A Fuzzy Approach based on Heterogeneous Metrics for Scaling Out Public Clouds

Valerio Persico, Domenico Grimaldi, Antonio Pescapè, Alessandro Salvi, and Stefania Santini

Abstract—Thanks to resource elasticity, cloud systems allow to build high performance applications by dynamically adapting resources to workload dynamics. In this paper, we present a novel approach for horizontally scaling cloud resources. The approach is based on an optimized feedback control scheme that leverages fuzzy logic to self-adjust its parameters in order to cope with unpredictable and highly time-varying public-cloud operating conditions. The proposed approach takes as input heterogeneous monitoring metrics related to distinct aspects of interest (i.e., CPU and network load) merged through a fitness function. Therefore, it is able to accomplish the application needs from different viewpoints. The extensive experimental evaluation performed in the Amazon EC2 environment showed how the proposed approach is robust against a number of realistic workloads—also when VM failures happen—and that it is flexible, as being suitable for applications with different needs. Finally, it also achieves better performance when compared to previously proposed solutions.

Index Terms—Cloud Computing, Cloud Control, Autoscaling, PID, Fuzzy Logic, QoS/QoE Metrics.

1 Introduction

Cloud computing (hereinafter, *cloud*) is nowadays a *de facto* standard in the IT world for providing services to final users. An increasing number of Internet services as well as private IT infrastructures have now been moved to the cloud, mainly due to several economical and technical benefits, e.g. on-demand services, reduced costs, optimized hardware and software resources utilization, and performance flexibility. Therefore, companies rely on cloud for different purposes, such as running batch jobs and hosting web applications, or for data storage and backup. The *pay-as-you-go* pricing model is the characteristic that more directly captures the appealing economic benefit to the customer [1]. Indeed, the absence of up-front expenses allows capital to be redirected to core business investments. This is achieved through *resource elasticity*—i.e., the ability to add or remove resources at a fine grain and with a lead time of minutes rather than weeks—that allows matching resources to workload much more closely. For instance, a cloud customer can decide to start on-demand new servers or allocate more storage capacity just when needed, and without any up-front provisioning. In this way, it is possible to dramatically raise the server utilization level that is estimated to be very low without cloud-based approaches—from 5% to 20% [2]—due to typical *overprovisioning* practices needed to properly manage peak workload [1]. Therefore a wide number of different solutions has been proposed to implement resource elasticity: they range from social-network sentiment analysis [3] or market-based methods [4], [5], to control strategies with diverse degrees of complexity and working at different levels [6], [7], [8], [9], [10]. Approaches for scaling cloud resources can be coarsely categorized as either horizontal scaling (or *scaling out*, that consists in adding new server replicas to distribute load among all available replicas through load balancers) or vertical scaling (also known as *scaling up*, and consisting in on-the-fly changing the resources assigned to an already running instance, for example, allocating more computational or memory resources to a running virtual machine (VM)).

- *V. Persico, D. Grimaldi, A. Salvi, and S. Santini are with the University of Napoli "Federico II" (Italy).*
  E-mail: {valerio.persico, alessandro.salvi, santini}@unina.it
  dom.grimaldi@studenti.unina.it,
- *A. Pescapè is with the University of Napoli "Federico II" (Italy) and with NM2 srl (Italy).*
  E-mail: pescape@unina.it

This work is partially funded by the MIUR in the context of art. 11 DM 593/2000 for NM2 srl. The experimental work in this paper was also supported by a grant provided by Amazon AWS in Education.

1.1 Motivation

Building high-quality cloud applications is a critical research problem. Indeed, *appropriately* dimensioning resources in real time is a crucial issue in practical scenarios, where elasticity has to be put into effect to dynamically scale resources according to changing demands. Implementing a valuable resource-allocation mechanism is of the utmost importance to obtain a suitable trade-off between cost and performance [1]. Since applications may face large fluctuating loads [11], it would be desirable to free the cloud customers from the burden of deciding how to adjust resources in presence of unplanned and unpredictable spike loads. In other words, it would be desirable to have an automatic strategy to adapt the amount of resources to be allocated on the base of the specific needs at any given time.

Approaches for automatically scaling out cloud resources (i.e. *autoscaling* approaches) like threshold-based rules are very popular. Cloud providers such as Amazon EC2 and third-party tools like RightScale make these appealing policies available to cloud customers. However, setting thresholds is a per-application task, and requires a deep understanding of workload trends. Thus, several solutions overcoming the limitations of threshold-based approaches have been proposed in the literature involving several

components of the infrastructure and taking advantage of different kinds of techniques like queuing theory, reinforcement learning, workload prediction, or control theory [11]. Notwithstanding the above, public-cloud customers have very limited choice for the autoscaling policies to implement, being in fact restricted to the limiting opportunities providers usually make available. To improve the quality of adaptation, modern autoscaling systems often cover more sophisticated aspects, including modelling, determining granularity of control, and decision making [12].

Automatic resources allocation techniques deeply rely on the accurate real-time estimation of the actual conditions of the cloud system and its performance evaluated through metrics representative of the QoS according to the specific application requirements [12], [13]. Usually, allocation rules evaluate cloud performance from a single viewpoint, e.g., they consider just one metric at a time, such as CPU utilization, memory consumption etc. [11]. The problem of identifying suitable metrics to rely on for implementing autoscaling techniques is further exacerbated when observed from the angle of public-cloud customers. This is because of the limited information publicly exposed through the poor provider-to-customer interface and the ways it can be effectively leveraged. In more details, in accordance with the recent literature the metrics related to the shared intra-datacenter network infrastructures should also be considered towards complete application scalability [14]. Nonetheless cloud network resources and the related performance are often overlooked, although this aspects could be critical. This happens both when different VMs share the same network (thus competing for resources in terms of available bandwidth) and when each VM is associated to a dedicated network slice. See [15] for further details. Furthermore, recent experimental results [16], [17] confirm that commercial providers guarantee to newly instantiated VMs a fixed—although *a priori* unknown—network slice in terms of bandwidth based on the cost.

In conclusion, it would be desirable for public cloud customers to have advanced autoscaling strategies available, to be provided with a powerful tool (i.e., able to capture the needs of the application from multiple points of view) to simply implement resource elasticity in public clouds.

1.2 Contribution

In this paper, we propose a novel Fuzzy-PID architecture to automatically scale out cloud resources at VM-granularity and according to heterogeneous metrics. These metrics are merged to feed the control logic by adopting recent methodologies designed to extract a representative QoS index from heterogeneous observations. The architecture leverages fuzzy logic to support resource scaling decisions such to guarantee a desired Service Level. The proposed control strategy is able to counteract the presence of large fluctuating loads, with no need of either previous knowledge of the system behavior or the estimation of disturbances acting on the cloud environment: therefore it is suitable to be enforced by general cloud customers in public clouds.

In more details, the metrics taken into account to implement our control strategy are related to different aspects—i.e., computational and network capability—so that the resource provisioning mechanism accomplishes the needs of applications running onto the cloud from several points of view. The control strategy implements a Proportional Integrative Derivative (PID) feedback control. Fuzzy logic implements a *gain-scheduling* policy to adapt the control action with respect to the conditions of the public cloud environment which are not easy to predict by customers having limited or null visibility of the underlying management strategies enforced by cloud providers.

This work extends the literature about control solutions for scaling cloud resources in the points summarized below:

i. heterogeneous metric observations are properly merged together in a single performance index and allow to take into account the current state of the managed cloud applications from different angles at once;
ii. the designed architecture requires neither any detailed knowledge of the dynamics of the controlled cloud infrastructure nor a performance model of the application; this does not limit its applicability to in-house environments and makes the proposed solution also suitable to be applied on public-clouds as it just relies on the knowledge available to the general customer;
iii. the designed fuzzy-based gain scheduling algorithm provides robustness with respect to synthetic and real-trace workloads characterized by high rate of variability and does not require any detailed knowledge or any prior information about the current workload, its on-line measurement/estimation, or the performance model of the application;
iv. the proposed solution has been implemented and extensively validated in a real public-cloud environment (AWS EC2), and therefore only taking advantage of the knowledge available to the general customers; experimental results show how the proposed approach is robust against a number of realistic workloads, also when VM failures happen; compared to solutions proposed in the literature, it achieves better performance;
v. the code and the material needed to evaluate the proposed approach is publicly released, to foster the replication of the experimental analyses, thus allowing the comparison to alternative approaches.

The paper is organized as follows. Sec. 2 provides an overall picture of the related literature and positions the paper accordingly. Sec. 3 describes the problem statement introducing names and definitions and details the designed architecture with all its composing blocks. Sec. 4 provides details on the control strategy. In Sec. 5 we present the evaluation of our proposal, first detailing the experimental setup (Sec. 5.1), and then discussing all the results (Sec. 5.2). Finally, conclusions are drown in Sec. 6.

2 Related work

Approaches for automatically scaling resources that leverage cloud elasticity have recently attracted the interest of the scientific community. Therefore, a number of different solutions has been proposed to deal with the dynamics of cloud systems without human intervention. While their main goal is to optimize either the QoS (e.g., in terms of service performance or availability) or a cost objective (e.g., power consumption or number of active VMs), existing solutions cover a set of specific goals, such as self-configuring, self-healing, self optimizing, and self-protecting [12].

