Genetic optimization of fuzzy membership functions for cloud resource provisioning

Amjad Ullah, Jingpeng Li, Amir Hussain
Division of Computing Science and Mathematics
University of Stirling
Stirling FK9 4LA, UK
Email: {aul,jli,ahu}@cs.stir.ac.uk

Yindong Shen
School of Automation
Huazhong University of Science and Technology
Wuhan 430074, China
Email: yindong@hust.edu.cn

***Abstract*—The successful usage of fuzzy systems can be seen in many application domains owing to their capabilities to model complex systems by exploiting knowledge of domain experts. Their accuracy and performance are, however, primarily dependent on the design of its membership functions and control rules. The commonly employed technique to design membership functions is to exploit the knowledge of domain experts. However, in certain application domains, the knowledge of domain experts are limited and therefore, cannot be relied upon. Alternatively, optimization techniques such as genetic algorithms are utilized to optimize the various design parameters of fuzzy systems. In this paper, we report a case study of optimizing the membership functions of a fuzzy system using genetic algorithm, which is an important part of our recently developed cloud elasticity framework. This work aims to improve the overall performance of the framework. Results obtained from this research work demonstrate performance improvement in comparison with our previous experimental settings.**

***Keywords*–Fuzzy logic; cloud resource provisioning; cloud elasticity; fuzzy membership functions parameters tuning; genetic algorithms; adaptive population size.**

I. Introduction

Cloud elasticity refers to the ability of a system to dynamically adjust the underlying computational resources in response to changes in demands in a way that at each time, the acquired resources closely match the demands, thus satisfying performance goals while reducing system running cost [1]. This definition indicates the following key purposes of elasticity: avoidance of system performance degradation, infrastructure capacity adjustments, minimizing running cost and energy consumption [2]. In order to make use of Cloud elasticity, an efficient elastic policy is needed to maintain system performance at a desired level whilst minimizing the running cost as well by adjusting the underlying computational resources.

In this regard, we have recently proposed a new methodology [3], [4] to handle cloud elasticity. The proposed method utilizes multiple feedback controllers simultaneously, where the selection of a suitable controller is realized through a Fuzzy Rule Based System (FRBS) at runtime. The results in [3], [4] demonstrate that our methodology has higher potential to improve system performance and reduce running cost in comparison to related state of the art elasticity approaches. The FRBS is the important part of the framework and therefore, this paper aims to find near optimal parameter settings for the design of the used membership functions with an objective to obtain more improved results, i.e. better system performance by minimizing the Service Level Objective (SLO) violations (explained in IV-B) and reduced system running cost.

An FRBS consists of two ingredients: (i) a Rule Base (RB) which is a collection of IF-Then rules, and (ii) a Data Base (DB) which contains the definitions and semantics for the linguistic terms of fuzzy sets. The commonly used approach of designing an FRBS is to acquire knowledge from domain experts and represent it using a DB and an RB [5]. However, in certain application areas, the knowledge of domain experts is limited and cannot be relied upon to implement an FRBS. Therefore, to overcome this issue, various optimization techniques have been used to determine the DB and RB of an FRBS. The use of such techniques are mostly for the following purposes: (i) to improve the accuracy by tuning the definition of existing membership functions, (ii) to enhance the interpretability by reducing the size of rule base, and/or (iii) to obtain an FRBS with a better trade-off between accuracy and interpretability [6], [7]. The scope of this paper is limited to tuning the already available membership functions to improve the accuracy of fuzzy system for the overall improvement of our previously proposed methodology and ultimately the system performance.

The commonly employed optimization techniques include evolutionary approaches [6]–[8], where the tuning of fuzzy membership functions is considered as a search problem. Evolutionary approaches are best known for their ability to identify near optimal parameter settings from a large search space even in the absence of a precise description of the underlying problem [6]. The evolutionary approaches based on the idea of natural biological evolution, where the survival of the fittest can be assured through natural processes such as randomly created population followed by reproduction and mutation [9]. The use of such techniques has successfully proven their suitability and has the potential to solve optimization problems from a wide range of domains. More specifically, there has been an extended use of genetic algorithms for the optimization of fuzzy systems, such as tuning membership functions for regression problems [7], [10], inference engine [11], [12], simultaneous learning of DB and RB [13] etc.

In this paper, we optimize the design of membership functions of a fuzzy system, which is an important part of our recently developed elasticity framework [3], [4]. For this purpose, we exploit a Multi-Objective Evolutionary Algorithm in combination with adaptive settings of population size and crossover probability. The rest of the paper is organized as follows. Section II provides a detailed overview of our existing framework. Section III explains the problem in hand and the parameters to be optimized, whereas Section IV describes the details of the employed genetic algorithm. Section V demonstrates the results obtained and Section VI concludes the paper.

