Fuzzy-Q&E: achieving QoS guarantees and energy savings for cloud applications with fuzzy control

Luca Albano*, Cosimo Anglano†, Massimo Canonico† and Marco Guazzone‡
*Postal and Communications Police, Ministry of Interior, Italy
email: luca.albano@gmail.com
†DiSIT - Computer Science Institute, Università del Piemonte Orientale, Italy
email: {cosimo.anglano,massimo.canonico}@di.unipmn.it
‡Dipartimento di Informatica, Università di Torino, Italy, email: marco.guazzone@di.unipmn.it

***Abstract*—We address the problem of managing cloud applications, consisting of a set of virtual machines (VMs), characterized by bursty and dynamic workloads, in such a way to provide guarantees on their Quality-of-Services (QoS) and, at the same time, to minimize the energy consumption of the physical infrastructure running them.**

We propose a fuzzy controller, *Fuzzy-Q&E*, that is able to allocate to the VMs of each cloud application the minimum amount of physical capacity needed to meet its QoS requirements. In this way, the number of physical resources that must be switched-on at any given time is reduced with respect to the case in which physical machines are statically provisioned and, consequently, less energy is required to run a given cloud workload.

We implement a prototype of our controller on a Xen-based testbed, and we perform a set of experiments using an E-Commerce benchmark in which we compare Fuzzy-Q&E against *DynaQoS*, a state-of-the-art fuzzy controller for virtualized resources. Experimental results show that Fuzzy-Q&E outperforms DynaQoS both in terms of the ability of meeting the QoS level of the application, and of the amount of physical capacity allocated to each VM.

I. Introduction

Many modern Internet services are implemented as cloud applications, that consist of a set of *Virtual Machines* (VMs) allocated and run on a physical infrastructure, typically owned by an *Infrastructure Provider* (IP), managed by a virtualization platform (e.g., Xen [1]). These applications typically require that specific levels of user-perceived *Quality-of-Service* (QoS), usually expressed as a high-level *Service Level Objective* (SLO), are guaranteed; failing to do so results in a money penalty for the IP.

A possible approach to fulfill the SLOs of an application, especially in presence of time-varying workloads (a characteristic common to many cloud applications [2]), consists in estimating and allocating to the application the amount of physical capacity it requires to meet its SLO under peak workload demand. In order to avoid that competing VMs, sharing the same physical resource(s), "steal" to each other the capacity allocated them, suitable non work-conserving scheduling mechanisms are used by the hypervisor: in general, each VM is associated with a *CAP* (a real number, usually taking values in the range between 0 and 100), that fixes the maximum amount of CPU that the VM will be able to consume, even if the host system has idle CPU cycles.

Provisioning for peak demand, however, implies that for off-peak workloads the CAP is over-allocated to the VMs of the applications. Consequently, less VMs can be placed on the same physical machine, and more physical machines must be switched on to run the workload that, in turn, induces a higher power consumption. To maximize profit, the IP must instead dynamically adjust the amount of physical capacity allocated to each VM to match the current workload demand, so that it can balance the need of reducing the number of switched-on physical machines with the one of ensuring that the VM receives enough physical resource capacity to meet its SLOs (so that money penalties are avoided).

In this paper, we propose *Fuzzy-Q&E* (where "Q" and "E" stand for "QoS" and "Energy", respectively), a *fuzzy feedback control system* [3], [4] that is able to meet the SLOs of cloud applications by dynamically allocating to their VMs just the amount of physical capacity that is needed to reach the SLOs for current workload conditions. In this way, Fuzzy-Q&E reduces the average CAP allocated to the various VMs of an application, thus enabling a greater consolidation level on the machines of the physical infrastructure and, hence, to reduce the number of switched-on machines.

We implement a prototype of Fuzzy-Q&E on a Xen-based testbed, and we perform a set of experiments using an E-Commerce benchmark [5], subject to a bursty and time-varying workload, in which we compare our controller against *DynaQoS* [6], a state-of-the-art fuzzy controller for virtualized resources that has been shown to outperform classical model-based controllers. Experimental results show that Fuzzy-Q&E outperforms DynaQoS both in terms of the ability of achieving application SLOs, and of the amount of physical capacity allocated to each VM. This, in turn, enables the achievement of a higher consolidation level (and hence greater energy savings) on a given physical infrastructure than existing fuzzy controllers.

The rest of the paper is organized as follows. In Section II, we define the context for the problem that we tackle in this paper. In Section III-B, we illustrate the design of the Fuzzy-Q&E control system. In Section IV, we present the results obtained from an experimental evaluation. Related works are discussed in Section V. Finally, in Section VI, we conclude the paper and discuss possible future works.

II. Background and motivation

We consider a physical computing infrastructure, managed by a virtualization platform that provides a CAP-based mechanism to specify and enforce limitations on the maximum amount of CPU capacity allocated to each VM running on it.

We assume that this infrastructure hosts a set of multi-tier cloud applications (each one consisting of a set of VMs), that provide services to a population of clients issuing streams of requests. We also assume that the number of VMs composing each application is statically fixed (i.e., it is not varied at run time to track changes in the intensity of the workload), and that the capacity available on each physical machine is sufficient to accommodate the aggregate demand of all the VMs it hosts. In case the aggregate demand exceeds the available capacity, either some VMs have to be migrated, or further VMs must be spawned, on other physical machines. We assume, however, that the initial placement of the VMs on the physical machines, as well as their migration or the spawning of additional VMs, is performed by the IP by using a suitable algorithm (e.g., [7]–[10]).

Each application is associated with a SLO, expressed as a bound r of the 95^{th} percentile of the response time of the requests it completes (using percentile-based SLOs is considered preferable for many applications than simpler metrics like average [11]). This means that 95% of the response time of the various requests must be smaller than, or equal to, the value r, during a prescribed time interval.