The **granularity of control** plays a major role, as it determines the objectives to be considered in the autoscaling decision process. Based on the control granularity, we may distinguish solutions working at different levels, such as (i) *cloud level* (managing utility, profits, and availability of the overall cloud system) [18]; (ii) *physical machine level* (managing QoS interference caused by co-hosted VMs) [19]; (iii) VM level (assigning resources to VMs are usually adapted in isolation) [10], [20], [21]; (iv) service level (aim at independently scaling an application or one of its tier) [13], [22]; sometimes, a one-to-one mapping between applications and VMs is also assumed therefore some proposals can be categorized as working at either the VM or the service level granularity [12]. Our solution manages cloud resources at VM-granularity (i.e., activating or deactivating VMs that considered as black boxes) to control the QoS of a service.

The granularity of the control also determines how the autoscaling problem is addressed and how the proposed solutions are evaluated. Indeed, the specific models and the hypotheses sustaining each of the proposed solutions also strongly impact their range of **applicability** and subsequently the **evaluation process** these solutions are subjected to. To the best of our knowledge, a limited set of control solutions proposed in the literature has been applied and evaluated onto public clouds and has demonstrated the validity of proposed approach through prototype implementations running on Amazon EC2 or Microsoft Azure [18], [23], [24], [25]. Due to system complexity some of the works do not address the problem in real systems, but propose the extensive use of *cloud simulators* to mimic the dynamic response of the cloud system under the proposed control action [11], [12], [26]. One of the main drawbacks of these proposals is that they rely on performance models instead of reality, hence results strongly depend on the reliability of the simulations. In fact, the experimental validation of cloud control strategies in a real environment has been often addressed by designing custom in-house testbeds such as *private cloud deployments* or simply server clusters [8], [20], [27]. More in general, all these approaches—being tailored for in-house testbeds—strongly leverage the precise knowledge of the inner mechanisms of the cloud systems under their control. Since the level of abstraction available in public clouds obfuscates it for commercial and security reasons, these approaches are not directly exploitable by common customers in public cloud environments as is. The approach we propose in this work is designed for being adopted even when no privileged point of view beyond the limited one of the cloud-customer is available. Therefore, differently than the majority of the advanced solutions discussed above, it is suitable to be put into practice also onto public-cloud environments.

An open challenge related to the implementation of a high scalable adaption mechanism a number of works try to address is the **characterization of the workload type and its accurate online profiling**, exploiting a Kriging model [9], Kalman filters [10], taking into consideration a predicted future load [28], or combining forecasting models using genetic algorithms [29]. In fact, results depend on the historical data and the observation interval [30], and sometimes require stable workloads to apply long learning [31]. Our solution requires no assumption about the workload acting on the system, its characteristics, or its trends.

The scientific literature adopted different **control architectures** to address autoscaling issues (see [12] and references therein for a recent survey on the state-of-the-art). Beyond threshold-based techniques (e.g., see [26], [11] and reference therein), a number of the proposed techniques exploits control theoretic approaches. Feedback Loop Control is the most commonly used architectural pattern [12], although more advanced patterns have been also proposed, such as Observe-Decide-Act (ODA) [32] or Monitor-Analysis-Plan-Execute (MAPE) [33] which increase the overall complexity of the control architecture since they usually require the design of observers for the correct on-line prediction of the working conditions. Both *Single* and *Multiple Loop Control* solutions have been proposed to address cloud autoscaling. The latter are effective for isolating the logical aspects of autoscaling, but can be difficult to be implemented due to necessity of guaranteeing low coupling for the proper design of each of the control loops [19], [21].

Control approaches usually leverage on mathematical models able to capture the relationship between the allocated resources and the high level metrics. For example *Padala et al.* [8] propose a MIMO adaptive controller that uses a second-order ARMA to model the non-linear and time-varying relationship between the resource allocation and its performance. *Kalyvianaki et al.* [10] designed different SISO and MIMO controllers to determine the CPU allocation of VMs. Solutions proposed by *Lama et al.* [33] and *Ghanbari et al.* [34] use a Model Predictive Control (MPC) approach, which involves optimizing a cost function to expresses the local control objectives and resource constraints. *Padala et al.* [35] propose an adaptive control strategy that exploits a black-box system modeling technique. In all these works, the determination of the system model in a dynamic cloud environment is not trivial. Workload and cloud dynamics make the identification of system models difficult [36] and this limits the effectiveness of the approach in real-world scenarios. Alternative control techniques, such as Proportional (P) Proportional-Integral (PI), Proportional-Integral-Derivative (PID) do not require a mathematical description of the system for designing the controller. Therefore, they have been analyzed to regulate one specific direct or indirect QoS parameter in cloud applications (e.g., application latency or CPU load) [6], [7], [23], [24], [37], [38]. The effectiveness of these controllers is strongly related to the values of the control parameters (such as control gains) whose optimal tuning over extended operating conditions is the main difficulty when applying these techniques in vague and time-varying environments, strongly influenced by the effect of unpredictable disturbances (e.g. varying workload) [39]. In addition, these approaches, do not consider heterogeneous metrics depending also on the state of the network resources associated to the VMs, and mainly scale resources based on computational aspects (e.g., the state of memory and CPU). Other model-free approaches exploit fuzzy logic, since it easily allows the translation from logic statements to a nonlinear mapping and it has been proven to effectively deal with complex and nonlinear systems [40]. Thus it as been also recently proposed for cloud management. For example, both *Anglano et al.* [20] and *Rao et al.* [41] exploit a pure fuzzy logic feedback control for the regulation

of CPU with a dynamic vertical-scaling approach, validating results through private virtualization platforms. However, the former [20] leverages a stochastic approximation algorithm for the on-line estimation of system features (hence performance depends upon the quality of the prediction), while the latter proposes a complex hierarchical structure based on multiple control layers [41]. Horizontal scaling has been instead proposed in *Frey et al.* [42], where the fuzzy controlling scaling architecture again leverages on the on-line forecast of the load. Here achievable performance depends on the quality of the predictions, and have been tested in simulation. The on-line estimate of the workload is also needed in the fuzzy Q-learning control scheme described in *Jamshidi et al.* [25] where a reinforcement learning algorithm have been implemented in a MAPE paradigm. Moreover, since learning process needs time to converge, the performance of the scaling actions produced during initial learning epochs, or during rapid transients originated by abrupt changing, at runtime may be poor. Fuzzy approaches have been also analyzed for addressing different problems, such as the consolidation and migration of VMs investigated in *Monil et al.* [43], where the migration decision depends on a stochastic detection algorithm that again implies the on-line prediction of host utilization. Achievable results are provided in a simulation. *Wang et al.* [21] analyze instead the case of virtualized databases. Here the management of resources is based on a cross-layer optimization algorithm that uses the fuzzy language, not for the control design, but for modelling the relationship between workload and VM resource demand. One more time, results, obtained in a Xen based virtualization environment, depend on the model ability in predicting the demand, since the occurrence of mispredictions affects the quick adaptation to the changes in application workload.

The control approach we propose in this work inherits the simplicity of single-loop PID and the flexibility and robustness of fuzzy logic for the self-adaptation of the control parameters [44]. The strategy extends our solution proposed in [38] that consists in a gain scheduling technique.

3 Problem Statement and System Architecture

In this section, we first describe the problem we aim to solve (Sec. 3.1). Then, we introduce the overall architecture we have designed to address it, together with the description of its constituting blocks (Sec. 3.2).

3.1 Problem statement

Thanks to cloud elasticity, the *cloud customer* is able to decide at runtime the resources she pays the provider for. In this framework, the proposed approach aims at *automatically dimensioning the set of resources allocated to a cloud service*, in order to guarantee the desired performance level to *final users*, in spite of the presence of dynamically fluctuating *workload*. The solution is based on the Infrastructure as a Service (IaaS) paradigm and therefore resources are considered at VM granularity: VMs are activated or terminated on request, thus composing a cluster of dynamically changing size. The goal is automatically activating and deactivating VMs, to achieve the desired service level (SL) and avoid revenue loss to the cloud-customer. Indeed, when the number of active VMs does not properly increase together with the workload, the performance perceived by the final user could dramatically fall down. On the other hand, over-sizing the set of active VMs is source of revenue loss.

TABLE 1
Actors and terms.

Public Cloud Provider	• Makes available cloud resources according to the *pay-as-you-go* paradigm
Cloud Customer	• Configures and manages cloud services by leveraging cloud resources • Is responsible for the service level guaranteed
Final User	• Takes advantage of the cloud services provided by the cloud customer • Requires guaranteed service levels
Resources	• Consist in VMs activated and deactivated at runtime (IaaS model) • Host services in charge of executing tasks on final users' request
System Workload	• Generated by the execution of tasks, started by requests from final users and executed by the VMs
Task	• Needs both computing and network communications among VMs to be accomplished
Service Level (SL)	• Is estimated by monitoring both CPU and network load • Impacts tasks completion time and latency perceived by final users
Control Objective	• Keeping the CPU and network load of the active VMs close to the SLO in order to guarantee expected performance to final users and avoid revenue loss to the cloud customer

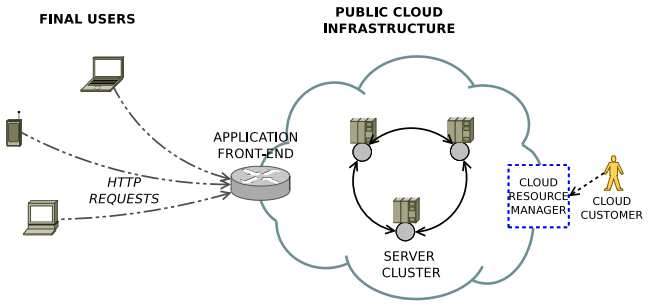

Fig. 1. Reference scenario.