II. A Multi-Controller Based Approach for Cloud Elasticity

In our earlier work [3], [4], we proposed a new methodology of cloud elasticity. This exploits the capability of multiple elastic feedback controllers at a time whilst considering the time-varying workload nature of web applications. The design of individual controllers are realized using the distribution of workload intensity into various categories namely low, medium and high. Whereas, the selection of a suitable controller amongst multiple controllers is realized at run time using an intelligent switching mechanism, which is implemented using a fuzzy system. The block diagram of the framework is provided in Figure 1 and the following sub sections explain the various aspects of the framework.

A. Control Policy

In the proposed control methodology, the three employed controllers are named as *Lazy*, *Moderate* and *Aggressive*. The well known integral control law [14] is used for each one. The performance metric used by our control methodology is the average CPU utilization, whereas the underlying computational resources (i.e. virtual machines) are used as control input. The methodology adjusts virtual machines to maintain the CPU utilization at a desired reference point. The following equation represent the applied integral control law in the context of our methodology:

$$u_{t+1} = u_t + K_i * (y_{ref} - y_t) \quad (1)$$

u_{t+1} represents new number of virtual machines at each iteration, while u_t denotes the number of virtual machines at that point of time. K_i is the integral gain parameter, which can be obtained off-line using any standard procedure [14]. y_{ref} and y_t represent the desired and measured CPU utilization.

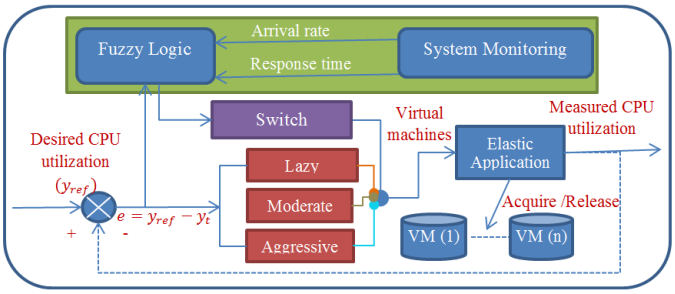


Fig. 1: Resource provisioning framework using multi-controller with fuzzy switching

B. System Monitoring

This component exploit the cloud provided Application Programming Interfaces (APIs) to obtain the up-to-dated status of system metrics (such as current CPU utilization). The latest information of the metrics represent the status of the system and is therefore required to make informed decision.

C. Switching Mechanism

The switching mechanism is responsible for the selection of a suitable controller at run time based on the current system behaviour. This mechanism is a fuzzy system that obtain the necessary information from *system monitoring* component and enables the output of one controller. Any fuzzy system can be constructed using the following three standard steps: (1) specifying domain knowledge into fuzzy sets, (2) defining membership functions, and (3) designing fuzzy rules. The brief description of each step in the context of switching mechanism is provided below.

- Domain knowledge: The design of the fuzzy system consists of three inputs (*Workload*, *ResponseTime*, *ControlError*) and one output (*Controller*). The *Workload* measure in percentage represents the incoming traffic in a particular time period, *ResponseTime* indicates performance of the system measured in terms of percentage Service Level Objective (SLO) violations (explained in IV-B) occurred in particular time period, and *ControlError* represents the percentage difference between desired and measured CPU utilization. Every input parameter is divided into three linguistic terms. Whereas, the output parameter is divided into four linguistic terms using the approach applied in [15], but no overlapping is considered for the output case.
- Membership functions: Each linguistic term of the fuzzy set in the fuzzy system is represented using a membership function. The membership functions for all the linguistic terms of each fuzzy set can be seen in Figure 2.
- Fuzzy rules: The fuzzy rules describe the relationship between system inputs and outputs. In our case, the fuzzy rules decide whether a scaling action is required or not. If the output comes as no scaling then there is no action required. Alternatively, a scaling action will be performed using the controller selected. The following is an example of our switching rules, where scale down operation is performed using the *Lazy* controller.

$$IF \overbrace{arrivalRate\ IS\ high}^{\text{Possible values: high, middle or low}} AND \overbrace{responseTime\ IS\ fast}^{\text{Possible values: fast, medium or low}}$$
$$AND \underbrace{error\ IS\ positive}_{\text{Possible values: Positive, Negative or Normal}} THEN \underbrace{controller\ IS\ lazy}_{\text{Possible values: Aggressive, Moderate or Lazy}}$$

There are 13 rules in total, which are available in [4]. Note that the focus of this paper is not to optimize these rules but the membership functions only. To sum up, the switching mechanism works as following at each iteration: the latest

