisfied by 850 rpm. However, in this case, there were 13% Service Level Objective (SLO) violations observed, which is not acceptable as per the employed performance objective (will explain in Sect. 2.2.2). Therefore, we do not select the 850 rpm and consider the next measurement, i.e. 800 rpm, that satisfies the criterion mentioned earlier. This means that on average one VM can fulfil maximum 800 rpm on a per-minute basis, while obtaining the desired performance level. Analogously, the number of rpm has a similar effect on CPU utilisation, i.e. the increase in rpm results in an increase in the CPU utilisation as well. For the *Reference input*, we record the corresponding measurement of CPU utilisation from Fig. 2 against the 800 rpm, which is 55%. Thus the control methodology is responsible to maintain the measurement of 55% as the *Reference CPU utilisation*.

2.1.2 Control input

The number of VMs is used as the Control input. This choice is obvious considering horizontal elasticity. Furthermore, we also perform an experiment to demonstrate the impact on mRT with a change in *Cluster size*. Figure 3 demonstrates the obtained results that indicate that increasing the number of VMs reduced the response time.

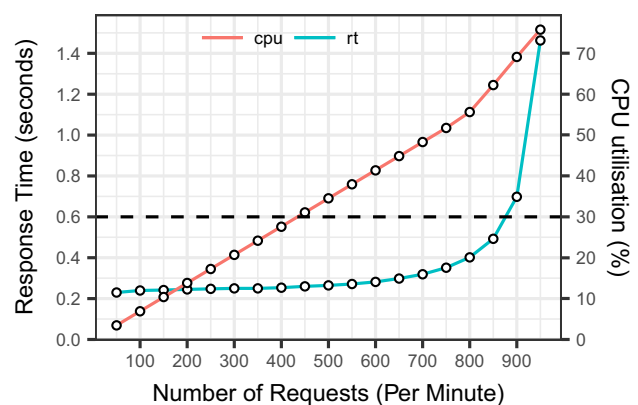


Fig. 2 VM performance

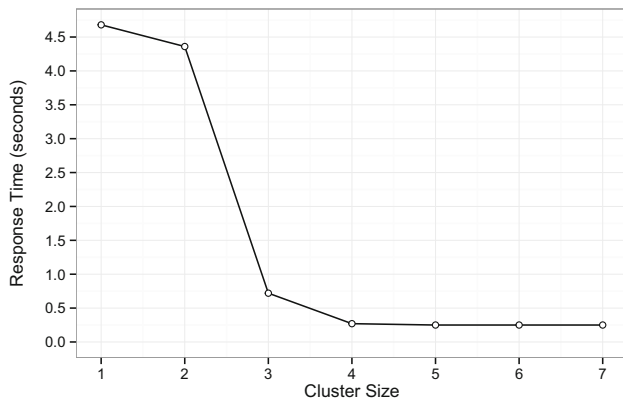


Fig. 3 mRT vs cluster size with constant workload (2800 rpm)

2.1.3 System modelling

This section identifies the system model that describes the relationship between input (number of VMs) and output (CPU utilisation) of the system. We follow the black box modelling approach that mainly consists of SID experiments to obtain training data, building and evaluating the model. The following subsections explain the process.

SID experiments design The SID experiments record the training data consisting of input-output pairs of system by changing the control input in a systematic way during the experiment. During this experiment, we assume that the historical information related to system workload is available and on that basis, we use domain experts based distribution of workload into three categories namely *Low*, *Moderate* and *High*. Using these categories and following the principles of *Gain scheduling* technique where workload-specific models are developed [35]. We conduct three workload category specific experiments. During each experiment, the value of control input is changed as per the discrete Sine wave equation given below:

$$y(t) = m + A * \sin(t) \quad (1)$$

The m in above equation represents mean, A represents amplitude and t represent time step. The time period for each experiment is 540 min long. The difference between each experiment is the use of different pair of (mean, amplitude) values and the use of different workload. The coverage of the input values generated using Eq. 1 during the experiments can be seen from Fig. 4a–c and the corresponding system output recorded in response can be seen from Fig. 4d–f respectively. In the case of system output, the vibrations in the measurement occur as a result that the majority of requests were cancelled because they were unable to complete their execution at a predefined maximum time (2 s).

System model and evaluation The Autoregressive Exogenous Model (ARX) approach is employed to

describe the relationship between the number of VMs and CPU utilisation. The following equation represents the general form of an ARX equation.

$$y(k+1) = a_1y(k) + \dots + a_ny(k-n+1) + b_1u(k) + \dots + b_mu(k-m+1) \quad (2)$$

The above equation represents a single input, single output system. The u and y represent the input and output of the system respectively. According to this equation, the output in next time unit ($k+1$) depends on the n number of previous output values and the m number of previous input values. The a_k and b_k are the constant coefficients values for each output and input value, whereas the m and n represent the order of the model. We use a 1st order ARX model of the following form that can be derived from Eq. 2 by setting $m = n = 1$.

$$y(k+1) = ay(k) + bu(k) \quad (3)$$

The 1st order model, in contrast to m and n order model, relies on the input and output from the previous time unit only. The key reason of selecting the 1st order model is its simplistic nature and the ability to avoid over-fitting [35]. We have to find values for parameter a and b of the above equation from the training data obtained from the SID experimentation. For this purpose, we employ the commonly used least square regression method to estimate the model parameters for all the three experiments mentioned in the previous sections, and the outcome is in the following equations:

$$y(k+1) = 0.89y(k) - 0.18u(k) \quad (4a)$$

$$y(k+1) = 0.93y(k) - 0.07u(k) \quad (4b)$$

$$y(k+1) = 0.95y(k) - 0.03u(k) \quad (4c)$$

These models after validation can be used to design controllers and the following two approaches are normally followed. Firstly, each model could be used to design a different controller as it is obtained based on the average rate of each workload category and thus can be treated as workload-specific models. Secondly, one model could be used to design different controllers where each differs from others based on the controller properties. We follow the second approach and use the model of Eq. 4a for controller design (explain in Sect. 2.1.4).

The next step is to evaluate the model to quantify its accuracy. For this purpose, we employ a widely used method known as the coefficient of determination (denoted by R^2). The value of R^2 can be calculated using the following equation:

$$R^2 = 1 - \frac{\text{var}(y - \hat{y})}{\text{var}(y)} \quad (5)$$

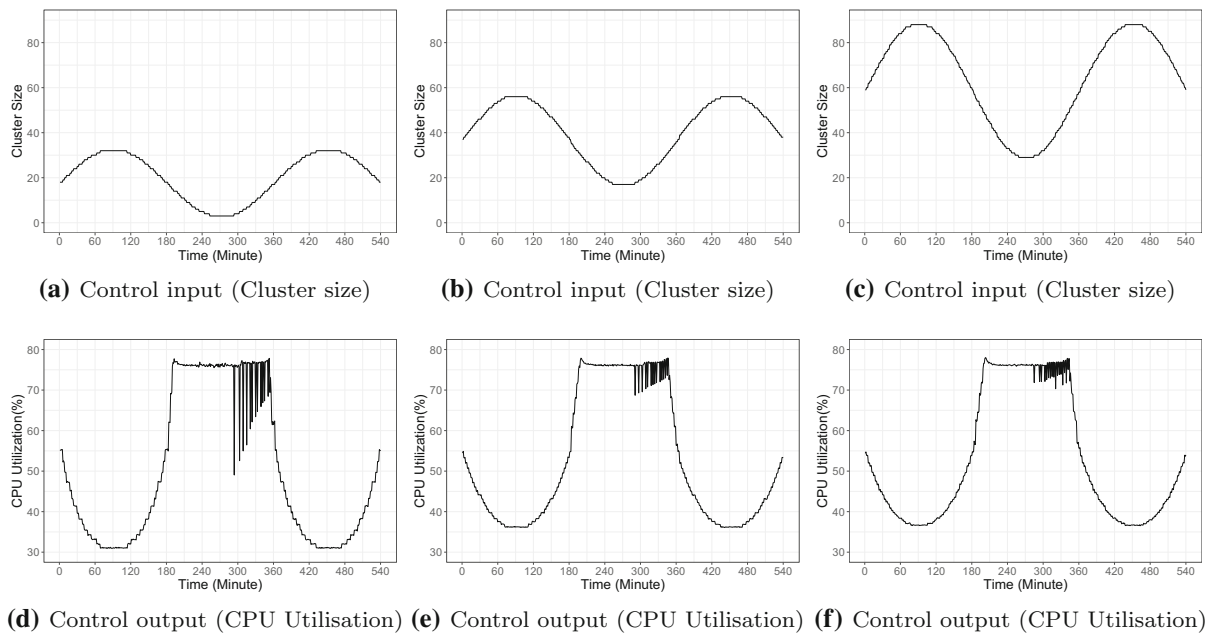


Fig. 4 SID experiments: each pair presents the relationship between cluster size and CPU utilisation

The y in above equation represents the actual system output value, where \hat{y} indicates the predicted value computed by the model. The R^2 value indicates the quality of the model, where a value ≥ 0.8 is considered as an acceptable range [35]. In our case, the value of R^2 is 0.96, which indicates a good fit. However, according to Hellerstein et al. [35], a larger value of R^2 can also be misleading in cases where data points are grouped together around extreme values. Therefore, to confirm the accuracy of the model, residual analysis plots are often recommended. Such a plot, in the context of our model, can be seen in Fig. 5 where the actual values of the output signal are plotted against the predicted values. It is evident from this plot that apart from few points, all other points are grouped around the diagonal line, which indicates better accuracy of the model.

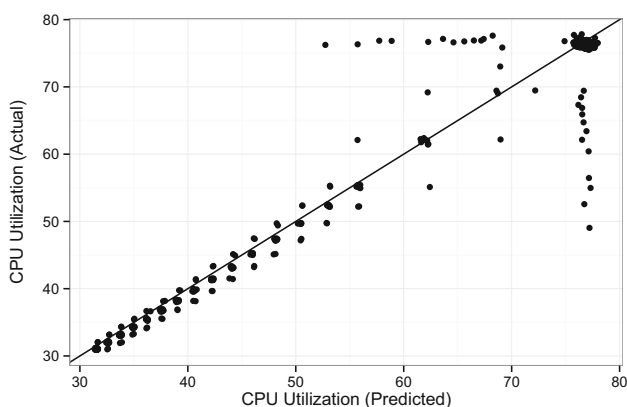


Fig. 5 Comparison of measured output vs predicted output

2.1.4 Controller design

The goal of the controller design step is to select the control law and any required parameters for the *Controller* component of the feedback control methodology. The control law determines the structure of *Controller* component and describes how it will operate [36]. In this paper, we adopt the Integral control law for each of the three employed controllers, i.e. *Lazy*, *Moderate*, and *Aggressive*. The key reasons behind this selection is its simplistic nature and its extensive use for similar problems, e.g. [9, 37–40].

The integral law can be defined using the following equation:

$$u(t) = u(t - 1) + K_i e(t) \quad (6)$$

$u(t)$ represents the new value for control input in time t , $e(t)$ is the control error that represents the difference between the desired and measured output, i.e. $e(t) = y_{ref} - y_t$, and K_i is referred to the integral gain parameter. In this paper, the number of VMs is the control input, whereas CPU utilisation is the measured output. The control error represents the difference between the desired CPU utilisation (i.e. 55%) and the measured CPU utilisation.