In more details, we consider a generic service, whose interface exposes a number of functionalities to final users through a web interface (see reference scenario in Fig. 1). Final users can submit tasks to the application through a *front-end*, that is in charge of scheduling users' requests and forwarding them to the server cluster. These functionalities require communication among the VMs of the cluster for being accomplished. Note that this service is representative of some typical applications running onto the cloud, such as applications for scientific computing (requiring communication among nodes to distribute shards of a complex task among a set of nodes) [45] or multi-tier applications (that separate roles—e.g., business logic and databases—into multiple layers exchanging data) [46], [47], [48].

According to the reference scenario, whose actors and terms are reported in Tab. 1, the problem we address consists in keeping the performance of the cloud application (i.e., its *service level*, *SL*) close to a desired performance

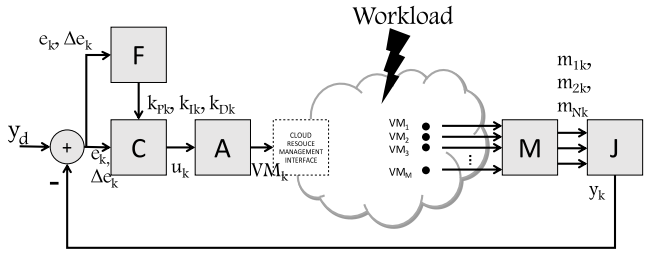

Fig. 2. Fuzzy-PID architecture designed and implemented to solve the resource allocation problem in public clouds. Different blocks can be identified: the *Control Block* (C), the *Fuzzy-Logic Block* (FL), the *Monitoring Block* (M), the *Actuation Block* (A), and the *Fitness Function Block* ($\mathcal{F}$).

level (i.e., the *service level objective*, *SLO*). To this aim, our solution dynamically activates and deactivates at runtime the resources leased from the cloud provider.

3.2 System architecture

In this section we describe the overall architecture of the system we have designed and implemented. As previously discussed, the aim is to regulate the SL of the application at the SLO. Note that at each cycle the SL is evaluated merging the relevant QoS parameter values through a *Fitness Function* $\mathcal{F}$ (as detailed in Sec. 3.2.2) providing a Key Performance Indicator (KPI). Desired values for SL have to be guaranteed in spite of unpredictable and time-varying disturbances (i.e., the workload).

The overall architecture is shown in Fig. 2. Here, starting from the metrics observations gathered and processed by the *Monitoring Block* (M), the actual QoS index that represents the SL of the cloud application at each cycle k, say y_k, is evaluated through the *Fitness Block* ($\mathcal{F}$). y_k is then compared to the target, say the desired QoS index value y_d .The actual performance error, say $e_k = (y_d - y_k)$ and its rate of variation (Δe_k) act as input signals of the Fuzzy PID *Control Block* (C). Control gains (k_{P_k}, k_{I_k}, and k_{D_k}) are automatically and dynamically self-adapted through Fuzzy Logic by the *Fuzzy-Logic Block* (FL), in order to dynamically counteract the effects of uncertainties and workload variations. The *Actuation Block* (A) connected to the public-cloud resource management interface, implements a scaling algorithm proportional to the amplitude of the control signal to start or terminate a different number of VMs at each cycle.

The remainder of the section is organized as follows: Sec. 3.2.1 presents the monitoring system (M block); Sec. 3.2.2 shows the Fitness Function ($\mathcal{F}$ block). Sec. 3.2.3 introduces the PID controller we designed for the control block (C block); Sec. 3.2.4 presents the actuation policy (A block).

3.2.1 Monitoring

In order to fulfill the needs of a generic application, the proposed approach takes advantage of heterogeneous metrics to capture the different aspects of interest, namely *CPU load* (ranging from 0% to 100%) and *network usage*. These are representative aspects of the state of the system: on the one hand, CPU load well captures the impact of computation on the performance as perceived by the end user [23]; on the other hand, network usage gives hints

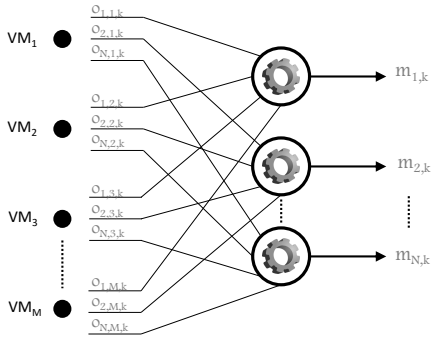

Fig. 3. Monitoring Block.

about the state of the network interconnections among VMs in the cluster. Note that recent experimental works on public clouds, disclose the importance of this last aspect [14], [16], [17] to achieve complete application scalability in shared datacenters. It is worth noting that we only consider as suitable metrics those available to the general customer because of the point of view adopted. Therefore, the metrics we consider are all available to the cloud customer at server side, differently than other solutions subjected to less strict constraints by design. For instance, we purposely do not consider either performance metrics that capture the QoS at the user side [41], or metrics related to layers beyond the scope of the general customer (e.g., the physical machine layer). Moreover, thanks to the fitness function we describe below, the proposed architecture is suitable to be extended with different monitoring blocks, such to consider all the heterogeneous metrics potentially available to cloud customers at server side.

As shown in Fig. 2, the Monitoring Block (M) is interfaced to multiple VMs and produces heterogeneous metrics. According to Fig. 3, for each of the M VMs that are active at the cycle k, the Monitoring Block extracts one sample for each of the N heterogeneous metrics, say $o_{i,j,k}$ where $i = 1, \ldots, N$, $j = 1, \ldots, M$. These M samples flow into N separate processing modules, that fuse them in a single metric observation, say $m_{i,k}$. Under the assumption that the samples concurring to compute a given metric observation are equal with respect to the active VMs, we have:

$$m_{i,k} = \sum_{j}^{M} \frac{o_{i,j,k}}{M} \tag{1}$$

Note that the value of M may change from cycle to cycle.

Given the general structure in Fig. 3, in this work we consider three different metrics ($i = 1, 2, 3$) namely the CPU load (`CPU`) for the computational capability and the amount traffic injected into or received (`NETOUT` and `NETIN`, respectively) for the communication network. The former is expressed as a percentage, while the others as the volume of traffic per cycle.

Indeed, this approach is in line with the granularity of the resources considered and with the idea that each element of the cluster equally contributes to the amount of virtual resources allocated to the application (i.e. computation and communication resources).

In public-cloud environments, different approaches can be implemented to gather observations from different VMs [49]. The different choices available reflect different levels of flexibility and scalability. From the one hand, the cloud customer can adopt her own monitoring module [50]. This approach can be applied to any public cloud, and completely relies on the customer, who is in charge of designing, developing, and deploying monitoring probes able to gather observations on each active VM. The main advantage of this approach consists in the higher flexibility (i.e. the customer can decide the metrics to monitor, and the granularity of the measure based on the application needs). On the other hand, the customer can also rely on monitoring modules directly supplied by the provider, when available. In this case the customer, being relieved of the design, implementation and deployment burden, is restricted in some implementation choices such as the available metrics or the granularity of the measurements (that directly impacts the duration of the cycles adopted by the overall architecture). In Sec. 5.1 we will detail how information gathering process can be implemented.

3.2.2 Fitness Function

Based on the metric observations, the next step is to construct an index that captures the overall state of the cloud system at each cycle k. To this aim here we design a fitness function block. The definition of a QoS index from heterogeneous metrics is a well known problem in the recent literature [51], [52], [53]. Here, with the idea in mind of evaluating how close the current state of the cloud system is with respect to an ideal reference behavior, we adopt the general approach proposed in [51] and propose the following fitness function:

$$\mathcal{F}_k = \sum_i^N \alpha_i \frac{m_{i,k}}{R_i} \tag{2}$$

where $m_{i,k}$ and R_i are the metric observation at the sampling time k and the reference value, respectively; $\alpha_i \in [0,\ 1]$ are positive weights so that:

$$\sum_i^N \alpha_i = 1 \tag{3}$$

By construction, when the system is working at the desired value for each metric (i.e. when $m_{i,k} = R_i, \forall i, \forall k$) we obtain $\mathcal{F}_k = \mathcal{F}_k^* = 1$. Otherwise, the output of the function reaches values larger than 1 when each measure $m_{i,k}$ is larger than the respective reference R_i, or values smaller than 1 when each measure $m_{i,k}$ is smaller than the respective reference R_i.

Given the overall constraints in (3) the value to be assigned to each α_i is crucial for the evaluation process, and the priority given to each single metric that contributes to the QoS index usually depends on the specific application, as shown in the very recent literature on key performance indices [51], [53].

In our proposal `CPU`, `NETIN`, and `NETOUT` are merged through the fitness function, in accordance to the goals of our work. It is worth noting that thanks to the nature of the fitness function we adopt the proposed approach is suitable to consider any of the the metric available to the general customer in spite of the specific metric selection we make here.

3.2.3 Control

Due to their well-known simplicity and flexibility, single loop architectures are often adopted for auto-scaling [12]. However, in their classical formulation, they only allow to cope with one resource at time, such as CPU [20], [38]. To overcome this limit, our approach embeds the fitness function described in Sec. 3.2.2 within the control loop (Block $\mathcal{F}$ in Fig.2). In so doing the autoscaling decision depends on a set of heterogeneous metrics merged together and allows to easily handle multi-objectivity criteria and to balance different requirements without increasing the complexity of the controller structure (e.g. via multiple loops).