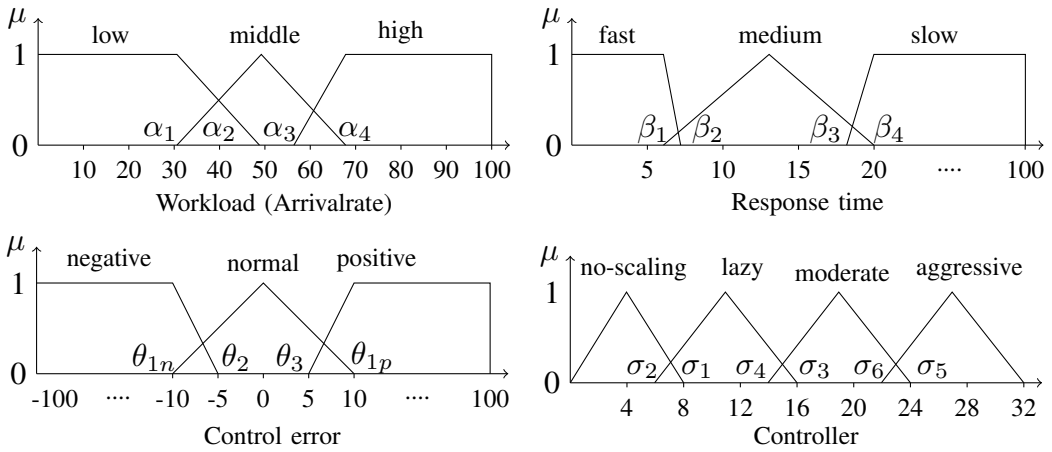


Fig. 2: Membership functions

information regarding inputs are obtained from the *System Monitoring* component and then fuzzified using the membership functions as shown in Figure 2. The inference engine then evaluates the rules to determine the output, i.e. *no-scaling* or *Controller*. If the output is *no-scaling* then no action is required, otherwise, the *Switch* component only enables the output of selected controller.

III. Definition of problem and parameters

The overall performance of our framework (described in Section II) is primarily dependent on the design of switching mechanism. In the current settings, the membership functions used (i.e. Figure 2) are partially derived from the related work [16] and others using trial and error approach. In this research, we have focused on the possibility to improve the overall performance of the framework by fine tuning the design of the existing membership functions using a multi-objective evolutionary algorithm. This section describes all the necessary details in this regard including problem definition and an overview of the parameters to be optimized and their corresponding design ranges.

The role of a membership function in a fuzzy system is to describe the information contained by fuzzy sets, which helps in converting to/from, crisp and fuzzy values. The membership function defines the degree of crisp value in the range 0 to 1 in accordance to a given linguistic variable. The membership functions can be of different types (e.g. Triangular, Trapezoidal, Gaussian, Sigmoidal, etc.) and their choice of selection is application dependent [17]. We have only used triangular and trapezoidal because they are comparatively simpler and efficient [18]. Irrespective of the type, every membership function contains three key ingredients referred as *Support, Boundary and Core/Prototype* [19]. They can be seen from the triangular and trapezoidal functions in Figure 3 as examples for a typical fuzzy sets, whereas their brief definitions summarized from [19] are given below:

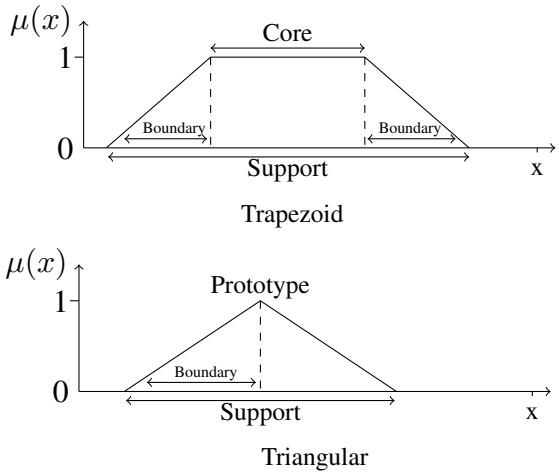


Fig. 3: Ingredients of Membership functions

- Support - The X refers the universe of a given fuzzy variable. Thus for a given fuzzy set A, *support* refers to that region of the universe X, where all elements $x \in X$ are characterized by $\mu_A(x) > 0$.
- Core/Prototype - For a given fuzzy set A, *core* refers to that region of the universe X, where all elements $x \in X$ are characterized by complete membership of set A, i.e $\mu_A(x) = 1$. The *prototype* is characterized by the same definition as *core* but with an exception that there is only one such element where $\mu_A(x) = 1$.
- boundary - Analogously, the *boundary* refers to that region of the universe X, where all elements $x \in X$ are characterized by $0 < \mu_A(x) < 1$.

In light of the above definitions and the underlying tuning problem, we are interested in finding out the near optimal values of the above ingredients for each membership function presented in Figure 2. The figure is marked with unique labels in the format of α_n, β_n, θ_n and σ_n to represent a different point over the universe. Each point is an independent parameter, and its collection in a specific order defines the ingredients of the membership function, e.g. the length from α_1 to α_4

identifing the *support* region for *middle* membership function of the *Workload* variable. Similarly, the *core* and *boundaries* will be calculated from these parameters automatically. For example, the length from α_1 to α_2 defines *boundary* for *low* and *middle* membership function, whereas the length from 0 to the value of α_1 identifies the *core* for *low* membership function. In total, 18 parameters can be seen in Figure 2. However, we consider parameters (θ_{1n} and $\theta 1p$) collectively as one parameter, i.e. with the same value but different +/- sign. Each parameter is assigned with a range of values, which are given in Table I.