The integral gain parameter indicates the aggressiveness of the controller that determines how fast the system will respond. The higher this value, the faster the system will react. However, careful attention is required while deciding the gain of the controller as higher value of the gain parameter could cause oscillation and may lead the system to instability. All the three employed controllers adopt the

same integral law specified by Eq. 6. However, their integral gain parameter is different. The following equations represent each employed controller:

$$u^L(t) = u(t-1) + K_i^L e(t) \quad (7)$$

$$u^M(t) = u(t-1) + K_i^M e(t) \quad (8)$$

$$u^A(t) = u(t-1) + K_i^A e(t) \quad (9)$$

The gains K_i^L , K_i^M and K_i^A are derived using the standard procedure of Root-locus that provides a systematic method to analyse and design feedback controllers. The Root-locus method require the transfer function of the feedback control system. Such a transfer function can be obtained by the corresponding transfer functions of the different components of the feedback loop. In our case, the different components include the integral controller (represented by Eq. 6) and the target system (represented by one of the model earlier described in Sect. 2.1.3). The transfer function of integral controller is given in Eq. 10, whereas the transfer function of the system model of Eq. 4a is provided in Eq. 11. Based on these equations, the transfer function of the entire feedback loop [35] is provided in Eq. 12.

$$C(z) = \frac{zK_i}{z-1} \quad (10)$$

$$G(z) = \frac{0.18}{z-0.89} \quad (11)$$

$$F_R(z) = \frac{0.18K_i z}{z^2 + (0.18K_i - 1.89)z + 0.89} \quad (12)$$

Using the Root-locus method by taking into account the transfer function of feedback loop (Eq. 12), we finalise the following values -0.06 , -0.2 , and -0.5 for K_i^L , K_i^M , and K_i^A gains respectively. The analysis performed using Root-locus indicate that the system remains stable (always reach to equilibrium) and accurate (steady-state error reach to zero) using all the selected gains. The finalised value has a settling time of less than 10 time interval, whereas, the maximum overshoot recorded is less than 15%.

2.2 The switching mechanism: a fuzzy control system

2.2.1 Overview

The deployed application over cloud environment automatically inherits the uncertainty related challenges associated with the cloud environment [41]. Hence the elastic method, responsible for the resource management of the application, has to deal with these challenges. The examples of such uncertainties, summarised from [14, 27, 28, 41, 42], include impreciseness in domain

knowledge, noise in monitoring data, inaccuracies in performance model, delay caused due to actuator operation and unpredictability in workload. Jamshidi et al. [14, 27] and Farokhi et al. [28] stressed the importance of the uncertainty aspects to be taken into consideration while designing the elastic controller. Otherwise, scaling decisions often result in unreliability as the available resources may fail to fulfil the requirements, or may not be cost-effective [28]. However, despite the importance, the implementation of uncertainty in the context of cloud elasticity has not yet been well received [28].

A step in this direction is the work of Jamshidi et al. in [14], where they proposed a fuzzy control system focusing mainly on two issues: (1) The quantitative nature of the *Rules-based* method by introducing the idea of qualitative elasticity rules; and (2) The lack of consideration regarding uncertainty raise due to noise in monitoring input data. Their fuzzy controller introduces elasticity rules of the following nature:

IF *workload* IS *high* AND *responsetime* IS *slow* THEN add 2 VMs

The elasticity engine executes such rules at runtime and makes decision, based on *Arrival rate* and *Response Time*. The output of their controller is the number of VMs to be added or removed. Their approach facilitates a dynamic response based on the aforementioned two parameters by making a scaling decision with different intensity levels, and consequently it improves the static scaling issue of the Rule-based approaches. However, the output (number of VMs) is a pre-defined range of constant integers, and these numbers are set-up based on the experiences of the experts rather than rely on a well-founded design approach. In contrast, our proposed approach relies on the systematic method of control theory to compute the number of VMs. Moreover, our approach is hybrid in nature, i.e. it also incorporates both the performance and capacity based metrics as opposed to their performance based approach only. This paper compliments and extends the work of Jamshidi et al. [14] aiming to develop a fuzzy control system to implement the switching mechanism of the proposed framework. The following subsections explain the design process of this switching mechanism.

2.2.2 The design process

The construction of a fuzzy system involves the following three steps: establishing domain knowledge, designing membership functions and composing fuzzy rules. The details of each of these steps in the context of our switching mechanism are provided below.

Domain knowledge The domain knowledge is concerned with the identification of inputs and outputs of the system. The inputs specify factors of the system that are important

to be considered for decision-making purposes. As mentioned earlier, the proposed method considers three different aspects of the system for decision-making. These aspects are the inputs of the fuzzy system and their brief description are provided below:

1. *Response time* indicates the performance level of the deployed application and is measured as the percentage number of SLO violations (i.e. when *Response time* of a job request > 0.6 s) in the last time unit.
2. *Arrival rate* indicates the workload behaviour in the last time unit and is measured as percentage number of job arrivals. The *System Monitor* component of the proposed method records the number of arrivals in the last time period to identify the intensity of the workload.
3. *Control error* inclusion as an input is the consideration of resource utilisation level into the decision-making. The *Control error* is the difference between the measured and desired CPU utilisation.

These inputs cover performance, disturbance and resource utilisation aspects in the decision-making mechanism. Contrary to the fuzzy controller of Jamshidi et al. in [14] that directly produces the pre-defined constant number of VMs as a scaling decision, the output of our fuzzy system is one of the employed controllers that will be used to compute the scaling decision.

The next step is to define fuzzy set for each input and output (commonly known as fuzzy variables). The fuzzy set of each variable comprises of defining linguistic terms and assigns ranges of values to them. Table 1 provides the definitions of all the linguistic terms for each fuzzy variable

Table 1 Ranges for fuzzy variables

Fuzzy variable	Set member	Range
Workload	Low	0–48.9
	Medium	30.7–67.94
	High	56.41–100
Response time	Desirable	$0-\beta_2$
	Okay	$\beta_1-\beta_4$
	Bad	β_3-100
Control error	Stronger negative (stNeg)	– 20 to – 100
	Weaker negative (weNeg)	– 5 to – 30
	Normal	– 10 to 10
	Weaker positive (wePos)	5–30
	Stronger positive (stPos)	20–100
Controller	No scaling	0–10
	Lazy	11–20
	Moderate	21–30
	Aggressive	31–40

and their corresponding ranges, whereas their brief description is given as follows:

- The linguistic terms and the corresponding ranges for the *Workload* (i.e. Arrival rate) variable are adapted from the work of Jamshidi et al. in [14], where the knowledge base is constructed using domain experts, i.e. architects and administrators. They constructed a fuzzy set of five linguistic terms for *Workload* variable including *Very low*, *Low*, *Medium*, *High* and *Very high*. We reduce them to three to minimise the number of rules, hence reduce the complexity. However, more fine-grained control over resources can be obtained by increasing the number of workload categories or the number of controllers.
- The linguistic terms of *Response time* variable reflect the overall performance objective of the application that can be defined by the SPs. In Table 1, we use symbols β_1 , β_2 , β_3 and β_4 to represent the customisable aspect of these parameters. Jamshidi et al. [14] in contrast, distributed the *Response time* into five categories with the values obtained from domain experts. However, considering that the application performance measurement for different applications is different, the values of the linguistic terms of *Response time* are customisable to reflect the desired performance objective and have to be defined by the SPs. In the current settings of this paper, we adopt the following values for evaluation purposes, i.e. $\beta_1 = 3\%$, $\beta_2 = 5\%$, $\beta_3 = 8\%$, and $\beta_4 = 10\%$.
- The linguistic terms of *Control error* are obtained by distributing the *Control error* measurement into five categories. An increase in these categories can provide more fine-grained control. However, it will also increase the complexity of the proposed method. The ranges of these linguistic terms are obtained using trial and error method, where various experiments are carried out using different ranges.
- The linguistic terms of *Controller* variable are the possible outcomes. These terms depend on the number of controllers, which in this case are three. We also consider one more output, i.e. *No-scaling* that specifies no action is required. The ranges of these linguistic terms are set based on the approach adopted in [43], where no overlapping of the range is required because the final decision represents a range that corresponds to a single output rather than a numerical value.

Membership Functions The next step is to define the membership functions that convert the crisp inputs into the corresponding fuzzy values. The membership function defines the degree of the crisp input against its linguistic variables in the range of 0 to 1. The design of the membership functions, adopted from Jamshidi et al. [14], use

triangular and trapezoidal types of function. These functions have the advantage of being simple and efficient in comparison with other types of membership functions [44]. Figure 6 represents the membership functions of our fuzzy control system.

Fuzzy rules The fuzzy rules describe the relationship between the inputs and outputs of the fuzzy control system. Each fuzzy rule, in this case, determines the type of the controller that makes the scaling decision. The fuzzy rules are made of using fuzzy logic statements and follow the *if-then* pattern. The fuzzy rules of the switching mechanism are made using the linguistic terms of the fuzzy variables explained earlier in Sect. 2.2.2. An example of such a rule is provided below:

IF *arrivalRate* IS *high* AND *responseTime* IS *desirable* AND *controlError* IS *wePos* THEN *controller* IS *lazy*.

In the above example, a *Lazy* controller is selected based on the values of *Arrival rate*, *Response time* and *Control error*. Such rules for an application scenario can be designed using the combination of linguistic terms provided for each parameter in the rule (see Table 1). Such rules can also be tuned for different situations using optimisation approaches. A full list of the rules employed for the experimentation conducted in this paper are provided in Table 2. These rules are designed using the following considerations: (1) Select those rules that react quickly if the application performance is poor; (2) If the application performance is desirable then aims to reduce system running cost; (3) Aim to maintain the CPU utilisation around the desired reference value.

3 Experimentation and computational results I

The experimental environment used for the evaluation is developed using Java language that integrates a well-known cloud simulation environment called *CloudSim* [34]

and an external Java-based library called *JFuzzyLogic* [45]. The following subsections explain the various aspects of experimentation and the obtained computational results.

3.1 Workloads

The commonly used approach to test an auto-scaling methodology is to evaluate its performance against different workloads, based on certain desirable criteria. Gandhi et al. in [46] and Jamshidi et al. in [14] evaluated their proposed elastic methods using workloads that follow different patterns. The key reason of using such an approach is to evaluate and analyse the performance of an elastic method in different scenarios. The workload patterns that they have used include *Quickly varying*, *Slowly varying*, *Dual phase*, *Tri phase*, *Big spike* and *Large variations*. Similarly, Mao and Humphrey [47] used *Stable*, *Cyclic*, *Growing* and *On-off* set of patterns. Each of these patterns represents a different class of applications [26]. This research also adopts the patterns mentioned above to analyse the performance of the proposed method. In this paper, we identify seven different workloads that can be seen from Fig. 7 to represent a single or multiple patterns. Amongst these, one is synthetically generated, whereas the remaining six are derived from the following real Internet-based sources including Wikipedia [48], FIFA World Cup [49] and WITS (Waikato Internet Traffic Storage) [50] project. All the derived workload traces are vertically scaled to a maximum of 60,000 rpm and the number of arrivals on per minute basis is obtained from the count of actual arrivals except for the synthetically generated one. Furthermore, the service time of each job request is randomly generated between 100 and 500 ms to incorporate the stochastic behaviour of the incoming arrivals.