As typical in real production environments, we assume that the value of r, henceforth referred to as the *SLO value*, is provided by the customer (i.e., the owner of the application), as it usually depends on the expected maximum workload and by other factors like the quality of experience that the IP cannot know in advance. Once the IP knows the SLO value and the details about the maximum workload intensity $W_{\max}$ expected for the application, (s)he allocates to the various VMs the CAP needed to guarantee that the 95^{th} percentile of the response time does not exceed r. Of course, it is clear that, when the workload intensity is lower than $W_{\max}$, the IP is free to reduce the above CAP value, as less physical resources are required to meet the SLO of the application.

Fuzzy control has been already used to tackle the problem described above. However, to the best of our knowledge, existing fuzzy controllers, unlike ours, base their decisions on the values of the error $e = (RT - SLO)$ between the measured (RT) and the SLO (SLO) values of response time and on the values of its first-order difference (Δe), and react to these values by increasing (when $e > 0$) or decreasing (when $e < 0$) the amount of CPU capacity allocated to the application. Unfortunately, this approach suffers from the following drawback.

Indeed, for practical reasons, controllers are typically designed to work with average values $\overline{RT}$ of response times, that are computed considering a suitable number of observed values. Unfortunately, the usage of $\overline{RT}$ as the only control parameter makes very hard for the controller to track abrupt changes in the workload intensity of the controlled application. As a matter of fact, consider a scenario in which the workload suddenly increases. In this case, a queue of requests waiting to be served builds up, and the lower the position of a request in the queue, the higher its waiting time and, consequently, its response time. However, $\overline{RT}$ (being an average value) grows slowly, especially if the system has operated under a lower workload intensity for a long time. Consequently, although the right control decision would be to abruptly increase the CPU capacity allocated to the application, the controller would increase it smoothly. Consider now a scenario in which the workload suddenly decreases, so that the queue of waiting requests drains. In this case, the value of $\overline{RT}$ decreases slowly even after the queue empties (as effect of the average), so the controller slowly reduces the capacity allocated to the application, although the correct control decision would be to decrease it abruptly.

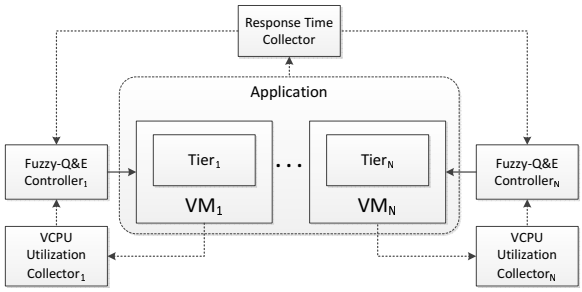


Fig. 1: Fuzzy-Q&E – The system architecture.

In order to deal with workloads exhibiting strong intensity variations, we use percentile-based response time metrics and we introduce an additional control parameter, namely the difference C_{res} between the CPU capacity allocated to a VM, and the capacity that is actually used by that VM. Under stationary workload conditions, we expect that $C_{\text{res}} \cong 0$, i.e. the system has received approximately the capacity it needs to meet its SLO. However, when the workload intensity abruptly increases, we expect to observe that $C_{\text{res}} = 0$ (i.e., the system is saturated), while when it decreases we expect to observe that $C_{\text{res}} \gg 0$.

We base the design of Fuzzy-Q&E on the idea that, by combining the information coming from the response time and C_{res}, better control decisions are possible, and our experimental results, discussed in Section IV, confirm our intuition.

III. The Fuzzy-Q&E system

In this section, we present Fuzzy-Q&E, a fuzzy control system that, given a set of VMs running the tiers of an application, is able to respect the application SLO and, at the same time, to reduce as much as possible the CAP of physical CPUs assigned to each of such VMs. In the rest of this section, we first present the overall architecture of our system (Section III-A), and then we illustrate in detail the design of our fuzzy controller component (Section III-B).

A. System Architecture

The overall architecture of the Fuzzy-Q&E system is depicted in Fig. 1. As can be observed in the figure, a set of one or more *Fuzzy-Q&E Controller* components act on a group of VMs (each of which runs a tier of a multi-tier application)

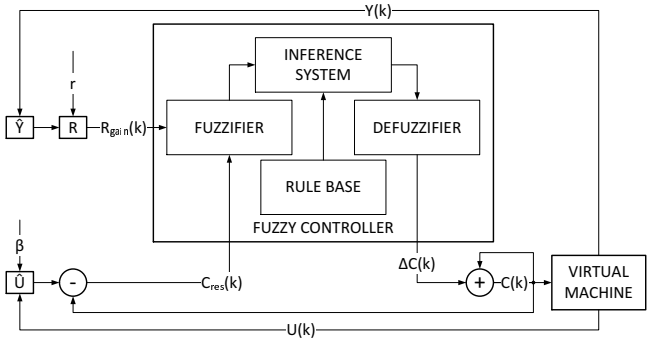


Fig. 2: Fuzzy-Q&E – The fuzzy controller.

to dynamically assign fractions of CPU capacity to them, in order to meet application SLOs. There is one Fuzzy-Q&E controller component for each VM. Note that the figure does not intentionally include details about the physical machines hosting the VMs of a given application, to stress that these VMs may run either on the same or on different machines.

The system also includes other auxiliary components, mainly used for collecting performance measures, namely the *VCPU Utilization Collector* and the *Response Time Collector* components. The *VCPU Utilization Collector* periodically measures the virtual CPU (VCPU) utilization of a VM, while the *Response Time Collector* periodically measures the response time of an application. Both values are "sent" to the related Fuzzy-Q&E controller, which, in turn, computes the new CPU CAP value for the controlled VM to meet the SLO of the associated application. The Fuzzy-Q&E controller enforces this new CAP to adjust (and to limit, if possible) the CPU usage of each VM.