Fuzzy Variable	Linguistic Terms					
	Start range — End range					
Controller	σ_1	σ_2	σ_3	σ_4	σ_5	σ_6
	4 – 12	4 – 12	12 – 20	12 – 20	20 – 28	20 – 28
Workload	α_1	α_2	α_3	α_4		
	15 – 35	30 – 50	50 – 70	60 – 80		
Response Time	β_1	β_2	β_3	β_4		
	2 – 10	6 – 10	10 – 16	16 – 20		
Control error	θ_{1n}	θ_2	θ_3	θ_{1p}		
	-35 – -10	-20 – 0	0 – 20	10 – 35		

TABLE I: The design range of each parameter

Apart from the above mentioned parameters, we also consider the possibility of identifying the suitable defuzzification operator from the list of commonly used operators, thus increasing the total number of parameters to 18. The defuzzification parameter is nominal and we have considered 4 different standard defuzzification procedures.

The genetic algorithm in hand (explained in Section IV) operates to explore the possibility to find out the near optimal combinations of the above parameters that result in improved performance of the proposed framework. However, during this process, each combination of the parameters shall satisfy the following constraints:

i $\alpha_1 \leq \alpha_2$, $\alpha_3 \leq \alpha_4$ and
ii $\beta_1 \leq \beta_2$, $\beta_3 \leq \beta_4$ and
iii $\theta_{1n} \leq \theta_2$, $\theta_3 \leq \theta_{1p}$ and
iv $\sigma_1 \geq \sigma_2$, $\sigma_3 \geq \sigma_4$ and $\sigma_5 \geq \sigma_6$.

The above constraints ensure that every developing fuzzy partition should be either adjacent or overlapped with neighbour partition. Thus confirming that every point x over the universe must exist in at-least one of the *support* region. In light of these constraints, if any generated solution does not satisfy the above constraints during the evolution process, the solution will be ignored without evaluation and will be replaced with another valid solution that satisfies the above constraints.

IV. The Design of Fuzzy Membership Functions Using Genetic Algorithm

The Genetic Algorithms (GAs) [9] are search mechanisms based on the idea of genetics and natural selection. They are iterative procedures, where they work on the population of individuals (also referred to as chromosomes or solutions). The algorithm usually starts with a randomly generated population of solutions and aims to evolve towards better solutions by applying different genetic operations including natural selection, recombination and mutation. Such algorithms are considered very useful for problems that demands efficient search in the largely available problem space [20]. Moreover, GAs have the ability to identify near optimal parameter settings from a large search space even in the absence of a precise description of the underlying problem [6]. The following sub sections explains the various details in the context of underlying problem.

A. Chromosome Encoding

The first stage of employing a GA for a problem is to represent the chromosome in a way that is suitable to be modified by the genetic operations. We have used the binary encoding and represented the chromosome in a fixed size (64) bits long binary string. Each parameter can be represented using a fixed (L) number of bits, which are then concatenated to form the binary string as following:

$$\underbrace{\overbrace{b_1..b_L}^{\alpha_1}....\overbrace{b_1..b_L}^{\alpha_4}}_{Workload(16)}\underbrace{\overbrace{b_1..b_L}^{\beta_1}....\overbrace{b_1..b_L}^{\beta_4}}_{ResponseTime(12)}\underbrace{\overbrace{b_1..b_L}^{\theta_1}....\overbrace{b_1..b_L}^{\theta_3}}_{Error(16)}\underbrace{\overbrace{b_1..b_L}^{\sigma_1}....\overbrace{b_1..b_L}^{\sigma_6}}_{Controller(18)}\overbrace{b_1b_2}^{\text{DO}}$$

The *Workload*, *Response Time*, *Error* and *Controller* are the fuzzy variables, where *DO* is the defuzzification operator. The length of each parameter in the binary string is relative to the fuzzy variable. Thus, while during the decoding process, the binary string is decomposed into five parts and then each parameter except *DO* is decoded to get its corresponding value using the following equation adapted from [19],

$$C_i = C_{min_i} + \frac{b}{2^L - 1}(C_{max_i} - C_{min_i}), \qquad (2)$$

where C_i represents parameter i, C_{min_i} and C_{max_i} represent the minimum and maximum range for parameter i, b is the corresponding decimal value of the binary bits, and L represents the number of bits length for parameter i.