Fig. 6 Membership functions

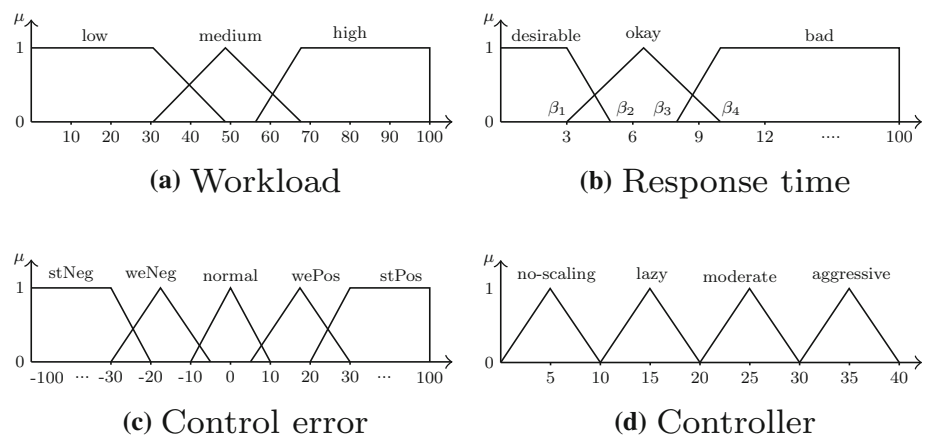
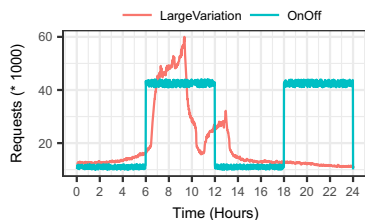
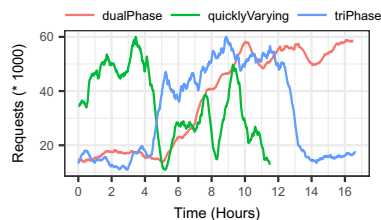
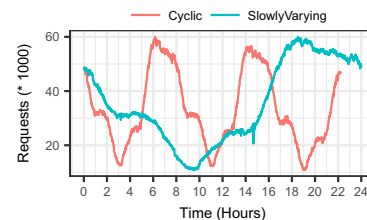


Table 2 All switching elasticity rules

W	RT	CE	No-scaling	Controller		
				Lazy	Moderate	Aggressive
High	Desirable	wePos		✓		
High	Desirable	stPos			✓	
Medium	Desirable	wePos			✓	
Medium	Desirable	stPos				✓
Low	Desirable	wePos			✓	
High	Desirable	stPos				✓
High	Desirable	weNeg		✓		
High	Desirable	stNeg			✓	
High	Okay	weNeg				✓
High	Okay	stNeg				✓
High	Bad	weNeg				✓
High	Bad	stNeg				✓
Medium	Desirable	weNeg		✓		
Medium	Desirable	stNeg			✓	
Medium	Okay	weNeg				✓
Medium	Okay	stNeg				✓
Medium	Bad	weNeg				✓
Medium	Bad	stNeg				✓
Low	Desirable	weNeg		✓		
Low	Desirable	stNeg			✓	
Low	Okay	weNeg		✓		
Low	Okay	stNeg			✓	
Low	Bad	weNeg				✓
Low	Bad	stNeg				✓
–	–	Normal	✓			

**(a)** FIFA Worldcup and On off (Synthetic)**(b)** WITS**(c)** Wikipedia**Fig. 7** Various workloads used for experimentation

3.2 Benchmark approaches/scenarios

3.2.1 Fixed gain feedback controller

We have used *Fixed gain* feedback controller as one of the benchmark methods. The key reason behind is that our proposed method is an extension of such an approach, where we use multiple *Fixed gain* controllers simultaneously. The individual elastic controllers are termed *Lazy*, *Moderate* and *Aggressive* respectively, thus aiming to demonstrate the effect of using the same controllers

independently versus using them collectively as in the proposed framework.

The nature of the individual controllers, i.e. *Lazy*, *Moderate* and *Aggressive*, in general are similar to those used in related elastic methodologies such as [9, 11, 37, 51]. The individual controllers are implemented following the proportional threshold approach of [9], where the *Reference input* is considered as a range rather than a scalar value. This approach avoids the unnecessary oscillations by restricting the controller not to take a decision if the measured output is within a certain range. In this paper,

we consider a range $\pm 10\%$ of *Reference input* (55%), because it is the same as the range of *Normal* linguistic term of *Control error* fuzzy variable used in our proposed switching mechanism.

3.2.2 RightScale: a rule-based approach

The *RightScale* [5] is a 3rd party commercially available auto-scaling approach, which is a *Rule-based* method. In the *RightScale* method, each VM engages in a voting process, where every VM decides whether a scaling decision is required or not. The decision by individual VMs is based on the set of elasticity rules. The implementation of *RightScale* includes the setting of decision threshold value for the voting process. For this purpose the value 51% is used. This represents, if just more than half of the VMs are in favour of the decision then the action will be performed. Otherwise, it will be ignored. Another important aspect of *RightScale* implementation includes the determination of system metric to be used for setting up the rules. For this purpose, we use *CPU utilisation* as a system metric based on its usage as the *Reference input* in the proposed method. The elasticity rules used for the implementation are as following:

For scale up

```
if CPU Utilisation > thrup then
n = n + sa and
do nothing for t seconds
```

For scale down

```
if CPU Utilisation < thrdown then
n = n - sr and
do nothing for t seconds
```

The value used for thr_{up} is 55%, i.e. the desired *Reference input* of our proposed method as we already know, the performance degrades when *CPU utilisation* becomes higher than 55%. The value for thr_{down} obtained by trying different possible values such as (20%, 30% and 40%) and then selected, the value that produces the better result regarding the evaluation criteria (explain in next section). Another important configuration required is the settings of values for s_a and s_r . For this purpose, we use the following four different settings: (1) $s_a = s_r = 2$, (2) $s_a = 2, s_r = 1$, (3) $s_a = 4, s_r = 2$ and (4) $s_a = 10\%, s_r = 5\%$. Lastly, the t in both of the above rules specifies.

3.3 Evaluation criteria

The key objective of implementing cloud elasticity is to improve the utilisation of computational resources whilst maintaining the desired performance of the system and reducing its operational cost. This statement hints on the

fundamental criteria, i.e. *Performance* and *Cost* for the assessment of an auto-scaling mechanism. The brief details for each aspect in the context of this paper is as follows.

1. *Service level objective (SLO) Violations* We consider *Response time* as a criterion to measure the performance of the elastic method. The requirement regarding desired performance objective in cloud computing is defined through SLO specification. In this paper, we consider that each job request of the workload must be completed in the pre-defined desired time, i.e. ≤ 0.6 s. Thus an SLO violation is considered, if the desired *Response time* for a job request has not been achieved.
2. *Cost* The *Cost* refers to the operational cost of the rented VMs. These VMs are used to execute the workload and each VM is associated with a cost per time unit. The total running time of all VMs is recorded for the entire experiment. This includes the time when a VM starts to the time it finishes execution, either as a result of a *Scale-down* action or when the experiment finishes. A rate of 0.013\$ per hour is applied to calculate the final cost based on the Amazon pricing [52] for the VM instances of “t2.micro” type.

3.4 Computational results and analysis

The benchmark methods as well as the proposed methods are implemented into the CloudSim environment. CloudSim is extensively used in the cloud related research activities for modelling and simulation of cloud computing systems and applications. We have used, and extended where necessary, its various functionalities, such as the scheduling strategies, creation and deletion of VMs, etc.

For the experiments, all VMs are identical and are considered as abstract servers, that imitate to serve a specific purpose, e.g. act as web servers. Furthermore, for each particular method, i.e. the benchmark methods and the proposed methods, the following related aspects of the simulation environment remain the same:

- *VM creation* The focus of our proposed method is from the SP perspective, where the main concern is with the management of rented VMs and not the underlying physical hardware that host VMs. Therefore, in this research work, we are not considering aspects like optimal placement of VMs on physical hosts, which is in itself researched as an independent problem. For the simplicity of the implementation, the default allocation and scheduling policies of CloudSim concerning the VM and Host related assignment and execution are used.
- *VM deletion* In the case of scale down operation, the VM with lowest number of jobs is selected to delete.

The action of delete is however not immediate and the deletion process wait until the completion of all jobs.

- *Jobs allocation* Analogous to the VM and Host related allocation and scheduling policies, the assignment of incoming jobs to the already available VMs are handled through a round robin policy.

The computational results obtained from the experimentation can be seen from Fig. 8. In this figure, *rs_21*, *rs_22*, *rs_42* and *rs_pro* represent the four different settings of the *Rightscale* method explained earlier in Sect. 3.2.2. Similarly, *Lazy*, *Mod* and *Agg* refer to the benchmark methods explained in Sect. 3.2.1 and *HS* represents the proposed *Hard switching* method.

The plots in the left column of Fig. 8 present an aggregated view of *Cost* versus *Performance* aspect of the overall experiment for each method. Some of these plots do not show results of few methods. The reason behind is that in such cases, the number of SLO violations were recorded as $\geq 5\%$, i.e. higher than the desirable performance objective. Therefore, those results were not of interest and are excluded to improve the readability of plots. The only exception to this criteria is in the case of On off scenario, where all methods results in $\geq 5\%$ SLO violations, except thle proposed method, i.e. *HS*. The plots in the right column present the corresponding time series view of the number of SLO violations in an hourly basis for the three methods that obtained comparatively better aggregate results. This section briefly discusses each of the applied methods in light of the obtained computational results.

1. *Rightscale* It is observed from the obtained results that some settings of the *Rightscale* method produce better performance in comparison to the other approaches, i.e. *Lazy*, *Mod*, *Agg* and *HS*. However, this better performance is obtained with a very higher cost. Such phenomena are only observed in those scenarios, where transitions in workloads are comparatively smooth, e.g. in the case of *Dual-phase*, *Cyclic* and *Slowly varying* scenarios. In other scenarios where sharp changes occur in workloads, e.g. in the case of *Large variations* and *On-off* scenarios, the performance is comparatively poorer than *HS* and *Moderate* despite being expensive. A key reason behind is the underlying static scaling behaviour of the *Rightscale* method, where a scaling action is performed using a uniform quantity.
2. *Aggressive* It is observed that the aggregated results of performance obtained using the *Aggressive* approach in the case of *Dual phase* and *Quickly varying* scenarios are comparatively better than *HS*. However, the time series analysis of those scenarios indicates that the performance of the system is poor in certain hours specifically when the arrival rate of the workloads is low. The key reason for this behaviour is the

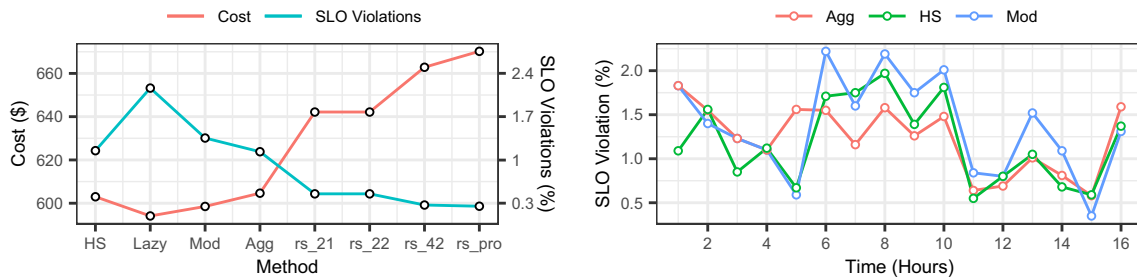
inappropriate scaling intensity that causes a bigger change in some cases, e.g. observe the time series view of *CPU utilisation* in Fig. 9 for the first two hours and 5th hour in the case of *Dual-phase* scenario and 6th hour in the case of *Quickly varying* scenario. The worst situation arises in the case of *Large variations*, where the system resources oscillate when the arrival rate of the workload remains low. This indicates that using a uniform *Aggressive* method at the entire time is not a good choice and could lead the system to an unstable state.