The control goal is to adjust the allocated VMs without human intervention, so as to drive performance indices towards their targets and keep QoS violations to zero. Indeed, the monitoring component checks the platform to display changes and violations of the QoS requirements by evaluating the fitness function at each cycle k, say $y_k = \mathcal{F}_k$. Then, the control algorithm uses a multi-input approach to decide the number of VMs to be allocated (horizontal scaling) at cycle k depending on the error e_k (i.e. the difference between the desired KPI value y_d and its actual value y_k) and its rate of variation $\Delta e_k = e_k - e_{k-1}$ which provides additional information and enhances the generalized damping of the control system [54].

When dealing with regulation processes in vague and unmodelled scenario, PID controllers are usually the first choice: they have a very simple structure, are easy to be implemented on-line, and show good results in terms of response time and precision, if control gains are well tuned [55]. However, controller tuning is difficult for uncertain nonlinear systems (such as public clouds). Moreover, fixed gains cannot cope with highly-variable environments and this limits the effectiveness of the approach [39]. For these reasons, in this work we implement a discrete PID controller (Block C in Fig.2) that leverages on the fuzzy control paradigm (Block FL in Fig.2). The idea is to combine the PID structure with a fuzzy-logic control scheme so that the controller preserves its simple structure, but it is also able to self-adjust its control parameters (or gains) depending on the actual conditions [56]. Thanks to this approach, the controller is cheaper to develop and implement, covers a wider range of operating conditions, and is more readily customizable in natural language terms [44].

Note that the number and the mix of the users requests (i.e. the workload) are here considered as unknown disturbances that cannot be predicted [36]. The amplitude of the allocation adjustment u_k (that depends on the value of the control gains and have to avoid both under- and over-provisioning of resources) is self-adaptive and it is determined by the fuzzy logic.

In that the on-line performance mainly depends on the proper selection of control gains, a crucial point for self-adaptive resource management is the identification of admissible gains regions (within which the gains can vary) that can provide optimal behavior over extended cloud operating regimes. Indeed, the desired dynamic response should always have minimum settling time with a small or no overshoot and undershoot when subjected to a small

step perturbations, while the control action, and its rate of variation, should always be accurately calibrated to guarantee the closed-loop stability even in the presence of high-frequency or high-amplitude disturbances. To this aim a genetic algorithm (GA) is applied and the design problem is formulated as an optimization problem, where GA is employed to optimize the gains regions of the Fuzzy-PID controller [57]. In so doing, not only the tuning procedure is straightforward, avoiding the ever cumbersome and subjectivity of trial-and-error-based heuristics, but also improves the control sensitivity and closed-loop performance. Details on control design and its analytic derivation are illustrated in Sec. 4.

3.2.4 Actuation

In our horizontal scaling scenario the number of VMs to be added or removed at cycle k, say VM_k, is proportional to the amplitude of the control signal u_k according to the following dead-zone with saturation (Block A in Fig.2):

$$VM_k = \begin{cases} \#VM_{\max} & \text{if} \quad \bar{u} \leq u_k \\ \alpha u_k - \beta & \text{if} \quad \epsilon \leq u_k < \bar{u} \\ 0 & \text{if} \quad -\epsilon < u_k < \epsilon \\ \alpha u_k + \beta & \text{if} \quad -\underline{u} < u_k \leq -\epsilon \\ -\#VM_{\max} & \text{if} \quad u_k \leq -\underline{u} \end{cases} \tag{4}$$

where α and β are design parameters impacting the aggressiveness of the actuation: their values are flexible and can be opportunely set to choose how fast the controller adds and removes VMs. Moreover, since small control signal variations can cause the system to oscillate around a target [37] (e.g. due to the presence of some monitoring uncertainties when working at steady state), to overcome this critical problem our scaling mechanism incorporates a symmetric dead-zone ϵ, according to good practice in the nonlinear actuators literature [58]. Other works have considered the oscillation problem, but instead they usually adopt a static thresholding mechanism for their elasticity management policy (see for example [59] and references therein). Note that classical strategies that employ a constant change in the actuator value (e.g. provisioning or terminating only one VM at time [23]) can be ineffective in real scenarios, since it may be too slow in scaling out in the case of sudden peak loads, or it may result in unwanted longer provisioning periods during scaling down.

When working with variable actuation policies, the controlled system may reveal overshoots or instability depending from the workload conditions. For this reason, when implemented, the physical actuator in (4) is subject to saturation of its maximum and minimum limits (maximum number of VMs to be added/removed at each cycle) [58]. These values can be set heuristically according to the systems observations. Here we consider a possible variation of ± 4 VMs per each cycle, on the base of our experimental experience and previous experimental evaluations (e.g. see [37] and references therein). Non-linear actuation characteristic has been discretized with the classical sample-and-hold method for its implementation in the platform.

It is worth noting that the designed actuation block manages homogeneous VMs (i.e., having same characteristics, such as family and the size, and expected performance). This design choice is in line with recent literature [23] and has the notable advantage of allowing to keep the complexity of the architecture bounded, as in this way components such as the control block, the actuation block, or the load balancer do not require additional knowledge (e.g. performance models) to suitably deal with heterogeneous VMs. Different approaches would require changes to one or more blocks of the designed architecture and are left as future work.

4 Fuzzy-PID Control design

In this section we provide details about the control block we have designed. In this work, we solve the regulation problem by complementing the PID controller with a fuzzy rule-based scheme, in order to automatically adapt the control action to the changing dynamics. The approach enables an easy connection between fuzzy parameters and operation of the PID controller. In more particular, here we describe the design principle of the PID controller together with the fuzzy reasoner implementing gain scheduling (Sec. 4.1) and the optimization of the related working regions (Sec. 4.2).

The control algorithm (see block C in Fig.2) is based on a discrete PID structure with variable gains, that can be mathematically formalized as:

$$u_k = k_{P_k} e_k + k_{I_k} \Delta t \sum_{q=1}^{k} e_q + \frac{k_{D_k}}{\Delta t} \Delta e_k, \tag{5}$$

where u_k is the control action at cycle k; $e_k = y_d - y_k$ is the closed-loop error (i.e. the difference of the desired output y_d and the measured output y_k); $\Delta e_k = e_k - e_{k-1}$ is the error change (or better its first difference), Δt is the sampling interval set as one cycle and k_{P_k}, k_{I_k} and k_{D_k} are the control gains.

4.1 Fuzzy Gain Scheduling

The fuzzy rules and reasoning block (see the block FL in Fig. 2) are used to modulate the control effort by self-adjusting the control gains in (5) on the base of the values assumed by the two inputs e_k and Δe_k; i.e. $k_{P_k} = k_P(e_k, \Delta e_k)$; $k_{I_k} = k_I(e_k, \Delta e_k)$; $k_{D_k} = k_D(e_k, \Delta e_k)$. To design the scheduling algorithm, it is assumed that control gains are in prescribed ranges, i.e., $k_{P_k} \in [k_P^{\min}, k_P^{\max}]$; $k_{I_k} \in [k_I^{\min}, k_I^{\max}]$; $k_{D_k} \in [k_D^{\min}, k_D^{\max}]$.

For convenience, the current values of PID gains at cycle k are normalized into the range between zero and one, i.e.

$$\begin{aligned} k'_{P_k} &= \tfrac{k_{P_k} - k_P^{\min}}{k_P^{\max} - k_P^{\min}}, \\ k'_{I_k} &= \tfrac{k_{I_k} - k_I^{\min}}{k_I^{\max} - k_I^{\min}}, \\ k'_{D_k} &= \tfrac{k_{D_k} - k_D^{\min}}{k_D^{\max} - k_D^{\min}}, \end{aligned} \tag{6}$$

while the normalized error and its normalized increment are given by

$$\begin{aligned} \widetilde{e}_k &= K_e e_k, \\ \Delta \widetilde{e}_k &= K_{\Delta e} \Delta e_k, \end{aligned} \tag{7}$$

being K_e and $K_{\Delta e}$ some scaling factors to be opportunely chosen. The definition of the parameters regions and, hence, the computation of the minimum and maximum value of each PID gain is carried out by off-line solving a constrained optimization problem as described in Sec. 4.2.

The fuzzy logic gain scheduler is designed according to the classical architecture of fuzzy logic systems [40] and its structure is based on four main components, namely the knowledge base, the fuzzification interface, the inference engine, also known as decision making logic, and the defuzzification interface [40]. The knowledge base system contains all the information required for the fuzzy system, namely the fuzzy control rule base and the data base. The inference engine performs inference procedures upon the fuzzy control rules, while the fuzzification interface defines a mapping from a real-value space to a fuzzy space and the defuzzification interface implements a mapping from a fuzzy space to a real-valued space. In what follows we provide details on each of the components for our two-inputs fuzzy PID configuration.

The knowledge base is combined by three separate rule bases, with simple rules for each different gain. Given the normalized variables (eqs. 6–7), the rules base structure takes the following form:

$$\begin{cases} \texttt{If } \widetilde{e}_k \texttt{ is } A_i \texttt{ and } \Delta\widetilde{e}_k \texttt{ is } B_j \\ \texttt{Then } k_{P_k}^{'} \texttt{ is } C_m,\ k_{D_k}^{'} \texttt{ is } D_\lambda \texttt{ and } k_{I_k}^{'} \texttt{ is } E_\gamma \end{cases} \tag{8}$$

where $i = 1, \ldots, I$ and $j = 1, \ldots, J$ represent the fuzzy states of the antecendents (being I and J the number of linguistic values associated to the antecendents); $m = 1, \ldots M$; $\lambda = 1, \ldots, \Lambda$; and $\gamma = 1, \ldots, \Gamma$ are the fuzzy states associated to the consequent (being M, Λ and Γ the number of linguistic values associated with the control gains); and A, B, C, D, E are the fuzzy sets.