In the rest of this section, we describe in detail the design of the Fuzzy-Q&E controller component.

B. The Fuzzy-Q&E Controller Component

The *Fuzzy-Q&E Controller* component is a feedback controller based on the fuzzy logic [3], [4]. The design of the controller is presented in Fig. 2. As can be observed in the figure, there are two control inputs and one control output. The control inputs are the *response time gain* R_{gain} and the *residual capacity* C_{res}. R_{gain} is the normalized difference between the reference value r (i.e., the SLO value, in our case) and the incremental estimate $\widehat{Y}$ of the 95^{th} percentile of the observed response times Y, obtained by means of the P^2 *algorithm* [12]. Specifically, for each sampling period k, R_{gain} is computed as follows:

$$R_{\text{gain}}(k) = \frac{r - \widehat{Y}(k)}{r} \tag{1}$$

C_{res} is the difference between the actual available CPU CAP value C and a smoothed value $\widehat{U}$ of the observed VCPU utilizations. $\widehat{U}$ is obtained by means of the *exponentially weighted moving average* (EWMA) method [13]. Specifically, for each sampling period k, C_{res} is computed as follows:

$$C_{\text{res}}(k) = C(k-1) - \widehat{U}(k) \tag{2}$$

The EWMA value $\widehat{U}$ of the observed VCPU utilizations is computed as follows:

$$\widehat{U}(k) = \beta \cdot u(k) + (1-\beta) \cdot \widehat{U}(k-1) \tag{3}$$

where $u(k)$ is the current observed VCPU utilization and $0 \leq \beta \leq 1$ is the smoothing factor, which represents the weight assigned to the most recently observed VCPU utilization $u(k)$.

For what regards the control output, it is represented by the *CPU CAP offset* ΔC, that is the adjustment to the CPU CAP to apply to the VM in order to satisfy the reference response time. Thus, the final CPU CAP value $C(k+1)$ to allocate to a VM during the $(k+1)^{\text{th}}$ control period is computed as:

$$C(k+1) = C(k) + \Delta C(k) \tag{4}$$

As shown in Fig. 2, the fuzzy controller consists of four components, namely:

- the *rule base* component, that contains a set of rules through which fuzzy control decisions are taken;
- the *fuzzifier* component, that converts numeric values of control inputs into equivalent fuzzy values by means of the *input membership functions*;
- the *inference system* component, that applies the rules of the rule base according to the "fuzzified inputs" (i.e., the output of the fuzzifier component) and generates *fuzzy conclusions*;
- the *defuzzifier* component, that combines the fuzzy conclusions and converts them to a single numerical output value.

The details concerning the way how a fuzzy logic system works is out of the scope of this paper. The interested reader can refer to [3], [14], [15] for more details.

In the rest of this section, we describe in detail how we implemented each of the above components.

1) The Rule Base Component: The *rule base* component contains a set of rules that are in the form of "IF-THEN" statements, and are defined by means of linguistic variables, that take linguistic values and represent the control inputs and outputs. Rules translate the control knowledge into a form that can be used by the inference system component. In particular, each rule defines the conditions under which it can be applied (the "IF" part, or *premise*), and the output deriving from its application (the "THEN" part, or *consequent*). Given that in fuzzy controllers these rules are based on "heuristic" (i.e., expert) control knowledge [3], to gain this knowledge we carried out several experiments (this is considered standard practice in the literature [6], [9]) to come up with a good set of rules. First of all, we defined a preliminary rule base and then we run the controlled application under different workloads to study the reactivity of Fuzzy-Q&E. If Fuzzy-Q&E was too aggressive, we modified the rule base in order to reduce the magnitude of the variations produced by Fuzzy-Q&E. Conversely, we acted on the rule base to increase these magnitudes. Once we came up with a well-defined rule base, we run again the controlled application to tune the certainty of each membership function.

For the design of our rule base, we use the following linguistic variables and values:

TABLE I: Fuzzy-Q&E – The rule base.

"ΔC"		"R_{gain}" *LOW*	*FINE*	*HIGH*
"C_{res}"	*LOW*	BUP	UP	UP
	FINE	UP	STY	DWN
	HIGH	STY	DWN	BDW

- "R_{gain}" and "C_{res}", which are the linguistic counterparts of R_{gain} and C_{res} control inputs, that can take *LOW*, *FINE*, or *HIGH* as linguistic values;
- "ΔC", which is the linguistic counterpart of the ΔC control output, that can take *BUP*, *UP*, *STY*, *DWN*, or *BDW* as linguistic values (which stand for "big up", "up", "stay", "down" and "big down", respectively).

The rules used by Fuzzy-Q&E are reported in Table I. Each cell of the table defines a particular rule. For instance, cell (LOW, LOW) corresponds to the rule: *if "C_{res}" is LOW and "R_{gain}" is LOW then "ΔC" is BUP*. This rule encodes the control knowledge stating that if the CPU CAP value used by the VM is close to that allocated by the controller (encoded by the *"C_{res}" is LOW* condition), and the 95^{th} percentile of the observed response times is close to or higher than the reference one (encoded by the *"R_{gain}" is LOW* condition), then the fuzzy controller has to significantly increase the allocated CPU CAP value (encoded by *"ΔC" is BUP* condition).

2) The Fuzzifier Component: The *fuzzifier* component carries out the so called fuzzification process, which assigns linguistic values to numeric inputs and determines their *certainties* μ (also referred to as *degrees of membership*) by using the *input membership functions*. A *membership function* is a graphical representation of the magnitude of participation of each input. As shown in Fig. 3, we decide to quantify the linguistic values by means of *triangular-shaped membership functions*, which are one of the most commonly used membership functions in practice. In particular, Fig. 3a and Fig. 3b show the membership functions for the fuzzy control inputs R_{gain} and C_{res}, respectively.