3. *Moderate* The performance of the *Moderate* policy works well in the following two cases. Firstly, where the incoming workload remains stable in a particular region, e.g. the segment after the 8th hour in the case of *Large variation*; Secondly, where the arrival rate changes slowly, e.g. in the case of *Slowly varying*. However, the *Moderate* method performs poorly in comparison to the *Agg* and *HS*, when there are sharp changes in the incoming workloads, e.g. the segment after the 7th hour in the case of *Quickly varying* for the 7th and 19th hours in the case of *On-off* etc.
4. *HS* It is evident that the performance obtained using *HS* in comparison to the above-mentioned methods remains better in all of the following scenarios without having an impact on cost, i.e. *Large variations*, *Tri-phase*, *Cyclic* and *On-off*. The exception is in the cases of *Dual-phase* and *Quickly varying*, where the aggregate performance is slightly poorer than that of the *Aggressive* method. However, considering the time series analysis, the *HS* approach maintains better performance during the entire time. Lastly, in the case of *Slowly varying*, the *HS* and *Moderate* policy have achieved similar performance.

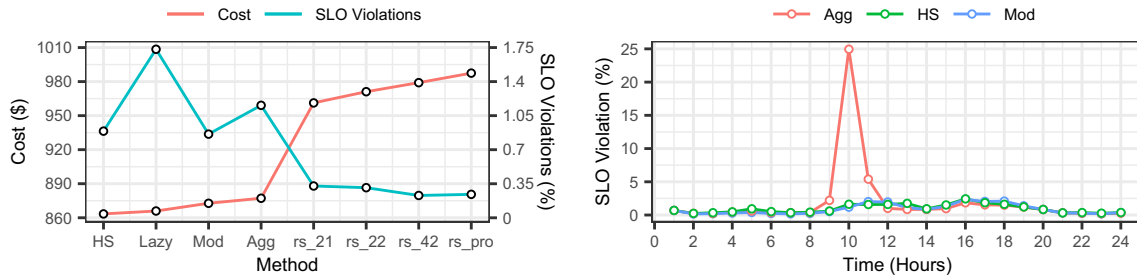
The above discussion indicates that using a uniform fixed policy is unable to cope with changing workload conditions. In contrast, the proposed *Hard switching* consists of the collection of the same policies with an additional switching mechanism result in an improved system performance without an increase in the operational cost.

4 Biologically-inspired elasticity framework: soft switching

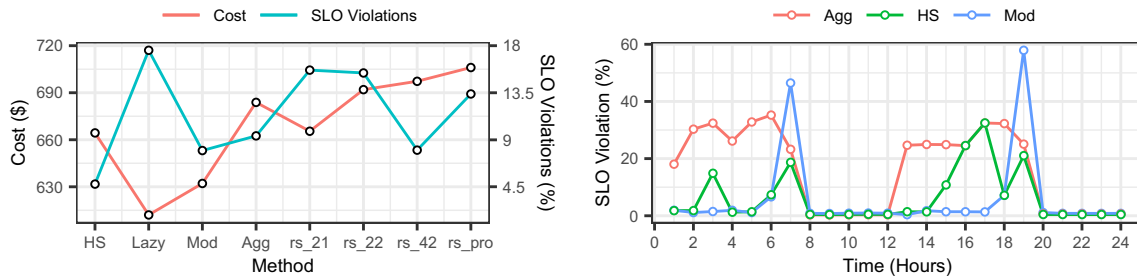
The *hard switching* approach described in previous section has the potential to improve system performance in comparison to the benchmark methods. However, such methodologies are often criticised for their associated unwanted behaviour, termed as bumpy transition, that could lead the system to an oscillatory state [35, 43, 53]. Figure 10 demonstrate the occurrences of such unwanted



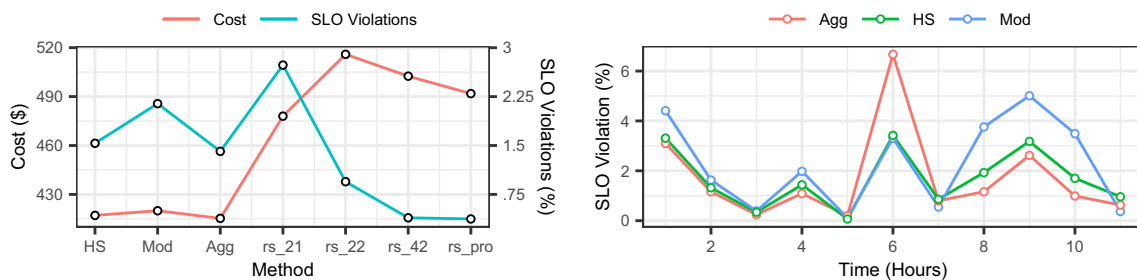
(a) DualPhase



(b) Slowly varying



(c) On off



(d) Quickly varying

Fig. 8 The left column presents the integrated results for each employed workload, whereas, the right column present the number of SLO violation (%) in a per hour basis

behaviour in the results obtained using our *hard switching* (HS) approach. The oscillation of resources may have deteriorating effects on the system performance as well as on the operational cost. It is therefore desirable that the

proposed method should result in smoother transitions to avoid any oscillation.

Soft switching, on the other hand, is an alternative technique used to avoid such unwanted behaviour. Such a

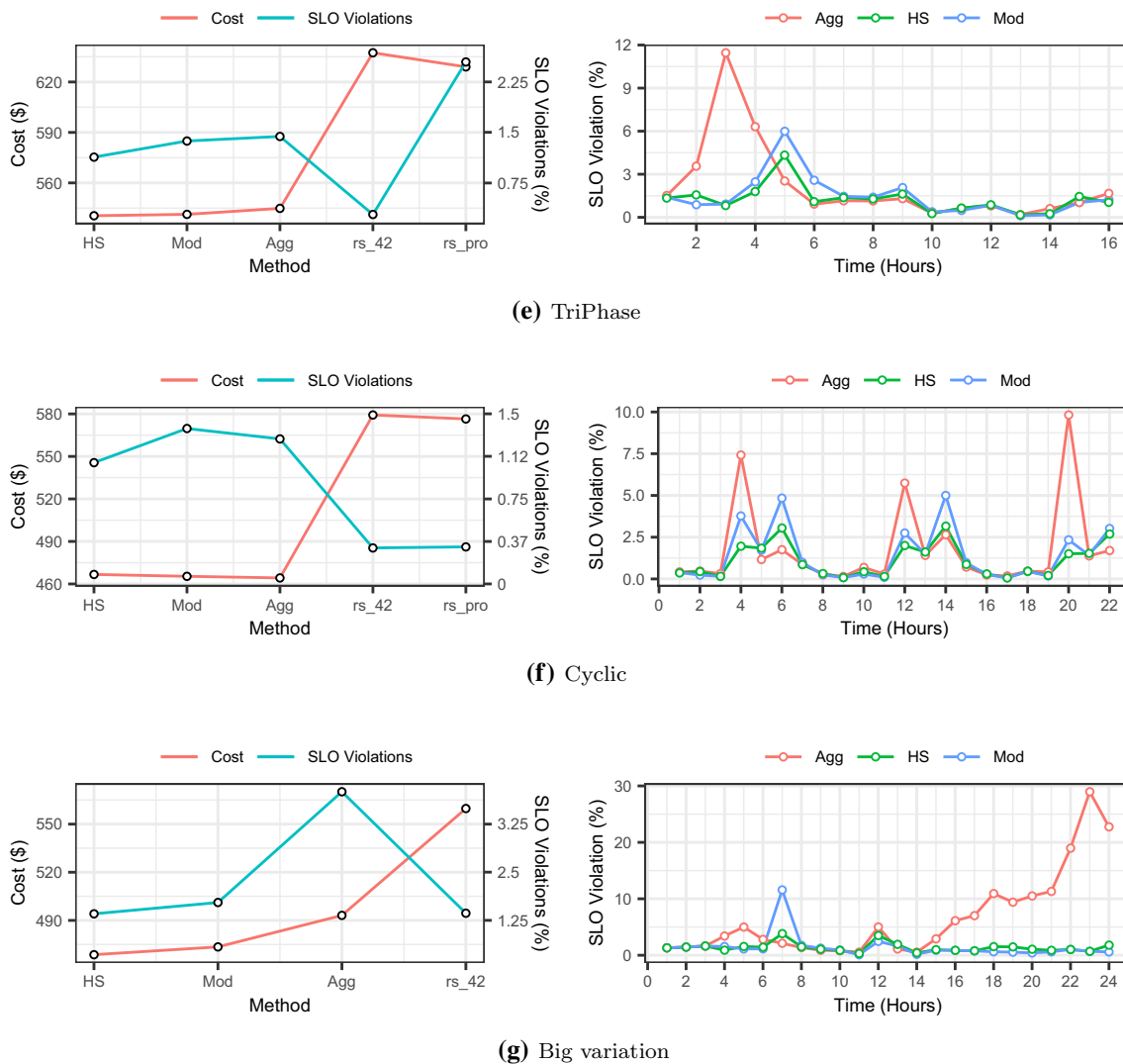


Fig. 8 continued

mechanism in contrast to *hard switching* has the possibility to select multiple actions rather than one best choice. The key benefits of such an approach include: (1) avoidance of singularity and sensitivity problems, (2) improvement of robustness and stability aspects and (3) elimination of chattering issues [54].

This section aim to explore the capabilities of a biologically (cognitive) inspired action selection process to implement *soft switching* behaviour, hence seeking the possibility of more smoother (bumpless) transitions to improve the stability perspective. Formally, an action selection is the process of deciding what to do next from a set of available actions by an agent, based on some knowledge of the internal state and some provided sensory information of the environmental context to best achieve its desired goal [55]. Over a period of time, researchers have learnt that in animal's brain, the problem of action

selection is handled through the use of a central switching mechanism [56, 57]. This mechanism is implemented by a group of subcortical nuclei collectively refers to as Basal Ganglia (BG) [56, 57]. For a functional anatomy of BG, refer to [55]. To incorporate such a mechanism into our framework, we integrate a well established BG based computational model of Gurney et al. [58, 59]. The key advantages of this computational model include its biological plausibility and computational efficiency [60]. The block diagram of the enhanced framework (Soft switching) can be seen from Fig. 11. Comparing this diagram with the *Hard switching* approach, the following three differences can be observed: (1) the integration of the BG component, (2) the output of the FIS component and (3) the final output of the control system. The details of each of these differences are provided in the following subsections:

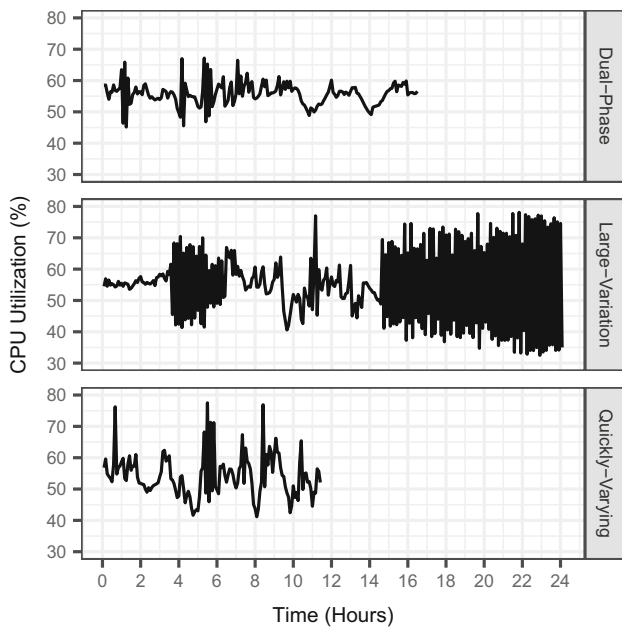


Fig. 9 CPU utilisation using aggressive method

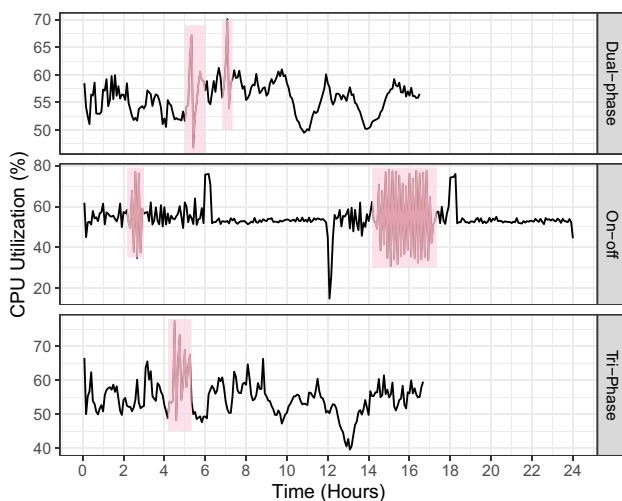


Fig. 10 Time series of CPU Utilization obtained using HS approach

4.1 The BG component

This component integrates the BG based computational model that builds on the functional anatomy of BG. Some examples of such models include [58, 59, 61–63]. Amongst these models, we utilised the computational model proposed in [58, 59]. However, any model can be used as our aim is not to identify the best action selection or biologically-inspired computational model rather to demonstrate the effectiveness of such an approach in the context of the cloud elasticity.