By opportunely setting K_e and $K_{\Delta e}$ in (7) (see Table 3a) , both the universes of disclosure of $\widetilde{e}(k)$ and $\Delta\widetilde{e}(k)$ have been scaled to $[-1, +1]$, while the corresponding membership functions are shown in Fig.4. Here "NB" stands for "Negative Big", "NM" is "Negative Medium", "NS" is "Negative Small", "Zero" is "ZO", "PS" represents "Positive Small", and "PM" and "PB" stand for "Positive Medium" and "Positive Big", respectively. Note that to cope with the real control problem, we adopt non-equal span membership functions since for highly nonlinear processes a fuzzy controller with equal-span triangular membership function is not adequate to achieve good control results [60].

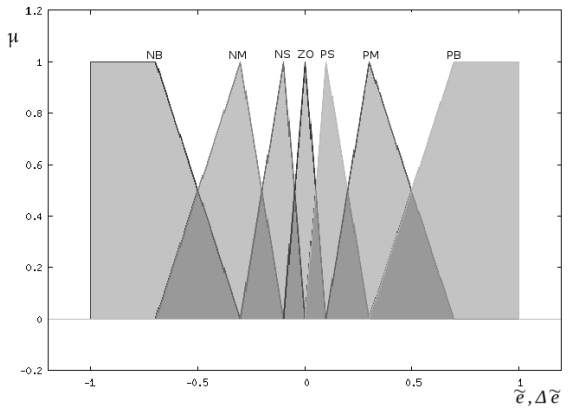

Fig. 4. Membership function for $\widetilde{e}(k)$ and $\Delta\widetilde{e}(k)$.

With respect to the outputs of the gain scheduling fuzzy modules, C_m, D_λ can be either "Big" or "Small" ("B" and "S", respectively), while E_γ can be instead "Very Big", "Big", "Small" and "Very Small" (i.e. "VB", "B", "S" , "VS"), and they are modelled as singleton membership functions [40].

The rule base of the Fuzzy-PID in (8) is depicted in Tab. 3a for $k_{P_k}^{'}$, in Tab. 3b for $k_{D_k}^{'}$, and in Tab. 3c for $k_{I_k}^{'}$. These tables show that the fuzzy reasoning block incorporate 49 standard rules. The selection of fuzzy rules is one of the important things to be considered to achieve smoother response and less oscillation at the transient state. It also impacts overshoot and undershoot of the disturbance, so as to improve the cloud system stability. To this aim the rule base is here similar to those in [61], [62], for which satisfactory results have been demonstrated. Since these rules play very important role in the achievable performance, they have been also investigated comprehensively by experimentally studying step responses of the cloud system under different dynamic conditions.

TABLE 2
Fuzzy tuning rules.

		$\Delta\widetilde{e}_k$						
		NB	NM	NS	ZO	PS	PM	PB
$\widetilde{e}_k$	NB	B	B	B	B	B	B	B
	NM	S	B	B	B	B	B	S
	NS	S	S	B	B	B	S	S
	ZO	S	S	S	B	S	S	S
	PS	S	S	B	B	B	S	S
	PM	S	B	B	B	B	B	S
	PB	B	B	B	B	B	B	B

(a) Fuzzy Tuning Rules for $k_P^{'}$.

		$\Delta\widetilde{e}_k$						
		NB	NM	NS	ZO	PS	PM	PB
$\widetilde{e}_k$	NB	S	S	S	S	S	S	S
	NM	B	B	S	S	S	B	B
	NS	B	B	B	S	B	B	B
	ZO	B	B	B	B	B	B	B
	PS	B	B	B	S	B	B	B
	PM	B	B	S	S	S	B	B
	PB	S	S	S	S	S	S	S

(b) Fuzzy Tuning Rules for $k_D^{'}$.

		$\Delta\widetilde{e}_k$						
		NB	NM	NS	ZO	PS	PM	PB
$\widetilde{e}_k$	NB	VS	VS	VS	VS	VS	VS	VS
	NM	S	S	VS	VS	VS	S	S
	NS	B	S	S	VS	S	S	B
	ZO	VB	B	S	VS	VS	B	VB
	PS	B	S	S	VS	S	S	B
	PM	S	S	VS	VS	VS	S	S
	PB	VS	VS	VS	VS	VS	VS	VS

(c) Fuzzy Tuning Rules for $k_I^{'}$.

Concerning the fuzzy implication operator implemented by the inference engine, the choice fell out upon the zero-order Sugeno-type approach modeled by the operator `min(;)`, since this method has been shown to be computationally effective and to work well with optimization and adaptive techniques, which makes it very attractive for the implementation control approaches in a real world environments [40]. Accordingly, the crisp normalized output is

then generated through a weighted average of rule outputs. Finally, the crisp control gains values to be provided at each time instant k are derived from the normalized ones (6).

4.2 Optimization of the gains regions

The values of the PID gains, selected via fuzzy logic, can vary within prescribed ranges identified by their minimum and maximum admissible value, i.e. $k_P^{\max}$, $k_P^{\min}$, $k_I^{\max}$, $k_I^{\min}$, $k_D^{\max}$, $k_D^{\min}$. Usually, admissible ranges are set heuristically according to the experience of designers (e.g., it is well known that higher value of k_{P_k} results in faster system response, but larger over-shoot, while higher value of k_{D_k} results in slower system response, but smaller over-shoot). Conventional techniques to determine these gains are: trial and error method, Zeigler-Nichols method [55], but these conventional methods are time consuming (e.g. due to the heuristic determination of all the possible set of three) and may not yield optimum controller gain set, while a wrong selection may also induce a loss of stability in some critical cases.

Here an optimization technique is employed to get the optimum values of controller gains in order to extract better dynamic performance from the Fuzzy-PID controlled cloud system. The cost function to be minimized over a finite control horizon expresses the closed-loop performance as follows:

$$\min_K J(e,K) = \min_K \sum_{k=1}^{N_p} [\psi_1 {e_k}^2 + \psi_2 {u_k}^2], \tag{9}$$

where $K = [k_P, k_I, k_D]^T$ and $N_p \in N^+$ is the prediction horizon. Note that the weights ψ_1 and ψ_2 have been introduced in the objective function with a provision of balancing the impact of the error and the control effort. We choose equal weights for the two objectives to be met by the controller, since the minimization of the error index is as equally important as the control effort is. The quadratic cost is hence minimized to find out the optimal ranges of control parameters which simultaneously reduces the regulation error e_k and the required control action u_k. In other words, the minimization of the squared control signal reduces the cost involved into the regulation process, trying to activate the less of VMs as possible while meeting performance requirements.

The offline optimization procedure exploits a non-linear single-input single-output dynamic system model described as

$$y(k) = g(\varphi(k)) \tag{10}$$

where φ is a regression vector consisting on past inputs and past outputs as

$$\varphi(k) = [y(k-1), \ldots, y(k-N_y), u(k-1), \ldots, u(k-N_u)],$$

being N_y and N_u the number of past output and input samples, respectively. Note that an on-line solution of the optimization problem in (9) is computationally time consuming, and increases the complexity of the controller during its practical use.

Since the large variability exhibited by system dynamics in the different operating points, the range of cloud operations has been divided into several different zones according to different workloads and time slots. The output and input samples are hence related to systems observations during step responses in different operating conditions for various initial conditions (30 in total). An hybrid genetic algorithm (HGA) (previously designed and used from authors in different applicative fields, e.g. see [63] and reference therein) is used to solve the optimization problem in the overall parameter space. The HGA is combination of GA and nonlinear least-square method (LS). The main idea is to merge the global-search properties of GAs with the fast local convergence of LS methods. The optimization procedure is initialized by selecting the gains with the classical closed-loop Ziegler and Nichols method and provides the optimization of the gains regions until a minimum of the chosen criterion is achieved within a prefixed tolerance or the maximum number of iterations is reached. Results of the optimization procedure are: $k_{P_k} \in [90, 120]$; $k_{I_k} \in [0.001, 0.05]$; $k_{D_k} \in [0.001, 0.002]$.

5 Evaluation

In this section we first describe the experimental setup, then we present the results achieved thanks to the proposed Fuzzy-PID approach.

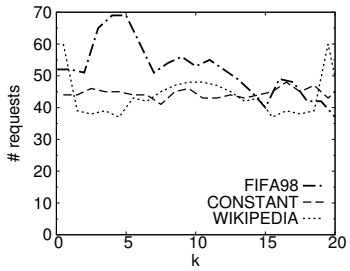

Fig. 5. Experimental setup: workloads.

5.1 Experimental set-up

In order to evaluate our proposal in a public-cloud environment we set up a testbed composed of three main elements: (i) the *cloud service*, deployed and running on a public-cloud infrastructure; (ii) the *master node*, hosting all the blocks of the architecture proposed (see Fig. 2) and managing cloud resources through their interaction; (iii) the *final-user emulation node*, in charge of issuing requests to the cloud application and imposing the workload to the system. In the following we present all the implementation details for each of the elements above. For the sake of repeatability and to foster further research, we publicly release all the code behind the evaluation of our solution[1].