B. The Multi-Objective Fitness Criteria

The fitness function quantifies the quality of an individual of a given population. The efficiency of a cloud resource provisioning strategy can be measured in terms of performance achievement. However, it is also necessary to take into account the corresponding running cost of that strategy as we can always achieve better performance by acquiring more resources than required. Therefore, a good strategy is to utilize the resources as efficiently as possible, aiming to improve system performance whilst reducing running cost.

Considering the multi-criterion nature of the underlying problem, we employed a multi-objective genetic algorithm with two objectives to obtain a set of Pareto optimal solutions rather than one best solution. A brief description of the objectives considered is given below:

i SLO violation: SLO stands for Service Level Objectives. The SLO is measurable element of Service Level Agreement (SLA), which defines the entire agreement between

a service provider and consumer specifying the details of the contract between two parties. SLO is one particular measurable unit of the SLA such as quality of service, response time, throughput, etc. For the problem in hand, we measure response time as a criterion to measure the performance of the underlying system with a given elastic policy. We consider that each service request must be completed in a predefined desired time unit. An SLO violation will be considered, if the execution time of a service request takes more than the predefined desired time unit.

ii Running cost: It refers to the cost of the computational resources (Virtual machines), which are acquired to provide execution services to the job requests. Each acquired resource is associated with a cost per time unit. Thus, during the operation of an experiment, the execution time of all acquired virtual machines are recorded and the cost are calculated for the entire experiment.

We are interested in maximizing the system performance and in the meantime minimizing the cost. Therefore, the lower the values of SLO violation and cost, the better the quality of a solution.

C. Employing Multi-Objective GA with Adaptive Attributes

The standard nondominated sorting algorithm-II (NSGA-II) [21] is utilized to tackle the underlying problem. The NSGA-II is a generic and commonly used standard algorithm to solve multi-objective problems. The NSGA-II starts as other evolutionary approaches with a population of competing solutions. It then sorts and ranks every solution of the population with respect to non-domination level followed by applying genetic operations (selection, crossover and mutation) to create offspring populations. The NSGA-II then takes the union of parent and offspring populations followed by partitioning of union in fronts. After this step, the NSGA-II applies a mechanism called crowding distance to enhance the spread and diversity of individuals followed by the step of elitism, where the population for the next generation is selected from the non-dominated individuals thus improving the convergence [22]. In addition to NSGA-II, the following methods are employed:

i Adaptive population size: The benefits of employing adaptive population size approach are evident in optimization literature (e.g. increase in speed [23], escape in stagnation and diversity [24], etc.). Considering such benefits of adaptive population size, we adopt the approach.
The lines from 6 to 12 in Algorithm 1 represent the method of [23]. This method suggests that the population size will be increased in two cases: (i) When fitness of offspring population improves. This will make the algorithm biased towards exploration of the population. (ii) When fitness of the offspring population does not improve for a longer period. This will help the algorithm to get out of stagnation. If both of the above conditions are not fulfilled then the population size will be reduced. The growth($\triangle_1$ and $\triangle_2$) and shrink rate ($\bigtriangledown$) can be adjusted as per the needs. We use constant proportionate rates of 10% for $\triangle_1$, 15% for $\triangle_2$) and 5% for $\bigtriangledown$. With reference to the bi-objectives nature of the problem, we consider the condition of *best fitness improved*, if the offspring individual is improved in either objective.

Algorithm 1 The approach with adaptive population size [23]

```
1:  Generate and initialize population
2:  Evaluate every individual of the population
3:  while not stopping criteria do
4:      Selection, Recombine and Mutate
5:      Evaluate every individual of the offspring
6:      if best fitness improved then
7:          Grow population size by rate △1
8:      else if no improvement for long period then
9:          Grow population size by rate △2
10:     else
11:         Shrink population size by rate ▽
12:     end if
13:     Evaluate new individuals
14: end while
```

ii Adaptive crossover probability: We also employ the approach of [27], which computes the probabilities of genetic operations (crossover and mutation) on runtime in accordance to the fitness values of the current population. This technique helps to maintain population diversity as well as strengthen converge capacity. According to the method, the probabilities are increased when the GA ceases in local optimum, whereas they are decreased, when the individuals are well dispersed in the solution space. The following equation from [27] derives the value of crossover probability:

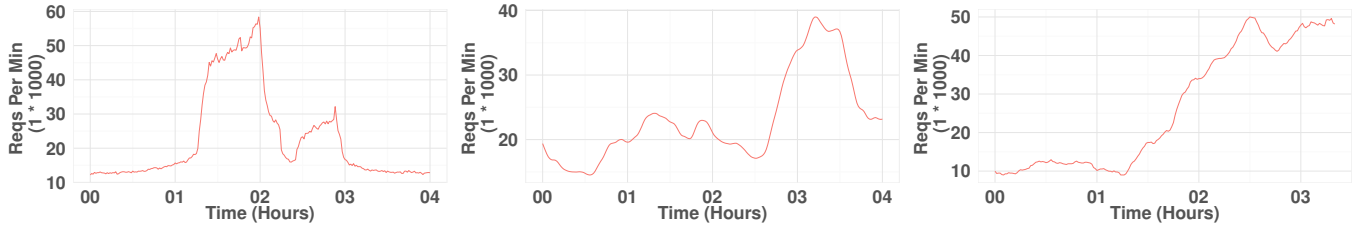
$$p_c = \begin{cases} k_1(f_{max} - f^{'})/(f_{max} - \bar{f}), & f^{'} \geq \bar{f} \\ k_2, & f^{'} < \bar{f} \end{cases} \quad (3)$$

f_{max} represents the best fitness of the population, $\bar{f}$ the average fitness, and $f^{'}$ the best fitness among the parents, which have to be recombined during crossover operation. The value of k_1 and k_2 can be between 0 and 1. We use the value 1 for both k_1 and k_2 as per the recommendations of [27]. These values ensure that all those solution, for which the fitness is below or equal to the average fitness must go through crossover operation. The probability of crossover decreases for solutions when their fitness values are better than the average fitness, and tends to zero as the fitness value reaching to the value of best fitness solution.

V. Computational Results

A. Experimental Set-up

We have previously set-up an experimental environment that includes a prototypical implementation of our elasticity framework. This includes the integration of a well known Java based cloud simulator termed as CloudSim [28] and a Java library named JFuzzylogic [29] to deal with FRBS. In this paper, we have extended the experimental environment



(a) Worldcup [25] (b) Nasa [26] (c) Dualphase

Fig. 4: Various workloads used for experimentation

by incorporated a well known Java based optimization library called jMetal [30] to utilize NSGA-II.

For the experimentation, we consider that elasticity framework manages a cluster of virtual machines on behalf of a web application. Each incoming web (http) request is therefore considered as a job for the framework. Each job is associated with a pre-defined service time, which arrives at a specific time. The arrival time represents the actual arrival of that http request obtained from real traffic workloads use for the experimentations, whereas service time of each job is randomly assigned between (10 to 500 millisecond). An SLO violation will be considered, if the execution of a job takes more than 1 second. The gain parameters used are the same as in [3] (i.e. *Lazy=-0.06, Moderate=-0.7* and *Aggressive=-1.1*) to conduct the experimentation.

One experiment consists of full evaluation of a case scenario, i.e. the execution of GA for a given test scenario with different workload. In each iteration, every individual is the design parameter settings of all membership functions, which will be evaluated from the elasticity framework in terms of number of SLO violations and cost.

Workload	Cost	SLO
Nasa	101.64	36436
Worldcup	105.97	402180
DualAuck	100.82	102321

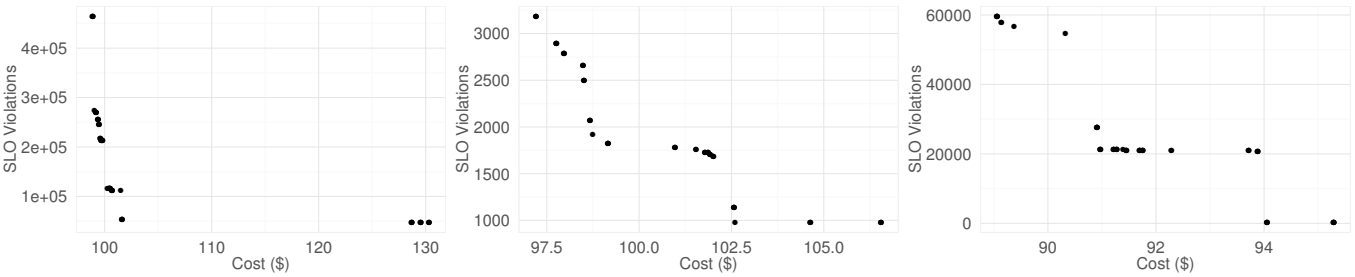
TABLE II: Results obtained using existing membership functions (Figure 2)

B. Results and Analysis

The various real workload traces utilized for the conducted experiments are provided in Figure 4. Each workload trace represents a different scenario. The result are recorded for each individual in terms of the multi-objective criteria, i.e. the number of SLO violation and the cost during all experiments. The Pareto fronts obtained from each workload scenario can be seen in Figure 5. Considering the trade-off between the evaluation objectives, it is difficult to decide the best solution in each scenario. However, in order to analyse the efficiency of the applied optimization method, we compare some aspects of the Pareto fronts with the results obtained using our existing membership functions. The key reasons behind this comparison are twofold. Firstly, the unavailability of knowledge regarding true optimum. Secondly, to answer whether applying optimization method can enhance the design of membership functions for the overall performance improvement of our existing elasticity framework.