Focusing on Gurney et al. [58, 59] computational model, the brain subsystems send excitatory signals that represent

the behavioural expressions to the BG. Each behavioural expression defines an action in BG and its strength is determined by the salience that represents the activity level of its neural representation. These actions are mediated through the release of inhibitory signals. Thus in each iteration, the functional model accepts a set of salience signals and produces a set of selected and unselected output signals. The functional model can select a maximum of one action (referred as Hard mode), similar to the Hard switching approach earlier described in Sect. 2. Alternatively, the functional model can also have the possibility to select multiple actions (referred as Soft mode). In this research work, we are interested in the Soft mode of the functional model, where it result in the selection of multiple actions. For a detailed description of the functional model refers to [58, 59].

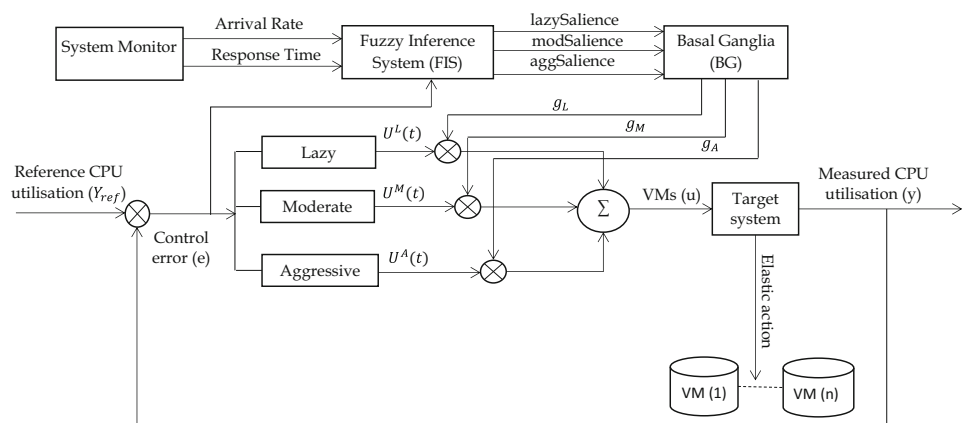
The BG component, shown in Fig. 11, accepts three inputs namely *lazySalience*, *modSalience* and *aggSalience*. These inputs represent the strength of selection for each controller (depicting as action). The values for these salience signals are computed by the FIS (details provided in the next section).

4.2 The modified FIS

The BG based computational model requires salience signals as inputs. Thus the first issue to be dealt with is the generation of salience signals. The method to generate the salience signals can make use of system's internal state, various performance metrics or available sensory information [60]. Therefore, we have extended the FIS, used as a switching mechanism in previous section, to generate the inputs (salience signals) for the BG component of the framework. The inputs of the modified FIS remains the same, i.e. *Workload*, *ResponseTime* and *ControlError*. However, the output is changed from one, i.e. *Controller* to three *lazySalience*, *modSalience* and *aggSalience*. Each of these outputs represents the salience strengths for the selection of each of the three controllers. The details of the changes carried out are as following:

1. *Membership functions* The inputs of the modified FIS do not change, and therefore the corresponding membership functions of the input fuzzy variables remain the same. However, the output is changed, therefore, the *Controller* membership function is replaced with three new membership functions, i.e. one for each newly introduced output. Similar to the *Controller* membership function, we have used the basic triangular type for all the outputs. The membership function for each salience signal variable is of the form shown in Fig. 12.

Fig. 11 Biologically-inspired soft switching framework



2. *Fuzzy rules* The fuzzy rules are responsible to generate the saliency inputs. The fuzzy rules described in previous section are revised accordingly. The inputs of the rules are the same as in the case of *Hard switching*. However, the output can be formed using the linguistic terms (*weak, average and strong*) for each saliency. An example of such rule is provided below:

IF *arrivalRate* IS *medium* AND *responseTime* IS *desirable* AND *controlError* IS *stPos* THEN *modSaliency* IS *strong* AND *aggSaliency* IS *average*

4.3 Derivation of final output

As mentioned earlier, in terms of the adopted functional model, we are interested in the mode, where it result in the possibility of multiple actions section. Hence the final decision, i.e. the number of VMs, will be derived using the output signals returned by the BG component and the outputs of the individual controllers. The following equation represents this derivation.

$$u_t = \frac{(u^L(t) * g_L) + (u^M(t) * g_M) + (u^A(t) * g_A)}{g} \quad (13)$$

The u_t in the above equation represent the final decision. $u^L(t)$, $u^M(t)$ and $u^A(t)$ represents the output of individual controllers, i.e. *lazy, moderate and aggressive* respectively. These outputs are computed as per the equations described in Sect. 2.1.4, i.e. Eqs. 7, 8 and 9 respectively. Whereas, g_L ,

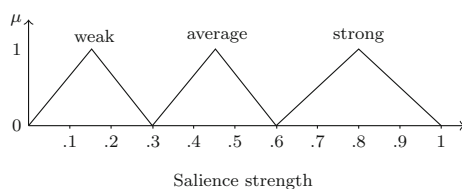


Fig. 12 Membership functions for each saliency variable, i.e. lazy, moderate and aggressive

g_M and g_A are the output signals returned by the BG component. The values of these signals lies between 0 to 1 and they signify the proportion of each action. The denominator g represents the number of those output signals with a value higher than zero. However, it is not always the case that more than one controller to be selected.

5 Computational results II

We have used the same experimental settings and scenarios (i.e. gains for controllers and workloads) as in the case of *Hard switching* to evaluate the *Soft switching* approach. It is already discussed previously that the *Hard switching* achieves better results compared to the benchmark methods. Therefore, in this section, we only compare *Soft switching* to *Hard switching* approach. We present and discuss the obtained computational results in the following two aspects:

5.1 Performance

Figure 13 shows the aggregated view of the results obtained using both the approaches, i.e. *Soft switching* and *Hard switching*. These approaches are represented as *SS* and *HS* respectively in the reported results. Considering the number of SLO violations, it is evident that the *SS* approach has obtained, lower number of SLO violations to that of *HS*, in each employed scenarios. On the other hand, the comparison of the cost perspective indicate the similar level of spending by both approaches, i.e. *SS* and *HS*. This demonstrates that the *SS* approach results in better performance compared to that of *HS* without increasing the operational cost of the system.

Figure 14 provides an insight into the performance of both the approaches on an hourly basis. Each plot of this diagram represents the result for every employed workload

scenario. The analysis of these plots hints the following: (1) The performance obtained, in the cases of each scenario, using *SS* approach in almost every hour is either similar to that of *HS* or comparatively better. This indicates a higher potential to maintain better performance during the entire period of the experiment. (2) The *SS* and *HS* approach behave almost similarly in scenarios, when there are sharp increases in workload, e.g. the 6th hour in the case of *Cyclic*, the 7th hour in the case of *Large variations* and the 7th and 19th hours in the case of *On-off* scenarios. (3) The *SS* approach performs comparatively better when the arrival rate of the workload remains low, e.g. initial 5 hours period in the case of *Dual-phase* and the hours from 15th to 18th in the case of *On-off*. This indicates that at the time of low workload, the decision of *HS* affects the performance more due to its best controller selection strategy in comparison to that of *SS* approach.

5.2 Oscillatory behaviour

The results presented in the previous section demonstrate the effectiveness of the *Soft switching* approach regarding the improvement of the overall performance. This section discusses the possibility of reducing the likelihood of bumpy transitions and oscillation in comparison to the *Hard switching* approach.

Figure 15 presents the measured CPU utilisation recorded for *HS* and *SS* approaches. The analysis of these plots hints on the following insights:

1. Focusing on the highlighted parts of *HS* plot for *On-off* scenario clearly hint at the presence of oscillations at two occasions, i.e. in the 3rd hours and in the 15th to

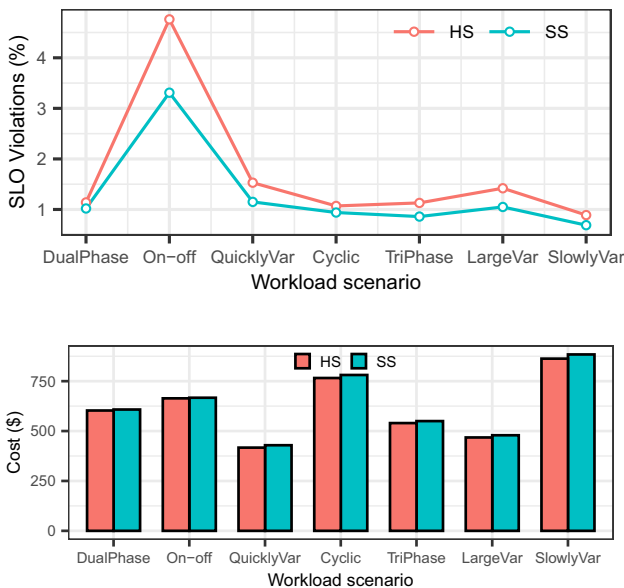


Fig. 13 Aggregated view (HS vs SS)