To denote the certainty of a linguistic value m (assumed by a specific input linguistic variable), we use the notation $\mu(m)$. For example, if R_{gain} is 20, the corresponding value of the associated linguistic variable "R_{gain}" is FINE, and its certainty $\mu(\text{FINE})$ is 1, since the numeric value of R_{gain} projects up to a peak of the membership function corresponding to the linguistic value FINE (see Fig. 3a).

3) The Inference System Component: The *inference system* component determines which rules (from the rule base) should be applied to reach fuzzy conclusions, according to the fuzzified inputs coming from the fuzzifier component. Let us denote with $\mu(m, n)$ the premise certainty of a rule (m, n) where m and n are the membership functions. Following the Mandani's "max-min" inference mechanism [16], the rules to be activated are defined as a set of rule (m, n) such that $\mu(m, n) > 0$, where $\mu(m, n) = \min(\mu(m), \mu(n))$. For example, if the input variables C_{res} is 22 (i.e., "C_{res}" is either LOW or FINE) and R_{gain} is 20 (i.e., "R_{gain}" is LOW), then the certainties $\mu(\text{FINE}, \text{LOW})$ and $\mu(\text{FINE}, \text{FINE})$ of rule (FINE,LOW) and rule (FINE,FINE) are 0.4 and 0.8, respectively.

4) The Defuzzifier Component: The *defuzzifier* component combines the rules activated by the inference system using the "center average" method [3] and calculates the control output. The fuzzy rules activated by the inference system may generate multiple fuzzy conclusions. In the center average method, the numeric value of the control output is computed as a weighted average of the certainty of the fuzzy conclusions, where the weight of each conclusion is the center point of the *output membership function* (see Fig. 3c). Similarly to what we do for the fuzzifier component, we quantify the linguistic values by means of *triangular-shaped membership functions*.

IV. Experimental Evaluation

In order to asses the capability of our approach, we performed an extensive performance evaluation on a testbed, composed by a physical computing infrastructure running the Xen hypervisor [1], on which we implemented a prototype of Fuzzy-Q&E, as well as of *DynaQoS* [6], a state-of-the-art fuzzy controller for cloud applications against which we compared our approach.

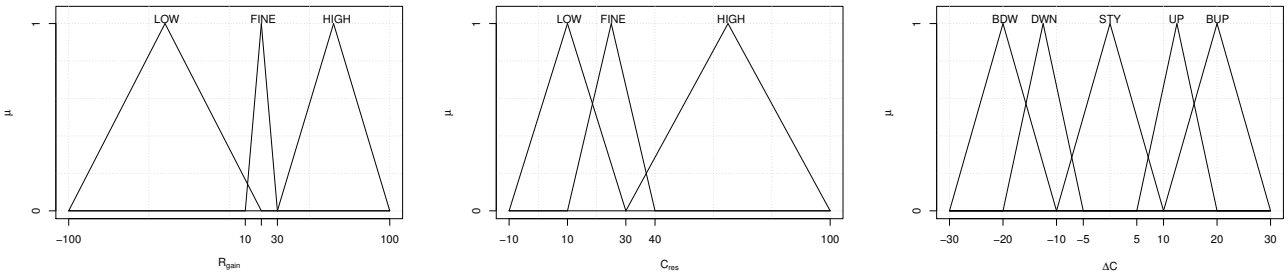
In the rest of this section, we first describe the testbed, and then we discuss the experimental scenario we considered for our experiments and the experimental results we obtained.

A. Experimental Testbed

For our experiments, we considered a physical infrastructure consisting of two identical machines, each one equipped with two 3.0 GHz Intel Xeon 5160 Dual-Core CPU and with 10GB of RAM, and running the Fedora 18 Linux distribution. These machines are connected by a Gigabit Ethernet network.

The first physical machine was used to run virtualized applications, and run the Xen hypervisor (version 4.2). On this machine, we run both the virtualized applications, the VCPU Utilization Collector component (see Section III-B), and one or multiple instances of the Fuzzy-Q&E controller (one for each application tier). We configured the Xen hypervisor to use the Xen *credit scheduler*, that provides suitable mechanisms to specify and enforce CAP allocations to the various VMs. The VCPU Utilization Collector component was implemented as a standalone program and was used to collect VCPU usage statistics through the Xen API (inside the `dom0`). Finally, the Fuzzy-Q&E controller was implemented in `C++` using the *fuzzylite* library [17] and run on the virtualized server (inside the `dom0`) as a standalone program, with one instance for each hosted VM.

The second physical machine was instead used to run the workload generator for the above virtualized applications (running on the first physical machine), and to run the Response Time Collector components (see Section III-B). In particular, for our experiments we considered the *RUBiS* application [18], an auction site prototype modeled after eBay.com, which uses a multi-tier setup consisting of a Web and a database tier. Among the various incarnations of RUBiS, we opted for the one based on Java Servlets from the *OW2 Consortium* [5] (version 1.4.3). Each tier of RUBiS runs in its own VM, that we configured with one virtual CPUs (VCPUs) and 1GB of RAM. We set up the Xen hypervisor so that one physical CPU (two cores) were pinned to `dom0` and the other one (two cores) were pinned to the two VMs running the RUBiS tiers.



(a) MF for the R_{gain} input variable. (b) MF for the C_{res} input variable. (c) MF for the ΔC output variable.

Fig. 3: Fuzzy-Q&E – The membership functions (MFs).

To generate the workload for RUBiS, we used the *RAIN* workload generator toolkit [19], that emulates multiple concurrent RUBiS clients and that has been specifically extended by us to support the RUBiS application [20]. The Response Time Collector component was implemented as a standalone program and was used to collect response time observations from the RAIN toolkit.