The results in Table II are obtained using our existing settings, where Table III demonstrates results of some selected solutions from Pareto fronts. Following the approach from [20], [31], we calculate Relative Percentage Deviations (RPD) using the following equations in order to assess the quality of the selected Pareto fronts solution in comparison with the quality of results in Table II.

$$RPD_{cost} = \left(\frac{GA_{cost} - Existing_{cost}}{Existing_{cost}}\right) * 100 \quad (4)$$



(a) Worldcup (b) Nasa (c) Dualphase

Fig. 5: The Pareto front of each workload scenario

Workloads	Cost wise best				SLO wise best				Combined best				
	Cost	RPD_{cost}	SLO	RPD_{slo}	SLO	RPD_{SLO}	Cost	$RPDcost$	Cost	RPD_{cost}	SLO	RPD_{slo}	$RPD_{cost+slo}$
Nasa	97.21	-4.35	3184	-91.26	980	-97.31	106.55	4.83	98.75	-2.85	1916	-94.74	-97.59
Worldcup	98.90	-6.67	464228	15.43	47878	-88.10	130.31	22.97	101.63	-4.09	53391	-86.72	-90.82
DualPhase	89.06	-11.66	59599	-41.75	327	-99.68	95.29	-5.49	94.06	-6.71	344	-99.66	-106.37

TABLE III: Some selected (best) solutions from Pareto fronts

$$RPD_{slo} = (\frac{GA_{slo} - Existing_{slo}}{Existing_{slo}}) * 100 \quad (5)$$

The RPD value of each objective specifies the percentile variation in comparison with the existing result. A positive RPD value shows the percentage degradation of the quality in comparison with the existing results, where a negative RPD value represents the percentage improvement in the quality. This implies that the lower the value of RPD, the better the quality of results.

Table III presents the results in three aspects: (i) *Cost wise best* highlighting the result, which is the best in terms of *Cost* objective, (ii) *SLO wise best* showing the best results in terms of *SLO* objective, and (iii) *combined best* presenting the best results with respect to lowest RPD value after adding the RPD_{cost} and RPD_{slo} together for each solution assuming both objectives are equally important. It is evident from Table III that results are improved in all aspects. More specifically,

i Considering the importance of any individual objectives (i.e. either *Cost* or *SLO*), we can see that the results are improved in every scenario (i.e. using various workloads).

ii Considering the scenario, where both objectives are equally important, it is evident that the best solution in every test scenarios is improved significantly.

This indicates that the existing membership functions have the possibility to be improved, which can increase the overall performance of our framework and by using multi-objective GA, we can obtain better parameter settings for the underlying membership functions.

VI. Conclusion

In this paper, a multi-objective genetic algorithm is exploited to address the issue of designing fuzzy membership functions as an optimization problem. The problem is tackled for a particular fuzzy system, which is an extension of cloud elasticity framework developed recently. The multi-objective genetic algorithm searches for better design parameters of target membership functions that result in improved system performance in comparison with the existing settings. The results are evaluated using two objectives of conflicting nature from cloud elasticity context, i.e. the number of SLO violations and cost. The experimental results indicate that the system performance can be improved significantly using the parameter settings obtained from evolutionary algorithm in comparison to our existing parameter settings.

Acknowledgment

The research described in this paper was funded by the UK Engineering and Physical Sciences Research Council (EPSRC), under grant EP/J017515/1. The work is also supported by Natural Science Foundation of China (under grants 71571076 and 71171087). A. Hussain was supported by the EPSRC grant no. EP/M026981/1.

References

[1] Nikolas Roman Herbst, Samuel Kounev, and Ralf Reussner. Elasticity in cloud computing: what it is , and what it is not. In *10th International Conference on Autonomic Computing*, pages 23–27, 2013.

[2] Guilherme Galante and Luis Carlos E De Bona. A survey on cloud computing elasticity. In *Proceedings - 2012 IEEE/ACM 5th International Conference on Utility and Cloud Computing, UCC 2012*, pages 263–270, 2012.

[3] Amjad Ullah, Jingpeng Li, Amir Hussain, and Erfu Yang. Towards a Biologically Inspired Soft Switching Approach for Cloud Resource Provisioning. *Cognitive Computation*, pages 1–14, 2016.

[4] Amjad Ullah, Jingpeng Li, and Amir Hussain. Towards workload-aware cloud resource provisioning using a multi-controller fuzzy switching approach. *International Journal of High Performance Computing and Networking*, 2016.

[5] Pietro Ducange and Francesco Marcelloni. Multi-objective Evolutionary Fuzzy Systems. In *WILF*, pages 83–90. Springer, 2011.

[6] Michela Fazzolari, Rafael Alcala, Yusuke Nojima, Hisao Ishibuchi, and Francisco Herrera. A review of the application of multiobjective evolutionary fuzzy systems: Current status and further directions. *Fuzzy Systems, IEEE Transactions on*, 21(1):45–65, 2013.