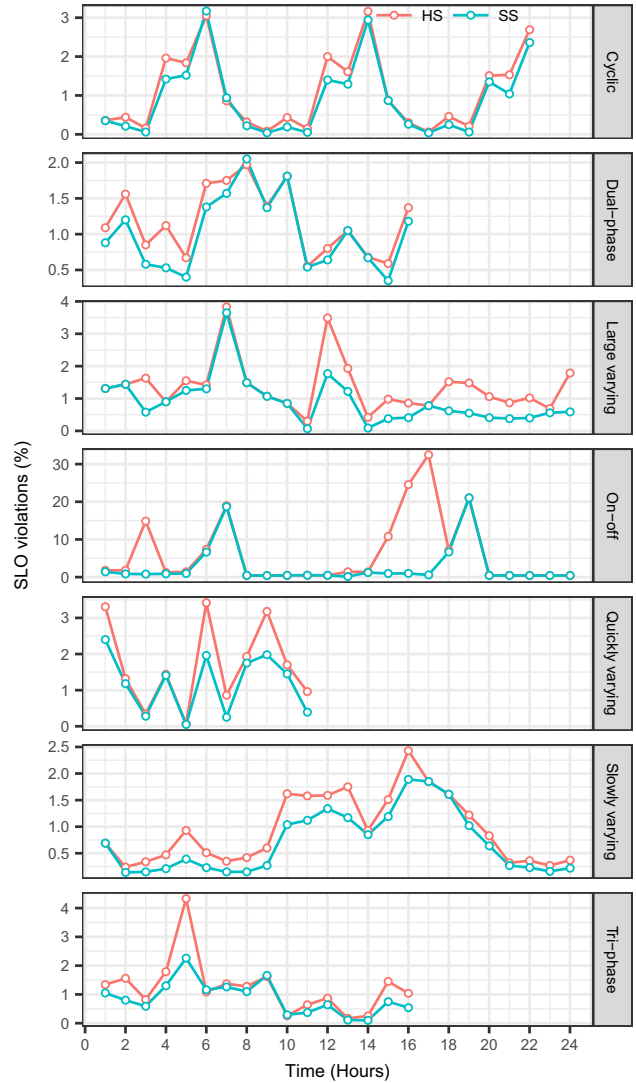


Fig. 14 Time series view of SLO violation

- 16th hours. On the other hand, using *SS* approach, no such oscillations can be seen in the corresponding *SS* plot that demonstrates clear improvements.
2. Considering the *DualPhase* scenario shown in Fig. 15, it is clear that there is no oscillation using both the approaches. However, the highlighted part in the case of *HS* shows some bumpy transitions, i.e. in the 6th and 8th hours. Whereas in the case of the *SS* approach, the intensity of these bumpy transitions is reduced as is evident by visual inspection of both plots. Moreover, the variance of *CPU utilisation* measurements of 3 h, i.e. from 6th to 8th is calculated for both cases. These calculations are recorded as 12.84 and 15.24 for the *SS* and *HS* respectively. This demonstrates that the *SS* results in fewer variations compared to that of *HS* in those 3 h. Similar results can be seen for the scenario of

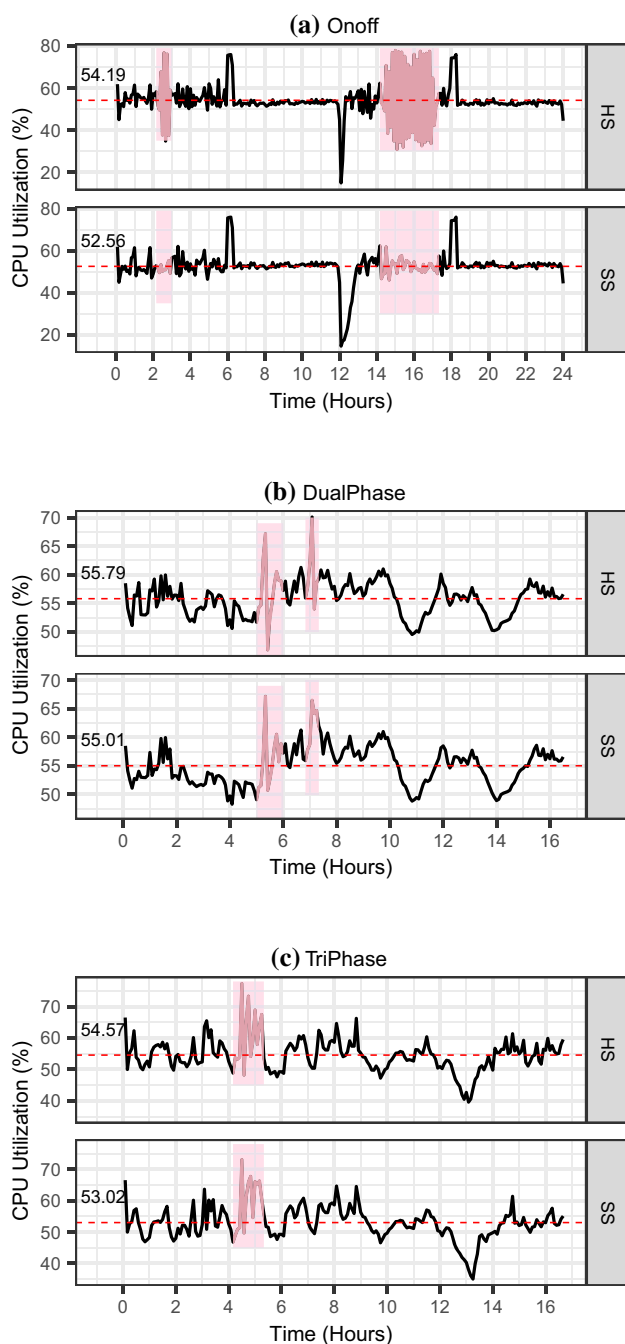


Fig. 15 CPU utilisation (HS vs SS)

TriPhase, where the variation in the case of *SS* from 4th to 6th is fewer than that of *HS*.

- The red dashed line in each plot of Fig. 15 represents the mean *CPU utilisation* obtained using the respective methods in each corresponding scenario. In all of the given three scenarios, the mean obtained using *SS* approach is comparatively less than that of *HS*, e.g. in the case of *On-off*, the means are 52.56 and 54.19 recorded using *SS* and *HS* respectively. This

demonstrates that the *SS* approach is comparatively better and maintains the *CPU utilisation* below 55% more often than the *HS* approach.

In light of the above discussions, we can claim that the BG based *Soft switching* approach has higher potential to reduce the number of SLO violations, hence, result in better system performance. Moreover, compared with the *HS* approach, it has demonstrated the possibility of reducing the likelihood of bumpy transitions and oscillatory behaviour. The intuitive explanation for this improvement is the integration of controllers, (shown in Eq. (13)), in a biologically-inspired fashion augmented with the BG process that facilitates the natural selection of actions, hence result in less 'bumping' at switching time [64]. Moreover, the computational model of [58, 59] in particular is successfully validated to avoid oscillation [62].

6 Related work and discussion

The proposed elasticity methods are developed using control-theoretical based multiple controllers and fuzzy control system. This synergy, on one hand, enables us to address the inherent uncertainty related issues of a cloud environment using the fuzzy control system. On the other hand, the systematic design of model-based feedback controllers helps in strengthening the reliability of the system. In this paper, we chose to address the cloud elasticity from a Service Providers (SPs) perspective. The key motivations behind this choice are that the cloud based applications are subject to varying workload conditions, and the Cloud Providers (CPs) lack control and visibility regarding application performance aspects that make it difficult to perform efficient scaling decisions [65]. In contrast, the SPs have full control and visibility of cloud resources using monitoring and management APIs provided by CPs, as well as an up-to-date knowledge of an application status using custom or 3rd party tools. The proposed methods are hybrid in nature, and consider application level metric (Response time) as well as system level metric (CPU utilisation). Additionally, we consider the *Arrival rate* that represents the incoming workload intensity level into the decision making process. The consideration of these three parameters empower the proposed methodology to make an informed scaling decision, as opposed to the majority of the existing related approaches that either rely on application level [14, 31] or system level metrics [11, 32, 33].

The existing *Rule-based* solutions in general are prevalent due to their intuitive, simplistic and, more importantly, commercial availability factors [66]. Such approaches [4–8] are easy to design and well understood by

the system designers and administrators alike. However, such methods lack a formal systematic design process as they are designed based on previous experiences or applying a trial and error approach [66, 67]. Moreover, they are criticised for the difficulty in setting-up various thresholds of the rules and their inability to cope with the changing environment behaviour [14, 26]. This is evident from the configurations and results of the *RightScale* approach discussed in Sect. 3.

The feedback control solutions [9–12, 33, 37, 68] follow the fixed gain design principle of control theory. Such fixed gain methodologies in general work well for systems that are subject to stable or slowly varying workload conditions [67]. However, due to the lack of adaptive behaviour at runtime, the performance suffers in scenarios where the operating conditions change quickly or when the environmental conditions and configuration spaces are too wide to be explored effectively [25]. The lack of adaptivity issue has been addressed by incorporating online learning algorithms such as the use of linear regression [69], optimisation [70], Kalman filter [71] and reinforcement learning [72]. In general, such adaptive control methodologies have the ability to modify themselves to the changing behaviour in the system environment that make them suitable for systems with changing workload conditions. However, they are also criticised for the additional computational cost caused due to the online learning [26], their associated risk of reducing the quality assurance of the resulted system, and the impossibility of deriving a convergence or stability proof [25]. Moreover, they are unable to cope with sudden changes in the workloads.

Al-Shishtawy and Vlassov [73] addressed the elasticity problem using a two-level approach, where they utilised a combination of an Model Predictive Controller (MPC) based feedforward control solution and a Proportional Integral (PI) based feedback control method. Using such an approach, the feed-forward method follows a predictive approach that takes scaling decisions for a longer time in advance; whereas the feedback method is responsible for making gradual changes in a reactive style. Such two-step hybrid control solutions are effective; however, currently our focus is on the efficiency of elastic solution implemented at the 2nd level that follows a reactive strategy. Al-Shishtawy and Vlassov [73] utilised a fixed gain PI feedback controller that suffers from various issues discussed earlier in this section, whereas the approach adopted in this thesis uses multiple fixed gain controllers. Wang et al., Kjaer et al. [74–76] followed a similar approach, i.e. the combination of feed-forward and feedback. However, they have focused on vertical elasticity.

The following proposals have also adopted a similar approach as employed in this paper. For example, Grimaldi et al. [32] used a PID gain scheduling. Their gain scheduler

is an optimal controller that derives the gains using an optimisation based tuning procedure. The key issues of such an approach are similar to that of an adaptive methods discussed earlier in this section. Saikrishna et al., Qin and Wang, and Taneli et al. [77–79] followed a Linear Parameter Varying (LPV) approach. CPU utilisation is considered as the single scheduling parameter by Saikrishna et al. [77], whereas Qin and Wang, and Taneli et al. [78, 79] rely on arrival rate and service rate. Patikirikorala et al. [31] followed a MMST based control solution. Their method use two different operating regions and consist of two different fixed gain controllers with an if-else switching that is based on *Response time* only. Saikrishna et al. [80], in contrast, used ten distinct operating regions and *Arrival rate* as a switching signal.

Jamshidi et al. [14, 15] highlighted the uncertainty related issues and the idea of qualitative elasticity rules using a fuzzy control system to address the issues of *Rule-based* approach. The inputs to their method consist of *Arrival rate* and *Response time*, whereas the output is the number of VMs to be added or removed. Their approach facilitates a dynamic response based on the aforementioned two parameters by making a scaling decision with different intensity level, and consequently it helps avoid the static scaling issue of the *Rule-based* approaches. However, the output (number of VMs) are a pre-defined range of constant integers, and it is not clear how these numbers are set-up. Therefore, it creates similar problems to that of the *Rule-based* approach, i.e. difficulty in setting-up threshold values of rules and lack of a well-founded design approach. On the other hand, machine learning based control solutions that utilise either reinforcement learning [19, 20] or neural networks [81, 82] provide high levels of flexibility and adaptivity. However, such flexibility and adaptivity come at the cost of long training delays, poor scalability, slower convergence rate, and the impossibility of deriving stability proof [25, 26, 83, 84].

It is concluded from the above discussion that the different elastic controllers due to their underlying implementation techniques have different pros and cons, hence there is no best solution and the choice of selecting suitable approaches depends on the requirements [25]. The research work carried out in this paper advocates the idea of a fixed-adaptive approach (also referred to as hybrid by Gambi et al. [25]) in contrast to either completely fixed or fully adaptive methods. The proposed elastic methodologies are implemented using the combination of the model-based control-theoretical approach and the knowledge based fuzzy control system. This combination, in comparison with the existing fixed-adaptive methods [31, 32, 77–80], addresses the uncertainty related issues and enables us to provide qualitative elasticity rules as well.