B. Experimental results

RUBiS provides workloads of different mixes and time-varying intensity; among them, we considered the so called "browsing-mix", that simulates a user browsing through an auction site. To do so, we configured the RAIN toolkit with the default RUBiS "browsing-mix" matrix and with a negative exponentially distributed think-time with an average of 90 seconds.

To compute the SLO value to use in our experiments and to fit with the available computing capacity of our testbed, we used a benchmark-like approach similar to the one described in [21], [22] for application profiling. From these benchmarks, we obtained a SLO value of 0.58 seconds. Then, in order to compute the maximum workload supported by our testbed to let the SLO value be a feasible target (i.e., that the 95^{th} percentile does not exceed 0.58 seconds), we run another series of RUBiS benchmarks at full CPU capacity and by progressively increasing the number of clients. From these benchmarks, we found a maximum workload of 40 RUBiS clients.

To assess the capability of Fuzzy-Q&E to react to different workload patterns while respecting the SLO and, at the same time, improving server consolidation, we considered a scenario with both variable number of RUBiS clients and different CPU utilization patterns. Specifically, we performed a series of runs, each of which lasted 2400 seconds, including 300 seconds of both ramp-up and ramp-down phase. For each run, we executed an instance of RUBiS, by hosting the two related VMs into the same physical machine, with a time-varying workload characterized by a mean user think-time of 90 seconds and by a number of clients that increased from 10 to 40, and then decreased to 20, every 600 seconds (all of this timings do not include the ramp-up and ramp-down phases). We parameterized our components in the following way. For the Fuzzy-Q&E controller, we set the control time to 10 seconds, while for the VCPU Utilization Collector component, we set the sampling time to 3 seconds and the smoothing factor β to 0.9.

TABLE II: Experimental Results – Performance Summary.

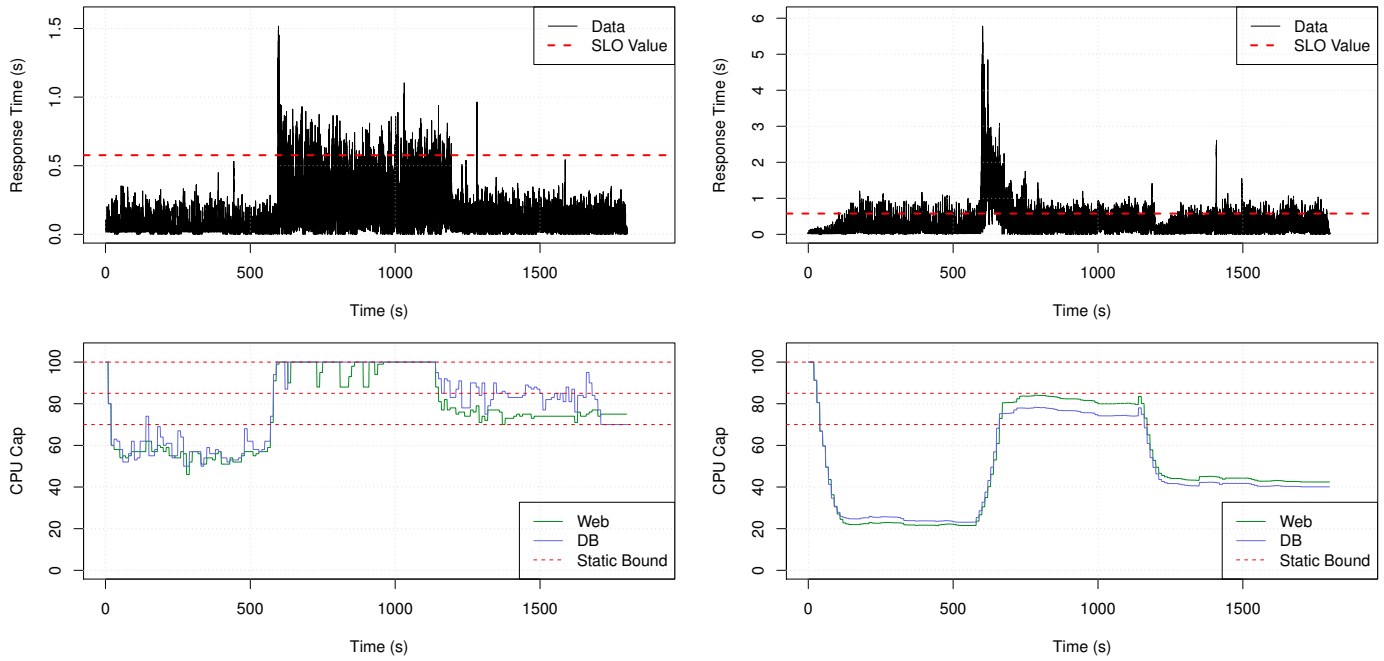
Approach	Response Time		CPU CAP	
	95^{th} Perc.	Δe_{SLO}	Web VM	DB VM
Fuzzy-Q&E	0.39	-0.33	$76.50\,(17.50)$	$80.96\,(17.57)$
DynaQoS	0.65	0.12	$50.36\,(24.30)$	$48.63\,(21.70)$

Furthermore, to understand how Fuzzy-Q&E compares to current state-of-the-art solutions, we implemented the *DynaQoS* approach [6], and run it against the same scenario, by setting the following parameters. We set the control time to 10 seconds and the discount factor γ to 0.8, as suggested by the authors in their paper (see [6] for more details about the meaning of these parameters).