[7] Rafael Alcalá, María José Gacto, Francisco Herrera, and Jesús Alcalá-Fdez. A multi-objective genetic algorithm for tuning and rule selection to obtain accurate and compact linguistic fuzzy rule-based systems. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 15(05):539–557, 2007.

[8] Francisco Herrera. Genetic fuzzy systems: Taxonomy, current research trends and prospects. *Evolutionary Intelligence*, 1(1):27–46, 2008.

[9] David E Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*, volume Addison-We. 1989.

[10] María José Gacto, Rafael Alcalá, and Francisco Herrera. Integration of an index to preserve the semantic interpretability in the multiobjective evolutionary rule selection and tuning of linguistic fuzzy systems. *IEEE Transactions on Fuzzy Systems*, 18(3):515–531, 2010.

[11] Antonio A Márquez, Francisco Alfredo Márquez, and Antonio Peregr'in. Rule Base and Inference System Cooperative Learning of Mamdani Fuzzy Systems with Multiobjective Genetic Algorithms. In *IFSA/EUSFLAT Conf.*, pages 1045–1050, 2009.

[12] Antonio A Márquez, Francisco A Márquez, and Antonio Peregrín. A multi-objective evolutionary algorithm with an interpretability improvement mechanism for linguistic fuzzy systems with adaptive defuzzification. In *Fuzzy Systems (FUZZ), 2010 IEEE International Conference on*, pages 1–7. IEEE, 2010.

[13] Rafael Alcalá, Pietro Ducange, Francisco Herrera, Beatrice Lazzerini, and Francesco Marcelloni. A multiobjective evolutionary approach to concurrently learn rule and data bases of linguistic fuzzy-rule-based systems. *IEEE Transactions on Fuzzy Systems*, 17(5):1106–1122, 2009.

[14] Joseph L Hellerstein, Yixin Diao, Sujay Parekh, and Dawn M Tilbury. *Feedback control of computing systems*. John Wiley & Sons, 2004.

[15] Rudwan Abdullah, Amir Hussain, Kevin Warwick, and Ali Zayed. Autonomous intelligent cruise control using a novel multiple-controller framework incorporating fuzzy-logic-based switching and tuning. *Neurocomputing*, 71(13):2727–2741, 2008.

[16] Pooyan Jamshidi, Aakash Ahmad, and Claus Pahl. Autonomic resource provisioning for cloud-based software. In *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 95–104. ACM, 2014.

[17] Chuen-Chien Lee. Fuzzy logic in control systems: fuzzy logic controller. II. *IEEE Transactions on systems, man, and cybernetics*, 20(2):419–435, 1990.

[18] Kevin M Passino, Stephen Yurkovich, and Michael Reinfrank. *Fuzzy control*, volume 42. Addison-wesley, Menlo Park, CA, 1998.

[19] Timothy J. (University of New Mexico) Ross. *Fuzzy logic with engineering applications*. 2010.

[20] Jingpeng Li and Kwan Raymond. A fuzzy genetic algorithm for driver scheduling. *European Journal Of Operational Research*, 147(2):334–344, 2003.

[21] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[22] Carlos a Coello Coello, Gary B Lamont, and David a Van Veldhuizen. *Evolutionary Algorithms for Solving Multi-Objective Problems Second Edition*. 2007.

[23] Agoston Endre Eiben, Elena Marchiori, and V A Valkó. Evolutionary Algorithms with On-the-Fly Population Size Adjustment. *Parallel Problem Solving from Nature, (PPSN VIII)*, 3242(8):41–50, 2004.

[24] Michael Affenzeller, Stefan Wagner, and Stephan Winkler. Self-adaptive population size adjustment for genetic algorithms. *Computer Aided Systems Theory ...*, pages 820–828, 2007.

[25] Internet Traffic Archive. Worldcup 1998 Web trace, 2015.

[26] Network Traffic Archive. Nasa-HTTP, 2015.

[27] M. Srinivas and L. M. Patnaik. Adaptive Probabilities of Crossover and Mutation in Genetic Algorithms. *IEEE Transactions on Systems, Man and Cybernetics*, 24(4):656–667, 1994.

[28] Rodrigo N Calheiros, Rajiv Ranjan, Anton Beloglazov, César A F De Rose, and Rajkumar Buyya. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience*, 41(1):23–50, 2011.

[29] Pablo Cingolani and Jesus Alcala-Fdez. jFuzzyLogic: a robust and flexible fuzzy-Logic inference system language implementation. In *FUZZ-IEEE*, pages 1–8. Citeseer, 2012.

[30] Juan J. Durillo and Antonio J. Nebro. JMetal: A Java framework for multi-objective optimization. *Advances in Engineering Software*, 42(10):760–771, 2011.

[31] Jingpeng Li. *Fuzzy Evolutionary Approaches for Bus and Rail Driver Scheduling*. PhD thesis, University of Leeds, 2002.