5.1.1 Cloud application

Our solution has been extensively tested on a *Amazon EC2* IaaS environment. Although the approach we propose does not depend on the specific provider, Amazon EC2 represents a valid test bench for the proposed solution as Amazon is claimed to be one of the leading providers [64]. The cloud application is deployed onto a group of VMs (i.e., the cluster) programmed to dynamically change its size at each cycle k, as demanded by the master node. The cluster is composed of a set of same-type VMs, connected to a load-balancer instructed to equally distribute the incoming requests across them. Each VM is in charge of hosting a

1. http://traffic.comics.unina.it/cloud

web server, in order to serve the HTTP requests generated by final users and forwarded by the load balancer. When a VM receives a request from the load balancer acts as *root node* for the request, starts a number of CPU-consuming and network-intensive tasks, sharing among all the VMs in the cluster the burden generated by the request. Each VM communicates with other peers in the cluster: based on the cardinality of the cluster, a different amount of traffic is generated towards and from each VM. Each VM at the end of the process, returns the control to the root node, that generates the reply directed to the final user.

For all our experimentations, we adopted general purpose micro VMs (`t2.micro`) representing a feasible choice for our long experimental sessions thanks to their limited cost. The selection of this particular kind of VMs does not represent a limitation, since the proposed control approach does not depend on the type of the VMs employed and no assumption has been made on provisioning dynamics. In order to distribute the incoming HTTP requests to the VMs of the cluster in a balanced fashion, we properly configured the *AWS Elastic Load Balancing* service made available by the provider itself. Finally, the network-intensive tasks are executed by synthetically generating real network traffic among the VMs of the cluster through the adoption of D-ITG traffic generator [65].

5.1.2 Master node

For what concerns the master node, two blocks have to be properly configured and tuned.

Monitoring Block. For the experimental evaluation, the samples to construct metric observations have been gathered through the monitoring solution supplied by the cloud provider itself i.e. *Amazon Cloudwatch* [66], since it is suitable to obtain all the metrics of interest for the proposed approach. We also developed a provider-independent solution, in order to foster the replication of the proposed approach to other public cloud providers.

Fitness Function Block. With respect to the weights of the fitness function (see eq. 2) where not explicitly stated otherwise, we balance the computational capability and the network communication aspects, hence we assigned an overall weight of 0.5 to both the `CPU` and network-related metrics `NETIN` and `NETOUT`, (subdivided as 0.25 each). This is an exemplar priority choice, made for a generic application.

5.1.3 Final user emulation

To issue web requests that impose a workload on the system, a geographically separated node was configured. The realistic workload has been generated by exploiting *Httpmon*, an HTTP request generator purposely designed for executing experiments related to computing capacity shortage avoidance in cloud computing [67]. The tool allows to emulate web users by generating request patterns in which the time between two consecutive requests is exponentially distributed. We instructed Httpmon to use the open model, i.e. to issue requests without depending on the system's response.

To evaluate our approach, we consider three different workloads: *(i)* a constant workload (CONSTANT) and two realistic benchmarks: *(ii)* the *WorldCup98 web-server workload* (FIFA98) [68]—note that it is a meaningful benchmark extensively used in the cloud scientific literature ([11], [69]). and *(iii)* the *WikiBench workload* (WIKIPEDIA)—related to the application used to host `wikipedia.org` which allows one to stress-test systems designed to host Web applications and cloud platforms. An example of the HTTP requests issued with the three workloads is depicted in Fig. 5. Note that because of the pattern followed by the adopted tool, the CONSTANT workload may not result in an amount of requests exactly constant over time. For each experiment, each workload has a duration of 120 minutes. The overall experimental activity (control tuning and extensive validation) amounted to 100 hours.

5.2 Experimental Results

Here the effectiveness of the proposed architecture is investigated. Through representative experimental examples (i) we evaluate the proposed Fuzzy-PID architecture against three different workloads (i.e. CONSTANT, FIFA98, and WIKIPEDIA workloads) also considering the impact on each metric individually; (ii) we disclose the stability and robustness of the approach with respect to different choices of the weights α_i that balance the different variables within the fitness function; (iii) we evaluate the robustness in the presence of VM failures; (iv) we compare the proposed solution against recently proposed approaches;

5.2.1 Performance in the presence of different workloads

Here we evaluate the ability of the proposed approach to counteract changes in workload variations, i.e. to increase or decrease the number of VMs according to workload intensity, so to regulate the current SL measured as $y_k = \mathcal{F}(\cdot)$ at its desired SLO ($y_d = \mathcal{F}^* = 1$). Note that to this aim the strategy does not exploit any online measure or prediction of the workload that is assumed to be unknown in the regulation process.

The analysis is therefore carried out for the three workloads introduced in Sec. 5.1.3. Results depicted in Fig. 6

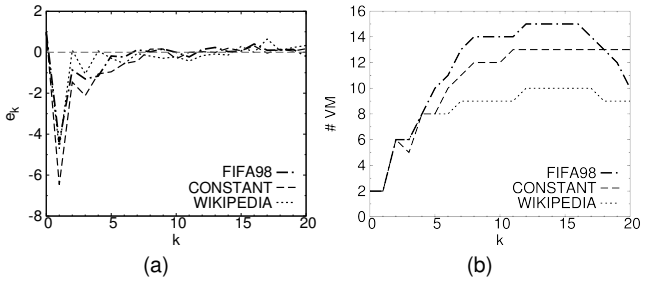

Fig. 6. Performance in the presence of the three different workloads in Fig. 5 (start-up transient and regime). (a): time history of the regulation error with the respect to the SLO $e_k = (y_d - y_k)$; (b): time history of the active VMs ($\#VM(k)$).

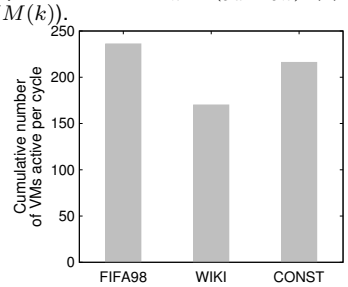

Fig. 8. Control cost associated to each workload, calculated as the cumulative number of VMs active per cycle. Best performance in terms of cost is achieved with the WIKIPEDIA workload.

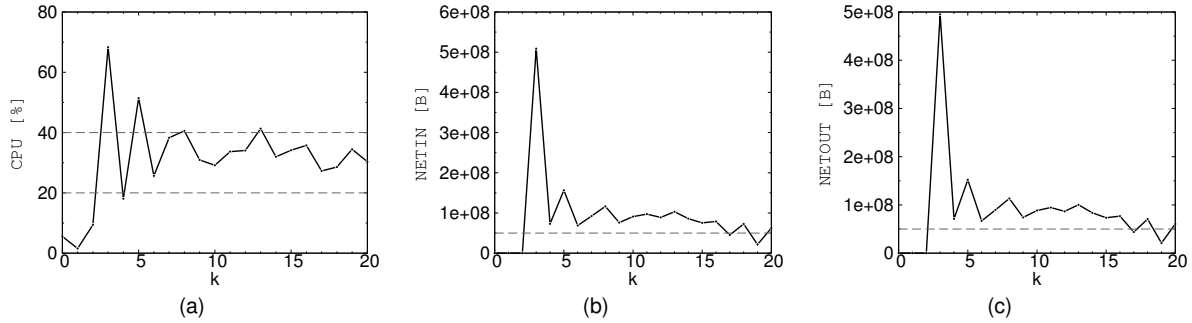

Fig. 7. Performance with respect to each metric considered individually (WIKIPEDIA workload). (a): time history of CPU; (b): time history of NETIN; (c): time history of NETOUT.

show similar performance in all cases. Specifically the systems guarantees the required SLO despite the different workload features in terms of intensity (i.e. number of requests per second) and frequency (i.e. rate of variation). As expected, overshooting appears only at switching-on transient (during the first 9 cycles) but it is negligible at steady state when controller gains are adapted, in spite of still occurring workload variations. Nevertheless the average error is always lower than 30%. Note that, because of the pay-as-you-go model, the number of VMs to be allocated at the time 0, is the solution of a trade-off between maximum acceptable error and cost to be paid, that is beyond the scope of the paper. Better switching-on performance can be easily achieved relaxing the management policies that here consider only two VMs to be allocated at start-up, implementing a conservative choice. This initial condition reflects the communication needs of the application, that at least relies on two VMs to run.

The cost associated to the control action depends on the workload and its variability (see Fig. 8), although the performance is similar in all cases, hence confirming the robustness of the approach. This cost can be easily computed by evaluating the area under the control signal $\#VMs$ reported in Fig. 6b. The overall control cost depends on the workload. The best performance in terms of cost is achieved with the WIKIPEDIA workload, as it is associated with the lowest cumulative number of requests, although it is characterized by a higher variability with respect to CONSTANT workload.

5.2.2 Analysis with respect to single metrics

Although the aim of the proposed approach is to guarantee a prefixed SL evaluated by a sole index computed merging heterogeneous metrics, in our experimental analysis we also evaluate its ability in appropriately regulating each of the metrics of interest individually. Results in Fig. 7 refer as a representative example to the WIKIPEDIA workload case.

Specifically, the Fuzzy-PID approach guarantees that CPU converges to the target CPU (30%), with an average CPU measurement that never exceeds the 28% of the target value, as depicted in Fig. 7a. A similar behavior is obtained for both NETIN and NETOUT, as reported in Fig. 7b and Fig. 7c (where 50 MB is the target value chosen for both). Similar results obtained for the other workloads have been omitted for the sake of brevity.

TABLE 3
Different choices of the fitness function weights considered. 1128 While set 1 and 2 balance computational and network aspects, set 3 only consider CPU.

Set	α_{CPU}	α_{NETIN}	α_{NETOUT}
1	0.5	0.25	0.25
2	0.7	0.15	0.15
3	1	0	0

5.2.3 Sensitivity analysis with respect to fitness function weights

To further analyze the flexibility of the approach we also performed a sensitivity analysis with respect to fitness function weights. As exemplar cases, here we report results for two alternative weight sets (see set 2 and set 3 in Tab. 3) for FIFA98 workload.