For each of the above run, we collected the response times of RUBiS operations, and the CPU CAP values of the VMs running the RUBiS application. For each of these quantities, we computed the mean and the standard deviation, and averaged them over the number of runs. In addition, we computed the 95^{th} percentile of the response time empirical distribution, and used its average over the number of runs to compute the metric Δe_{SLO}, which represents the relative deviation of the average 95^{th} response time percentile from the SLO value. The sign of Δe_{SLO} gives indication on SLO preservation; specifically, a negative sign means that the SLO has been preserved since the average 95^{th} response time percentile is lower than the SLO value, while a positive sign means that SLO has been violated. The magnitude of Δe_{SLO} indicates how far the average 95^{th} response time percentile is from the SLO value; specifically, a small value means that the average 95^{th} response time percentile is close to the SLO value, while a large value means that it is far from the SLO value.

The results of the experiments are reported in Table II, where the column "Approach" indicates to what approach, among Fuzzy-Q&E and DynaQoS, the statistics in the same row refer; the column "Response Time" reports the 95^{th} percentile and the Δe_{SLO} metric of the observed response time of RUBiS operations; finally, the column "CPU CAP" shows the mean and the standard deviation (inside parenthesis) of the CPU CAP values assigned to the RUBiS VMs by the given approach.

As can be observed from the table, Fuzzy-Q&E is able to guarantee the SLO (i.e., Δe_{SLO} is negative), while DynaQoS is not, and indeed it exceeds the SLO value by more than 12%.



(a) Fuzzy-Q&E.

(b) DynaQoS.

Fig. 4: Experimental Results – Response times and CPU CAP values obtained with Fuzzy-Q&E and DynaQoS.

As a matter of fact, DynaQoS is unable to react in time to workload changes, thus causing the response time to largely increase in magnitude and to exhibit a higher variability. This can also be seen both in Fig. 4 and in the box-plot presented in Fig. 5.

Specifically, in Fig. 4a, we can observe that Fuzzy-Q&E is able to readily adapt to current load conditions by varying the CPU CAPs to suitable values. The same does not happen for DynaQoS, which, as shown in Fig. 4b, becomes aware of changes to load conditions too late to prevent too many SLO violations. In fact, for DynaQoS, there are SLO violations for all the duration of the experiment; while, for Fuzzy-Q&E, the times that the response time exceeds the SLO value are nearly concentrated in the high-load phase (i.e., when the number of concurrent RUBiS users is 40).

From Fig. 5, we can instead have better insights on the orders of magnitude of the response time obtained with the two approaches. In particular, we can note that the observed response times obtained with DynaQoS span several orders of magnitude (by reaching values that are nearly 10 times larger than the SLO value), while the ones obtained with Fuzzy-Q&E grows moderately (which are no larger than 2.6 times the SLO value).

Moreover, we want to point out that Fuzzy-Q&E is able to better capture load variations and differences between the various application tiers, and thus can improve server consolidation. This can be observed in the lower plot of Fig. 4a, where the trend exhibited by the CPU CAP curve for the Web tier is different from the one related to the database tier. Conversely, the CPU CAP assigned by DynaQoS to the two RUBiS tiers is nearly the same (see the lower plot of Fig. 4b). This means that Fuzzy-Q&E can potentially improve server consolidation since, with respect to DynaQoS, it can better exploit different load conditions inside the VMs hosted by the same physical machine.

Finally, with respect to approaches based on static capacity allocation, the gain obtained by Fuzzy-Q&E for server consolidation increases as the fixed CAP, henceforth referred to as *static bound*, assigned by a static approach increases. This can be observed in the lower plot of Fig. 4b, where the considered three possible static allocations corresponding to a CAP of 70, 85, and 100, and represented them by three red dashed horizontal lines. From the figure we can note that the higher is the static bound, the lower is the opportunity to consolidate several VMs on a same physical machine with the static allocation approach, and thus the higher is the advantage obtained by using Fuzzy-Q&E in server consolidation.

Let us now quantify the energy savings achieved by Fuzzy-Q&E thanks to its ability of consolidating a set of VMs on a smaller number of physical machines [23]. To do so, we measured the energy consumed to run *two instances* of the RUBiS applications, under the same workload and SLO specifications considered before, in two distinct scenarios:

- *Static allocation*: to respect SLO values, all the four VMs (two for each RUBiS instance) were allocated a CAP value equal to 100. Thus, to run the four VMs, we needed to power on *two* physical machines since the capacity provided by a single physical machine was not enough to accommodate the aggregate CAP value (i.e., 400), because a suitable share of physical capacity had to be allocated to Xen services (otherwise the system would have become unresponsive and, eventually, would have crashed).

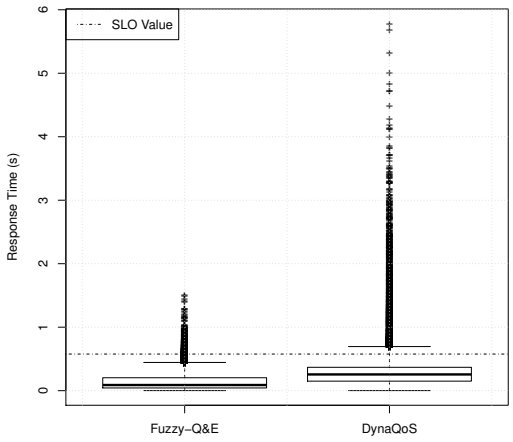


Fig. 5: Experimental Results – Response time distribution obtained with Fuzzy-Q&E and DynaQoS.

- *Controlled allocation*: all the four VMs were allocated on the same physical machine, and the capacity allocated to each one of them was set by using Fuzzy-Q&E.

Power measurements, collected by means of a *Watts Up? .NET* power meter connected to the physical machines, showed that in the static allocation scenario 376 Wh were consumed, while only 207.9 Wh were necessary in the controlled allocation scenario, that represent nearly a 44.7% saving.