7 Conclusion

This paper investigates the horizontal elasticity problem from the *SPs* perspective and proposes biologically-inspired auto-scaling solutions. The proposed elastic methods follow a *Reactive* triggering approach, target *Web applications*, and aim to maintain the desired performance level whilst reducing operational cost. The proposed methods are implemented using a *Control theoretical feedback technique* and a *Fuzzy control system*. The proposed approach integrates a functional model of basal ganglia (BG) that augments the methodology to select the right set of controllers in a natural biologically plausible way thus reducing the likelihood of oscillation and enhancing the stability perspective of auto-scaling. We evaluate the proposed methodology using a large set of different real workload patterns against some of the existing elasticity methods. The experimental results demonstrate that the biological inspired method performs better in both evaluation perspective (i.e. performance and cost) than all other approaches. Moreover, the *Soft switching* method reduces the bumpy transitions and oscillatory behaviour observed using the proposed *Hard switching* approach, thus having the potential to increase the stability of underlying system. In future, we aim to extend the developed framework in the following ways: (1) a detailed theoretical convergence and stability analysis is required to formally evaluate the proposed approach against other state-of-the-art approaches, (2) enhancement of switching rules to learn at runtime and (3) to explore the possibility of enhancing the framework by incorporating the vertical elasticity as well.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

- Almeida, V., Arlitt, M., Rolia, J.: Analyzing a web-based system's performance measures at multiple time scales. *SIGMETRICS Perform. Eval. Rev.* **30**(2), 3–9 (2002)
- Arlitt, M., Jin, T.: A workload characterization study of the 1998 world cup web site. *IEEE Netw.* **14**(3), 30–37 (2000)
- Herbst, N.R., Kounev, S., Reussner, R.: Elasticity in cloud computing: what it is, and what it is not. In: 10th International Conference on Autonomic Computing, pp. 23–27 (2013)
- Amazon: Amazon auto scaling (2015)
- Rightscale: Set up autoscaling using alert escalations (2015)
- Han, R., Guo, L., Ghanem, M.M., Guo, Y.: Lightweight resource scaling for cloud applications. In: 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), pp. 644–651. IEEE (2012)
- Koperek, P., Funika, W.: Dynamic business metrics-driven resource provisioning in cloud environments. In: *Parallel Processing and Applied Mathematics*, pp. 171–180 (2012)
- Hasan, M.Z., Magana, E., Clemm, A., Tucker, L., Gudreddi, S.L.D.: Integrated and autonomic cloud resource scaling. In: *Proceedings of the 2012 IEEE Network Operations and Management Symposium, NOMS 2012*, pp. 1327–1334 (2012)
- Lim, H.C., Babu, S., Chase, J.S., Parekh, S.S.: Automated control in cloud computing: challenges and opportunities. In: *Proceedings of the 1st Workshop on Automated Control for Datacenters and Clouds*, pp. 13–18. ACM (2009)
- Arman, A., Al-Shishtawy, A., Vlassov, V.: Elasticity controller for Cloud-based key-value stores. In: *Proceedings of the International Conference on Parallel and Distributed Systems—ICPADS*, pp. 268–275 (2012)
- Gergin, I., Simmons, B., Litoiu, M.: A decentralized autonomic architecture for performance control in the cloud. In: *Proceedings—2014 IEEE International Conference on Cloud Engineering, IC2E 2014*, pp. 574–579 (2014)
- Ashraf, A., Byholm, B., Porres, I.: CRAMP: cost-efficient resource allocation for multiple web applications with proactive scaling. In: *CloudCom 2012—Proceedings: 2012 4th IEEE International Conference on Cloud Computing Technology and Science*, pp. 581–586 (2012)
- Ullah, A., Li, J., Shen, Y., Hussain, A.: A control theoretical view of cloud elasticity: taxonomy, survey and challenges. *Clust. Comput.* **21**(4), 1735–1764 (2018)
- Jamshidi, P., Ahmad, A., Pahl, C.: Autonomic resource provisioning for cloud-based software. In: *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-managing Systems*, pp. 95–104. ACM (2014)
- Jamshidi, P., Sharifloo, A., Pahl, C., Arabnejad, H., Metzger, A., Estrada, G.: Fuzzy self-learning controllers for elasticity management in dynamic cloud architectures. In: *Proceedings—2016 12th International ACM SIGSOFT Conference on Quality of Software Architectures, QoSA 2016*, pp. 70–79 (2016)
- Zhang, Q., Zhani, M.F., Zhang, S., Zhu, Q., Boutaba, R., Hellerstein, J.L.: Dynamic energy-aware capacity provisioning for cloud computing environments. In: *Proceedings of the 9th International Conference on Autonomic Computing*, pp. 145–154 (2012)
- Zhang, Q., Zhani, M.F., Boutaba, R., Hellerstein, J.L.: Dynamic heterogeneity-aware resource provisioning in the cloud. *IEEE Trans. Cloud Comput.* **2**(1), 14–28 (2015)
- Ghanbari, H., Simmons, B., Litoiu, M., Barna, C., Iszlai, G.: Optimal autoscaling in a IaaS cloud. In: *Proceedings of the 9th International Conference on Autonomic Computing—ICAC '12 2(i)*, 173 (2012)
- Liu, J., Zhang, Y., Zhou, Y., Zhang, D., Liu, H.: Aggressive resource provisioning for ensuring QoS in virtualized environments. *IEEE Trans. Cloud Comput.* **02**(03), 119–131 (2014)
- Cheng-Zhong, X., Rao, J., Xiangping, B.: URL: a unified reinforcement learning approach for autonomic cloud management. *J. Parallel Distrib. Comput.* **72**(2), 95–105 (2012)
- Ali-Eldin, A., Tordsson, J., Elmroth, E.: An adaptive hybrid elasticity controller for cloud infrastructures. In: *Network Operations and Management Symposium (NOMS), 2012 IEEE*, pp. 204–212. IEEE (2012)
- Ranjan, R., Wang, L., Zomaya, A.Y., Georgakopoulos, D., Sun, X.-H., Wang, G.: Recent advances in autonomic provisioning of

- big data applications on clouds. *IEEE Trans. Cloud Comput.* **3**(2), 101–104 (2015)
23. Singh, S., Chana, I.: QoS-aware autonomic resource management in cloud computing: a systematic review. *ACM Comput. Surv. (CSUR)* **48**(3), 42 (2015)
 24. Gmach, D., Rolia, J., Cherkasova, L., Kemper, A.: Resource pool management: Reactive versus proactive or let's be friends. *Comput. Netw.* **53**(17), 2905–2922 (2009)
 25. Gambi, A., Toffetti, G., Pezze, M.: Assurance of self-adaptive controllers for the cloud. In: *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7740 LNCS, pp. 311–339 (2013)
 26. Lorido-Botran, T., Miguel-Alonso, J., Lozano, J.A.: A review of auto-scaling techniques for elastic applications in cloud environments. *J. Grid Comput.* **12**(4), 559–592 (2014)
 27. Jamshidi, P., Pahl, C., Mendonça, N.C.: Managing uncertainty in autonomic cloud elasticity controllers. *IEEE Cloud Comput.* **3**(3), 50–60 (2016)
 28. Farokhi, S., Jamshidi, P., Brandic, I., Elmroth, E.: Self-adaptation challenges for cloud-based applications: a control theoretic perspective. In: *10th International Workshop on Feedback Computing*, Seattle (2015)
 29. Hussain, A., Abdullah, R., Yang, E., Gurney, K.: An intelligent multiple-controller framework for the integrated control of autonomous vehicles. In: *Advances in Brain Inspired Cognitive Systems*, pp. 92–101. Springer (2012)
 30. Filieri, A., Maggio, M., Angelopoulos, K., D'Ippolito, N., Gerostathopoulos, I., Hempel, A.B., Hoffmann, H., Jamshidi, P., Kalyvianaki, E., Klein, C., Krikava, F., Misailovic, S., Papadopoulos, A.V., Ray, S., Sharifloo, A.M., Shevtsov, S., Ujma, M., Vogel, T.: Software engineering meets control theory. In: *Proceedings—10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2015*, pp. 71–82 (2015)
 31. Patikirikorala, T., Colman, A., Han, J.: 4M-Switch: multi-mode-multi-model supervisory control framework for performance differentiation in virtual machine environments. In: *2014 10th International Conference on Network and Service Management (CNSM)*, pp. 145–153. IEEE (2014)
 32. Grimaldi, D., Persico, V., Pescapé, A., Salvi, A., Santini, S.: A feedback-control approach for resource management in public clouds. In: *Global Communications Conference (GLOBECOM), 2015 IEEE*, pp. 1–7. IEEE (2015)
 33. Barna, C., Fokaefs, M., Litou, M., Shtern, M., Wigglesworth, J.: Cloud adaptation with control theory in industrial clouds. In: *2016 IEEE International Conference on Cloud Engineering Workshop (IC2EW)*, pp. 231–238. IEEE (2016)
 34. Calheiros, R.N., Ranjan, R., Beloglazov, A., De Rose, C.A.F., Buyya, R.: CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Softw. Pract. Exp.* **41**(1), 23–50 (2011)
 35. Hellerstein, J.L., Diao, Y., Parekh, S., Tilbury, D.M.: *Feedback Control of Computing Systems*. Wiley, Hoboken (2004)
 36. Parekh, S., Gandhi, N., Hellerstein, J., Tilbury, D., Jayram, T., Bigus, J.: Using control theory to achieve service level objectives in performance management. *Real-Time Syst.* **23**(1–2), 127–141 (2002)
 37. Lim, H.C., Babu, S., Chase, J.S.: Automated control for elastic storage. In: *Proceedings of the 7th International Conference on Autonomic Computing*, pp. 1–10. ACM (2010)
 38. Padala, P., Shin, K.G., Zhu, X., Uysal, M., Wang, Z., Singhal, S., Merchant, A., Salem, K.: Adaptive control of virtualized resources in utility computing environments. In: *ACM SIGOPS Operating Systems Review*, vol. 41, pp. 289–302. ACM (2007)
 39. Zhu, X., Wang, Z., Singhal, S.: Utility-driven workload management using nested control design. In: *American Control Conference*, 2006, p. 6. IEEE (2006)
 40. Dawoud, W., Takouna, I., Meinel, C.: Elastic VM for cloud resources provisioning optimization. In: *Communications in Computer and Information Science*, vol. 190 CCIS, pp. 431–445 (2011)
 41. Lu, Q., Xu, X., Zhu, L., Bass, L., Li, Z., Sakr, S., Bannerman, P.L., Liu, A.: Incorporating uncertainty into in-cloud application deployment decisions for availability. In: *2013 IEEE Sixth International Conference on Cloud Computing (CLOUD)*, pp. 454–461. IEEE (2013)
 42. Papadopoulos, A.V.: Design and performance guarantees in cloud computing: challenges and opportunities. In: *10th International Workshop on Feedback Computing* (2015)
 43. Abdullah, R., Hussain, A., Warwick, K., Zayed, A.: Autonomous intelligent cruise control using a novel multiple-controller framework incorporating fuzzy-logic-based switching and tuning. *Neurocomputing* **71**(13), 2727–2741 (2008)
 44. Passino, K.M., Yurkovich, S., Reinfrank, M.: *Fuzzy Control*, vol. 42. Addison-wesley, Menlo Park, CA (1998)
 45. Cingolani, P., Alcalá-Fdez, J.: jFuzzyLogic: a robust and flexible fuzzy-Logic inference system language implementation. In: *FUZZ-IEEE*, pp. 1–8. Citeseer (2012)
 46. Gandhi, A., Harchol-Balter, M., Raghunathan, R., Kozuch, M.A.: Autoscale: dynamic, robust capacity management for multi-tier data centers. *ACM Trans. Comput. Syst. (TOCS)* **30**(4), 14 (2012)
 47. Mao, M., Humphrey, M.: Auto-scaling to minimize cost and meet application deadlines in cloud workflows. In: *2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pp. 1–12. IEEE (2011)
 48. Wikibench: Wikipedia access traces (2009)
 49. Internet Traffic Archive. Worldcup 1998 Web trace (2015)
 50. WAND: WITS: Waikato Internet Traffic Storage (2017)
 51. Ashraf, A., Byholm, B., Lehtinen, J., Porres, I.: Feedback control algorithms to deploy and scale multiple web applications per virtual machine. In: *2012 38th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*, pp. 431–438 (2012)
 52. Amazon: Amazon EC2 pricing (2015)
 53. Qin, G., Duan, Z., Wen, G., Yan, Y., Jiang, Z.: An improved anti-windup bumpless transfer structures design for controllers switching. *Asian J. Control* **16**(4), 1245–1251 (2014)
 54. Lyshevski, S.E.: *Control Systems Theory with Engineering Applications*. Springer, New York (2012)
 55. Tony Prescott, M.: *Action Selection* (2008)
 56. Redgrave, P., Prescott, T.J., Gurney, K.: The basal ganglia: a vertebrate solution to the selection problem? *Neuroscience* **89**(4), 1009–1023 (1999)
 57. Prescott, T.J., Redgrave, P., Gurney, K.: Layered control architectures in robots and vertebrates. *Adapt. Behav.* **7**(1), 99–127 (1999)
 58. Gurney, K., Prescott, T.J., Redgrave, P.: A computational model of action selection in the basal ganglia. I. A new functional anatomy. *Biol. Cybern.* **84**(6), 401–410 (2001)
 59. Gurney, K., Prescott, T.J., Redgrave, P.: A computational model of action selection in the basal ganglia. II. Analysis and simulation of behaviour. *Biol. Cybern.* **84**(6), 411–423 (2001)
 60. Yang, E., Hussain, A., Gurney, K.: A basal ganglia inspired soft switching approach to the motion control of a car-like autonomous vehicle. *Adv. Brain Inspired Cogn. Syst.* **7888**, 245–254 (2013)
 61. Gurney, K.N.: Reverse engineering the vertebrate brain: methodological principles for a biologically grounded