Results depicted in Fig. 9a show how the regulation error is driven and kept to zero, independently from the specific choice of the weights set. Also in this case, the average percentage error never exceeds the 30%. Conversely, the history of the VM activation is impacted by the choice of the weights (see Fig.9b). Indeed, the number of active VMs is greater when we balance the CPU and network capability aspects (set 1). Less control effort is instead necessary for the application considered when a grater priority is given to the CPU load (set 2). The lower values of active VMs at each cycle is obtained for set 3, i.e., when only considering the CPU capability as metric of interest. Further work is needed to investigate the trade-off between costs and performance raising from the choice of metrics and weights, and its optimization.

Although the overall performance is always guaranteed, the regulation of each single metric to its desired value is impacted by the choice of the weights. For example, results

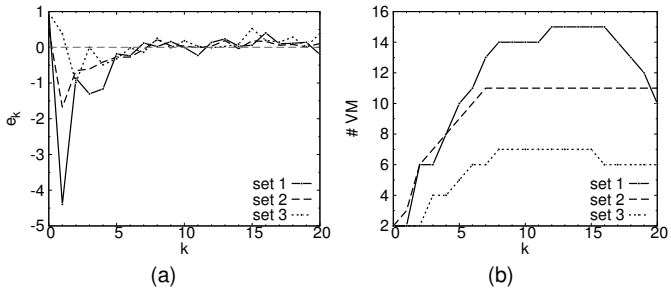

Fig. 9. Sensitivity analysis for alternative fitness-function weights sets as in Tab. 3 (FIFA98 workload). (a): time history of the error with the respect to the SLO $e_k = (yd - yk)$; (b): time history of the active VMs ($\#VM(k)$.

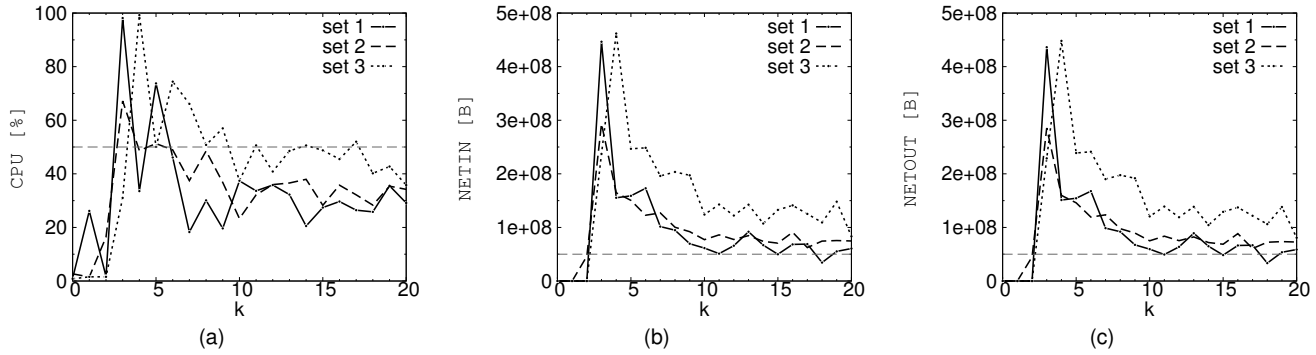

Fig. 10. Sensitivity analysis for alternative fitness-function weights sets as in Tab. 3 for each metric considered individually (FIFA98 workload). (a): CPU time history; (b): NETIN time history; (c): NETOUT time history.

in Fig. 10 disclose that better performance is obtained when only one single metric is taken into consideration (set 3). Note that to highlight the flexibility of the approach with respect to the specific choice of the reference value, the CPU requirement for this experiment has been set to 50%, while the NETIN and NETOUT SLO is kept unchanged.

5.2.4 Robustness in the presence of VM failures

To analyze the robustness with respect to failures, we consider the Fuzzy-PID architecture under the action of the CONSTANT workload (see Fig. 5) to better capture the effects of sudden and unwanted VM termination.

The system is at its steady-state equilibrium point $y_k = y_d = 1$ when we cause a hard failure to happen: specifically more thn $1/3$ of VMs that are running crash at cycle $k = 11$ (4 out of the 14 active VMs, see Fig. 11b).

Due to this critical event, the regulation error increases (see Fig. 11a) and, accordingly, the control action varies, online adapts its gains, and counteracts the effect of the failures (see Fig. 11b). In so doing, the error is then again driven to zero at cycle $k = 14$, as shown in Fig.11a.

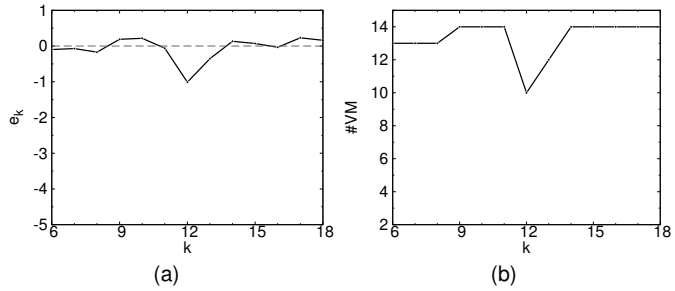

Fig. 11. Robustness in the presence of VM failures. (a): time history of the error with the respect to the SLO $e_k = (y_d - y_k)$; (b): time history of the active VMs ($\#VM(k)$).

5.2.5 Comparison against PID and Gain Scheduling

Finally, in this section we evaluate the proposed Fuzzy-PID approach comparing it to a simple PID control strategy [23] and a plain gain-scheduling algorithm (GS) [38].

It is worth noting here that the architecture presented in this paper solves a more wide problem introducing heterogeneous metrics, while previous attempts in the state of the art (as already detailed in Sec. 2) usually only considered CPU-related aspects. So, in order to perform a fair comparison, we choose to downgrade our architecture by selecting the trivial weight set (i.e., the set 3).

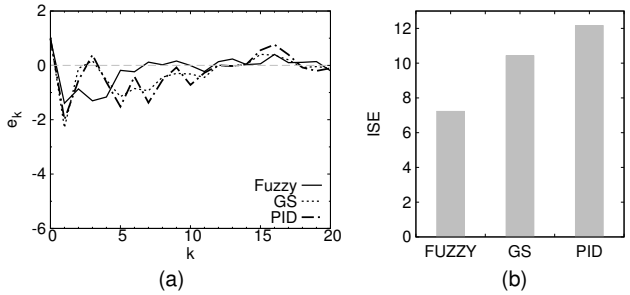

Fig. 12. Comparison among Fuzzy-PID, PID, Gain Scheduling controllers (FIFA98 workload). (a) Time history of the error ($e_k = (y_d - y_k)$) between the actual SL (measured in terms of CPU) and its SLO. (b) Integral of Squared Errors—ISE.

Results in Fig. 12a show that the Fuzzy-PID controller provides better performance than previously proposed approaches, due to its greater ability in self-adaptation of the gains with respect to time-varying conditions of the public-cloud system. In order to better evaluate how the three approaches meet strict SLO ($e_k = 0$), here we exploit the Integral of Squared Errors (ISE) [58]. Results in figure Fig. 12b show that the Fuzzy-PID approach guarantees a lower ISE than PID and GS.

6 Conclusion

In this paper we have proposed a novel control architecture to automatically scale out public-cloud resources, only leveraging the base of knowledge available to the customer. In more details, we have investigated the effectiveness of a Fuzzy-PID architecture—taking as input heterogeneous monitoring metrics related to CPU and network capabilities and merged through a fitness function—to cope with highly dynamic cloud operating conditions, whose characteristics are not known in advance. The results of the experimentation performed within Amazon public-cloud environment show that the proposed approach is robust against different realistic workloads, also in presence of VM failures. Moreover, we found that the proposed architecture is flexible, thus proving to be suitable customers with different needs. Finally, when compared to previous control solutions applied to autoscaling, such as PID and GS, the proposed architecture is able to provide better performance thanks to its ability in self-adaptation to workload changes. As a future work, we plan to extend (i) the proposed architecture exploiting its modularity; (ii) the comparison with other scaling architectures (e.g., those proposed in [7], [8])

considering also for them the adoption of heterogeneous metrics.

REFERENCES

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Commun. ACM*, vol. 53, pp. 50–58, Apr. 2010.

[2] K. Rangan, A. Cooke, J. Post, and N. Schindler, "The cloud wars: $100+ billion at stake," tech. rep., Tech. rep., Merrill Lynch, 2008.

[3] A. A. D. P. Souza and M. a. S. Netto, "Using Application Data for SLA-aware Auto-scaling in Cloud Environments," no. i, p. 9, 2015.

[4] M. M. Nejad, L. Mashayekhy, and D. Grosu, "Truthful Greedy Mechanisms for Dynamic Virtual Machine Provisioning and Allocation in Clouds," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 2, pp. 594–603, 2015.

[5] Z. Kang, H. Wang, and L. Wang, "Performance-Aware Cloud Resource Allocation via Fitness-enabled Auction," *IEEE Transactions on Parallel and Distributed Systems*, vol. XX, no. X, pp. 1–1, 2015.

[6] H. C. Lim, S. Babu, and J. S. Chase, "Automated control for elastic storage," in *Proceedings of the 7th international conference on Autonomic computing*, pp. 1–10, ACM, 2010.

[7] M. Maggio, C. Klein, and K.-E. Årzén, "Control strategies for predictable brownouts in cloud computing," IFAC, 2014.