We can therefore conclude that Fuzzy-Q&E not only is able to preserve SLO, but, with respect to DynaQoS, is able (1) to react in time to time-varying and bursty workloads, and (2) to show a much smaller variability in response time. Moreover, with respect to approaches based on static capacity allocation, Fuzzy-Q&E is also able to improve server consolidation and thus energy-efficiency. Finally, since in [6] it has been shown that DynaQoS outperforms other state-of-the-art control-theoretic approaches, we believe that Fuzzy-Q&E has the potential to outperform these approaches as well. Further experimentation, carried out with a wider set of applications and workloads, is necessary in order to gain confidence about the generality of our approach.

V. Related Works

The provisioning of physical resources to guarantee application QoS is an active research topic in the last years but, to the best of our knowledge, only few works exploit the fuzzy logic extensively. The approach taken by such works to the fuzzy logic usually falls into two different categories, whereby (1) the fuzzy logic is used to model the behavior of a system, or (2) the fuzzy logic is used to design a controller to act on the system at run-time in order to guarantee a specific QoS. Works like [11], [24] belong to the first approach, while work such as [6], [9], [25], [26] belong to the second approach.

Regarding the first approach, in [11], [24] the authors use fuzzy logic to estimate the relationship between performance and CPU utilization (and also energy consumption, in [24]). These works use the output of the fuzzy logic to predict the performance of the system. This approach may be inaccurate since the performance of the application in a dynamic cloud environment may vary significantly due to various measures (CPU usage, memory usage, network bandwidth, hypervisor load, and so on) that makes hard to obtain any accurate performance forecast.

Concerning the second approach, in [6], [9], [26], the authors propose a fuzzy controller for the allocation of virtualized resources in order to respect the application response time. Both works only consider the response time and its deviation from the SLO value as input parameters to the controller. Instead, in this paper, we combine the information regarding the response time with the VCPU utilization. The combination of these two parameters allows our Fuzzy-Q&E controller to gain more knowledge on the system load, thus resulting in a more accurate CPU capacity allocation. In [25], the authors propose a novel load balancing algorithm using fuzzy logic in cloud computing. In particular, the fuzzy controller requires two input data like processor speed and assigned load of VM and provides the balance load to reduce the application response time. This approach can be exploited only for CPU-intense applications where the response time is related to CPU speed. Conversely, our approach is more general since it does not make any assumption concerning the CPU speed of the machines available.

As a final remark, we want to mention that the literature provides various works that use model-based linear feedback control (e.g., [27], [28]). To work, these approaches build (either offline or online) a linear mathematical model to describe the relationships between control inputs and control outputs. There are some possible issues concerning these approaches: (1) since a computing system (i.e., the system they try to model) is inherently nonlinear [4], they have to use linearization techniques around some operating region to make locally linear the input-output relationships; (2) in dynamic systems (like cloud computing systems are) characterized by time-varying and bursty workloads, could be hard to know such operating regions and thus might be impossible to build accurate linear models (even if adaptive techniques are employed). With respect to such works, our approach does not require the creation of mathematical models and thus it can easily handle with nonlinear relationships.

VI. Conclusions

In this paper, we presented *Fuzzy-Q&E*, a fuzzy controller that is able to allocate to the set of VMs associated to a cloud applications the minimum amount of physical capacity needed to meet the application SLO, even in the presence of time-varying and bursty workloads.

The performance of Fuzzy-Q&E has been assessed by means of an experimental evaluation on a real testbed. To do so, we implemented a real prototype of Fuzzy-Q&E and used it to control the RUBiS application. Furthermore, we compared the obtained results with the ones achieved by DynaQoS, a state-of-the-art approach. The results show that, unlike DynaQoS, Fuzzy-Q&E is able (1) to meet SLOs, (2) to react in time to non-stationary and bursty workloads, and (3)

to show a moderate variability in response time. Furthermore, with respect to approaches based on static capacity allocation, Fuzzy-Q&E is also able to improve server consolidation since, instead of keeping fixed the CAP needed during the peak load (which is useless during low-load workload conditions), it can exploit load variations by dynamically varying the CAP assigned to the VM. As a consequence, Fuzzy-Q&E can also improve energy-efficiency since a better server consolidation can result in a reduced number of switched-on physical machines.

Regarding possible future works there are some interesting activity that can be carried on. First of all, we want to evaluate our system with other type of applications and under different non-stationary workload conditions. Then, we would like to integrate our work with current state-of-the-art power- and performance-aware resource management frameworks for cloud computing systems (like [29], [30]). Furthermore, we would like to extend the system to also take into consideration other resource types (e.g., memory and disks). Finally, we plan to integrate our system into a cloud management platform (like Eucalyptus [31] and OpenStack [32]), and to support different hypervisors (e.g., KVM [33]).

REFERENCES

[1] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *Proc. of the* 19th *ACM Symposyum on Operating Systems Principles (SOSP'03)*, 2003, pp. 164–177.

[2] R. Singh and U. Sharma and E. Cecchet and P. Shenoy, "Autonomic mix-aware provisioning for non-stationary data center workloads," in *Proc. of the* 7th *International Conference on Autonomic Computing (ICAC'10)*, 2010.

[3] G. Chen and T.T. Pham, *Introduction to Fuzzy Sets, Fuzzy Logic, and Fuzzy Control Systems*. CRC Press, 2001.

[4] J.L. Hellerstein and Y. Diao and S. Parekh and D.M. Tilbury, *Feedback Control of Computing Systems*. Wiley-IEEE Press, 2004.

[5] OW2 Consortium, "RUBiS: Rice university bidding system," 2008. [Online]. Available: http://rubis.ow2.org/index.html

[6] J. Rao, Y. Wei, J. Gong, and C.-Z. Xu, "QoS Guarantees and Service Differentiation for Dynamic Cloud Applications," *IEEE Transactions on Network and Service Management*, vol. 10, no. 1, Mar. 2013.