- programme of cognitive modelling. *Cogn. Comput.* **1**(1), 29–41 (2009)
62. Girard, B., Tabareau, N., Pham, Q.-C., Berthoz, A., Slotine, J.-J.: Where neuroscience and dynamic system theory meet autonomous robotics: a contracting basal ganglia model for action selection. *Neural Netw.* **21**(4), 628–641 (2008)
 63. Mandali, A., Rengaswamy, M., Srinivasa Chakravarthy, V., Moustafa, A.A.: A spiking basal ganglia model of synchrony, exploration and decision making. *Front. Neurosci.* **9**, 191 (2015)
 64. Yang, E., Hussain, A., Gurney, K.: Neurobiologically-inspired soft switching control of autonomous vehicles. In: *Advances in Brain Inspired Cognitive Systems*, pp. 82–91. Springer (2012)
 65. Gandhi, A., Dube, P., Karve, A., Kochut, A., Zhang, L.: Adaptive, model-driven autoscaling for cloud applications. *ICAC* **14**, 57–64 (2014)
 66. Ghanbari, H., Simmons, B., Litoiu, M., Iszlai, G.: Exploring alternative approaches to implement an elasticity policy. In: 2011 IEEE International Conference on Cloud Computing (CLOUD), pp. 716–723. IEEE (2011)
 67. Patikirikorala, T., Colman, A.: Feedback controllers in the cloud. In: *Proceedings of APSEC* (2010)
 68. Heo, J., Zhu, X., Padala, P., Wang, Z.: Memory overbooking and dynamic control of xen virtual machines in consolidated environments. In: 2009 IFIP/IEEE International Symposium on Integrated Network Management, IM 2009, pp. 630–637 (2009)
 69. Farokhi, S., Jamshidi, P., Lucanin, D., Brandic, I.: Performance-based vertical memory elasticity. *Proceedings—IEEE International Conference on Autonomic Computing, ICAC 2015*, pp. 151–152 (2015)
 70. Padala, P., Hou, K.-Y., Shin, K.G., Zhu, X., Uysal, M., Wang, Z., Singhal, S., Merchant, A.: Automated control of multiple virtualized resources. In: *EuroSys'09*, pp. 13–26 (2009)
 71. Kalyvianaki, E., Charalambous, T., Hand, S.: Self-adaptive and self-configured cpu resource provisioning for virtualized servers using kalman filters. In: *Proceedings of the 6th International Conference on Autonomic Computing*, pp. 117–126. ACM (2009)
 72. Zhu, Q., Agrawal, G.: Resource provisioning with budget constraints for adaptive applications in cloud environments. *IEEE Trans. Serv. Comput.* **5**(4), 497–511 (2012)
 73. Al-Shishtawy, A., Vlassov, V.: ElastMan: elasticity manager for elastic key-value stores in the cloud. In: *Cloud and Autonomic Computing Conference (CAC'13)*, p. 1 (2013)
 74. Wang, Z., Liu, X., Zhang, A., Stewart, C., Zhu, X., Kelly, T., Singhal, S., et al.: AutoParam: automated control of application-level performance in virtualized server environments. In: *Proceedings of the 2nd IEEE International Workshop on Feedback Control Implementation in Computing Systems and Networks (FeBid)*. Citeseer (2007)
 75. Wang, Z., Chen, Y., Gmach, D., Singhal, S., Watson, B.J., Rivera, W., Zhu, X., Hyser, C.D.: AppRAISE: application-level performance management in virtualized server environments. *IEEE Trans. Netw. Serv. Manage.* **6**, 4 (2009)
 76. Kjær, M.A., Kihl, M., Robertsson, A.: Resource allocation and disturbance rejection in web servers using slas and virtualized servers. *IEEE Trans. Netw. Serv. Manage.* **6**, 4 (2009)
 77. Saikrishna, P.S., Pasumarthy, R., Bhatt, N.P.: Identification and multivariable gain-scheduling control for cloud computing systems. *IEEE Transactions on Control Systems Technology* (2016)
 78. Qin, W., Wang, Q.: Modeling and control design for performance management of web servers via an LPV approach. *IEEE Trans. Control Syst. Technol.* **15**(2), 259–275 (2007)
 79. Tanelli, M., Ardagna, D., Lovera, M.: Identification of LPV state space models for autonomous web service systems. *IEEE Trans. Control Syst. Technol.* **19**(1), 93–103 (2011)
 80. Saikrishna, P.S., Pasumarthy, R.: Multi-objective switching controller for cloud computing systems. *Control Eng. Pract.* **57**, 72–83 (2016)
 81. Lama, P., Zhou, X.: Autonomic provisioning with self-adaptive neural fuzzy control for percentile-based delay guarantee. *ACM Trans. Auton. Adapt. Syst. (TAAS)* **8**(2), 9 (2013)
 82. Islam, S., Keung, J., Lee, K., Liu, A.: Empirical prediction models for adaptive resource provisioning in the cloud. *Future Gener. Comput. Syst.* **28**(1), 155–162 (2012)
 83. Qu, C., Calheiros, R.N., Buyya, R.: Auto-scaling web applications in clouds: a taxonomy and survey. *arXiv preprint arXiv:1609.09224* (2016)
 84. Gambi, A., Pezze, M., Toffetti, G.: Kriging-based self-adaptive cloud controllers. *IEEE Trans. Serv. Comput.* **9**(3), 368–381 (2016)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.



methods and bio-inspired methods.

Amjad Ullah received the M.Sc. degree in computer science from Quaid-i-Azam University, Islamabad Pakistan in 2005, M.Sc. in Advanced distributed system degree from University of Leicester, UK, in 2011 and Ph.D. degree in computer science from University of Stirling, Scotland, UK in 2017. His research interests include dynamic resource provisioning in cloud computing, fuzzy logic, feedback controllers, multi-objective optimization, search



Jingpeng Li is presently a Reader at the Division of Computer Science & Mathematics, University of Stirling, UK. He is also the head of CHORDS (Computational Heuristics, Operational Research and Decision Support) Research Group at Stirling. Dr Li received the M.Sc. degree in computational mathematics from Huazhong University of Science and Technology, China, in 1998, and the Ph.D. in operational research from University of Leeds, UK, in 2002. His research areas include Intelligent Transport Scheduling and Planning, Metaheuristics, Multi-Objective Decision Making, Optimization and Search Methodologies, Machine Learning, Data Mining, Fuzzy Logic, and software engineering. He has published over 50 technical papers in which more than 20 papers are in the top international journals (e.g. *Evolutionary Computation*, *IEEE Trans. on Evolutionary Computation*, *European Journal of Operational Research*, *Transportation Research Part B*, *Knowledge-Based systems*, etc).



Amir Hussain is Professor Science at the Edinburgh Napier University in Scotland. He obtained his B.E. in Electronic and Electrical Engineering (with the highest 1st Class Honours, with distinction) and Ph.D. (in novel neural network architectures and algorithms for real-world applications), both from the University of Strathclyde in Glasgow, UK, in 1992 and 1997 respectively. Following a Research Fellowship at the University of Paisley (now West

of Scotland), UK (1996–1998), and a Research Lectureship at the University of Dundee, UK (1998–2000), and professor at the University of Stirling in Scotland, UK (2000–2018), he joined the Edinburgh Napier university, where he is currently a professor. He is founding Editor-in-Chief of Springer's Cognitive Computation journal and the new BMC/Springer journal of Big Data Analytics. He is founding Series Editor for the Springer Book Series on Socio-Affective Computing and Spring Briefs on Cognitive Computation. He also serves on the Editorial Board of a number of other leading journals including, the IEEE Transactions on Neural Networks and Learning Systems and the IEEE Computational Intelligence

Magazine. His research interests are cross-disciplinary and industry focused, aimed at pioneering next-generation brain-inspired multi-modal Big Data cognitive technology for solving complex real world problems. He has (co)authored over 300 publications (including over a dozen Books, over 100 journal papers, and the worlds first research monographs on the multi-disciplinary areas of: cognitively inspired audio-visual speech filtering for multi-modal hearing-aids, sentic computing for natural language processing, and cognitive agent based computing). He has led more than 50 major multi-disciplinary research projects, as Principal Investigator, funded by national and European research councils, local and international charities and industry. He has supervised more than 30 Ph.D.s to-date, and serves as an International Advisor to various Governmental Higher Education and Research Councils, Universities and Companies. He regularly acts as invited Keynote Speaker, and has organized (as General/Organizing co-Chair) over 50 leading international Conferences to-date (including IEEE WCCI, IEEE SSCI, IJCNN, BICS and INNS Big Data Conference series). He is an invited member of several IEEE TCs, including the IEEE SMC TC on Cognitive Computing, and the IEEE CIS Emergent Technologies TC. He is Chapter Chair of the IEEE UK & RI Industry Applications Society Chapter, and founding co-Chair of the INNS Big Data Section. He is a Fellow of the UK Higher Education Academy (HEA), and Senior Fellow of the Brain Sciences Foundation (USA). More details on his research profile can be found on his homepage: <https://www.napier.ac.uk/people/amir-hussain>.