[7] D. Borgetto, H. Casanova, G. D. Costa, and J.-M. Pierson, "Energy-aware service allocation," *Future Generation Computer Systems*, vol. 28, no. 5, pp. 769–779, 2012.

[8] D. Gmach, J. Rolia, L. Cherkasova, and A. Kemper, "Resource pool management: Reactive versus proactive or let's be friends," *Computer Networks*, vol. 53, no. 17, pp. 2905–2922, Dec. 2009.

[9] P. Lama and X. Zhou, "Efficient server provisioning with control for end-to-end response time guarantee on multitier clusters," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 1, pp. 78–86, 2012.

[10] A. Verma, P. Ahuja, and A. Neogi, "pMapper: Power and migration cost aware application placement in virtualized systems," in *Proc. of the* 9th *ACM/IFIP/USENIX International Conference on Middleware (Middleware'08)*, 2008, pp. 243–264.

[11] P. Lama and X. Zhou, "PERFUME: Power and performance guarantee with fuzzy MIMO control in virtualized servers," in *Proc. of the 2011 IEEE* 19th *International Workshop on Quality of Service (IWQoS'11)*, 2011, pp. 1–9.

[12] R. Jain and I. Chlamtac, "The P2 algorithm for dynamic calculation of quantiles and histograms without storing observations," *Communications of the ACM*, vol. 28, no. 10, pp. 1076–1085, Oct. 1985.

[13] S. Roberts, "Control chart tests based on geometric moving averages," *Technometrics*, vol. 1, no. 3, pp. 239–250, 1959.

[14] E. Mamdani, "Application of fuzzy logic to approximate reasoning using linguistic synthesis," *IEEE Transactions on Computers*, vol. 26, no. 12, pp. 1182–1191, 1977.

[15] L. A. Zadeh, "Fuzzy sets," *Information and Control*, vol. 8, no. 3, pp. 338–353, 1965.

[16] E. Mamdani and S. Assilian, "An experiment in linguistic synthesis with a fuzzy logic controller," *International Journal of Man-Machine Studies*, vol. 7, no. 1, pp. 1–13, 1975.

[17] J. Rada-Vilela, "fuzzylite: A fuzzy logic control library written in C++," 2013. [Online]. Available: http://www.fuzzylite.com

[18] C. Amza, A. Chanda, A. Cox, S. Elnikety, R. Gil, K. Rajamani, W. Zwaenepoel, E. Cecchet, and J. Marguerite, "Specification and implementation of dynamic web site benchmarks," in *Proc. of the 2002 IEEE International Workshop on Workload Characterization (WWC-5)*, 2002, pp. 3–13.

[19] A. Beitch, B. Liu, T. Yung, R. Griffith, A. Fox, and D. A. Patterson, University of California at Berkeley, Technical Report UCB/EECS-2010-14, Feb. 2010.

[20] M. Guazzone, "The RUBiS workload implementation for RAIN," 2013. [Online]. Available: https://github.com/sguazt/rain-workload-toolkit

[21] L. T. Yang et al., "Cross-platform performance prediction of parallel applications using partial execution," in *Proc. of the 2005 ACM/IEEE Conference on Supercomputing (SC'05)*, 2005.

[22] T. Wood et al., "Profiling and modeling resource usage of virtualized applications," in *Proc. of the* 9th *ACM/IFIP/USENIX Int. Conf. on Middleware (Middleware'08)*, 2008, pp. 366–387.

[23] R. Talaber, T. Brey, and L. Lamers, "Using virtualization to improve server efficiency," The Green Grid, White Paper 19, 2009.

[24] P. Lama and X. Zhou, "NINEPIN: Non-invasive and energy efficient performance isolation in virtualized servers," in *Proc. of the* 42nd *Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'12)*, 2012, pp. 1–12.

[25] S. Sethi, A. Sahu, and S. K. Jena, "Efficient load balancing in cloud computing using fuzzy logic," *IOSR Journal of Engineering*, vol. 2, no. 7, pp. 65–71, 2012.

[26] Y. Guo, P. Lama, and X. Zhou, "Automated and agile server parameter tuning with learning and control," in *Proc. of* 26th *IEEE International Parallel & Distributed Processing Symposium (IPDPS'12)*, 2012, pp. 656–667.

[27] P. Padala, K.-Y. Hou, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, and A. Merchant, "Automated control of multiple virtualized resources," in *Proc. of the* 4th *ACM European Conference on Computer Systems (EuroSys'09)*, 2009, pp. 13–26.

[28] M. Guazzone, C. Anglano, and M. Canonico, "Energy-efficient resource management for cloud computing infrastructures," in *Proc. of the* 3rd *IEEE International Conference on Cloud Computing Technology and Science (CloudCom'11)*, 2011.

[29] C. Mastroianni, M. Meo, and G. Papuzzo, "Self-economy in cloud data centers: Statistical assignment and migration of virtual machines," in *Proc. of the* 17th *International European Conference on Parallel and Distributed Computing (EuroPar'11)*, 2011.

[30] M. Guazzone, C. Anglano, and M. Canonico, "Exploiting VM migration for the automated power and performance management of green cloud computing systems," in *Proc. of the* 1st *International Workshop on Energy-Efficient Data Centres (E2DC'12)*, 2012.

[31] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, "The Eucalyptus open-source cloud-computing system," in *Proc. of the* 9th *Cluster IEEE/ACM International Symposium on Computing and the Grid (CCGRID'09)*, 2009.

[32] The OpenStack Foundation, "OpenStack: The open source cloud operating system," 2010. [Online]. Available: http://www.openstack.org

[33] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori, "KVM: the linux virtual machine monitor," in *Proc. of the 2007 Ottawa Linux Symposium*, vol. 1, 2007, pp. 225–230.