[8] 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 *Proceedings of the 4th ACM European Conference on Computer Systems*, EuroSys '09, (New York, NY, USA), pp. 13–26, ACM, 2009.

[9] A. Gambi and G. Toffetti, "Modeling cloud performance with kriging," in *Software Engineering (ICSE), 2012 34th International Conference on*, pp. 1439–1440, June 2012.

[10] E. Kalyvianaki, T. Charalambous, and S. Hand, "Self-adaptive and self-configured cpu resource provisioning for virtualized servers using kalman filters," in *Proceedings of the 6th International Conference on Autonomic Computing*, ICAC '09, (New York, NY, USA), pp. 117–126, ACM, 2009.

[11] T. Lorido-Botran, J. Miguel-Alonso, and J. Lozano, "A review of auto-scaling techniques for elastic applications in cloud environments," *Journal of Grid Computing*, vol. 12, no. 4, pp. 559–592, 2014.

[12] T. Chen and R. Bahsoon, "Survey and taxonomy of self-aware and self-adaptive autoscaling systems in the cloud," *arXiv preprint arXiv:1609.03590*, 2016.

[13] H. Fernandez, G. Pierre, and T. Kielmann, "Autoscaling Web Applications in Heterogeneous Cloud Infrastructures," *Proceedings of the 2014 IEEE International Conference on Cloud Engineering (IC2E '14)*, 2014.

[14] L. M. Vaquero, L. Rodero-Merino, and R. Buyya, "Dynamically scaling applications in the cloud," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 1, p. 45, 2011.

[15] J. C. Mogul and L. Popa, "What we talk about when we talk about cloud network performance," *SIGCOMM Comput. Commun. Rev.*, vol. 42, pp. 44–48, Sept. 2012.

[16] V. Persico, P. Marchetta, A. Botta, and A. Pescapé, "On network throughput variability in microsoft azure cloud," in *Global Communications Conference (GLOBECOM), 2014 IEEE*, 2015.

[17] V. Persico, P. Marchetta, A. Botta, and A. Pescapé, "Measuring network throughput in the cloud: the case of amazon ec2," *Elsevier, Computer Networks, Special Issue on Cloud Networking and Communications II*.

[18] A. Ashraf, B. Byholm, J. Lehtinen, and I. Porres, "Feedback Control Algorithms to Deploy and Scale Multiple Web Applications per Virtual Machine," *2012 38th Euromicro Conference on Software Engineering and Advanced Applications*, pp. 431–438, 2012.

[19] C.-Z. Xu, J. Rao, and X. Bu, "Url: A unified reinforcement learning approach for autonomic cloud management," *Journal of Parallel and Distributed Computing*, vol. 72, no. 2, pp. 95–105, 2012.

[20] C. Anglano, M. Canonico, and M. Guazzone, "Fc2q: exploiting fuzzy control in server consolidation for cloud applications with sla constraints," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 17, pp. 4491–4514, 2015.

[21] L. Wang, J. Xu, and M. Zhao, "Application-aware cross-layer virtual machine resource management," in *Proceedings of the 9th international conference on Autonomic computing*, pp. 13–22, ACM, 2012.

[22] Z. Gong, X. Gu, and J. Wilkes, "Press: Predictive elastic resource scaling for cloud systems," in *Network and Service Management (CNSM), 2010 International Conference on*, pp. 9–16, IEEE, 2010.

[23] I. Gergin, B. Simmons, and M. Litoiu, "A decentralized autonomic architecture for performance control in the cloud," in *IEEE International Conference on Cloud Engineering (IC2E)*, pp. 574–579, IEEE, 2014.

[24] C. Barna, M. Fokaefs, M. Litoiu, M. Shtern, and J. Wigglesworth, "Cloud adaptation with control theory in industrial clouds," in *Cloud Engineering Workshop (IC2EW), 2016 IEEE International Conference on*, pp. 231–238, IEEE, 2016.

[25] P. Jamshidi, A. M. Sharifloo, C. Pahl, A. Metzger, and G. Estrada, "Self-learning cloud controllers: Fuzzy q-learning for knowledge evolution," in *Cloud and Autonomic Computing (ICCAC), 2015 International Conference on*, pp. 208–211, IEEE, 2015.

[26] J. Choi, Y. Ahn, S. Kim, Y. Kim, and J. Choi, "VM auto-scaling methods for high throughput computing on hybrid infrastructure," *Cluster Computing*, vol. 18, no. 3, pp. 1063–1073, 2015.

[27] Z. Xiao, W. Song, and Q. Chen, "Dynamic resource allocation using virtual machines for cloud computing environment," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 6, pp. 1107–1117, 2013.

[28] D. Ardagna, C. Ghezzi, B. Panicucci, and M. Trubian, "Service provisioning on the cloud: Distributed algorithms for joint capacity allocation and admission control," in *Towards a Service-Based Internet*, pp. 1–12, Springer, 2010.

[29] V. R. Messias, J. C. Estrella, R. Ehlers, M. J. Santana, R. C. Santana, and S. Reiff-Marganiec, "Combining time series prediction models using genetic algorithm to autoscaling web applications hosted in the cloud infrastructure," *Neural Computing and Applications*, pp. 1–24, 2016.

[30] P. Jamshidi, A. Ahmad, and C. Pahl, "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.

[31] G. Santos, J. Maia, L. Moreira, F. Sousa, and J. Machado, "Scale-space filtering for workload analysis and forecast," in *Cloud Computing (CLOUD), 2013 IEEE Sixth International Conference on*, pp. 677–684, June 2013.

[32] N. Huber, F. Brosig, and S. Kounev, "Model-based self-adaptive resource allocation in virtualized environments," in *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pp. 90–99, ACM, 2011.

[33] P. Lama, Y. Guo, and X. Zhou, "Autonomic performance and power control for co-located web applications on virtualized servers," in *Quality of Service (IWQoS), 2013 IEEE/ACM 21st International Symposium on*, pp. 1–10, IEEE, 2013.

[34] H. Ghanbari, M. Litoiu, P. Pawluk, and C. Barna, "Replica placement in cloud through simple stochastic model predictive control," in *2014 IEEE 7th International Conference on Cloud Computing*, pp. 80–87, IEEE, 2014.

[35] P. Padala, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, and K. Salem, "Adaptive control of virtualized resources in utility computing environments," in *ACM SIGOPS Operating Systems Review*, vol. 41, pp. 289–302, ACM, 2007.

[36] R. N. Calheiros, E. Masoumi, R. Ranjan, and R. Buyya, "Workload prediction using arima model and its impact on cloud applications's qos," *IEEE Transactions on Cloud Computing*, vol. 3, no. 4, pp. 449–458, 2015.

[37] H. C. Lim, S. Babu, J. S. Chase, and S. S. Parekh, "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.

[38] D. Grimaldi, V. Persico, A. Pescape, A. Salvi, and S. Santini, "A feedback-control approach for resource management in public clouds," in *2015 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–7, Dec 2015.

[39] K. H. Ang, G. Chong, and Y. Li, "Pid control system analysis, design, and technology," *IEEE transactions on control systems technology*, vol. 13, no. 4, pp. 559–576, 2005.

[40] R. Babuška, *Fuzzy modeling for control*, vol. 12. Springer Science & Business Media, 2012.

[41] 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, pp. 43–55, 2013.

[42] S. Frey, V. Huwwa, and C. Reich, "Fuzzy controled qos for scalable cloud computing services," in *CLOUD COMPUTING 2013, The Fourth International Conference on Cloud Computing, GRIDs, and Virtualization*, 2013.

[43] M. A. H. Monil and R. M. Rahman, "Vm consolidation approach based on heuristics, fuzzy logic, and migration control," *Journal of Cloud Computing*, vol. 5, no. 1, p. 8, 2016.

[44] A. I. Al-Odienat and A. A. Al-Lawama, "The advantages of pid fuzzy controllers over the conventional types," *American Journal of Applied Sciences*, vol. 5, no. 6, pp. 653–658, 2008.

[45] J. Dean and S. Ghemawat, "Mapreduce: a flexible data processing tool," *Communications of the ACM*, vol. 53, no. 1, pp. 72–77, 2010.

[46] H. Goudarzi and M. Pedram, "Multi-dimensional sla-based resource allocation for multi-tier cloud computing systems," in *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pp. 324–331, IEEE, 2011.

[47] W. Iqbal, M. N. Dailey, and D. Carrera, "Sla-driven dynamic resource management for multi-tier web applications in a cloud," in *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*, pp. 832–837, IEEE, 2010.

[48] "Web application hosting in the aws cloud best practices." https://media.amazonwebservices.com/AWS_Web_Hosting_Best_Practices.pdf. Online; accessed Nov '15.

[49] G. Aceto, A. Botta, W. de Donato, and A. Pescapè, "Cloud monitoring: Definitions, issues and future directions," in *1st IEEE International Conference on Cloud Networking, CLOUDNET 2012, Paris, France, November 28-30, 2012*, pp. 63–67, 2012.

[50] V. Persico, A. Montieri, and A. Pescapé, "CloudSurf: a platform for monitoring public-cloud networks," in *2016 IEEE 2nd International Forum on Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI) (IEEE RTSI 2016)*, (Bologna, Italy), pp. 487–492, Sept. 2016.

[51] Y. G. Y. Gong, F. Y. F. Yang, L. H. L. Huang, and S. S. S. Su, "Model-Based Approach to Measuring Quality of Experience," *2009 First International Conference on Emerging Network Intelligence*, pp. 1–4, 2009.

[52] P. Khqj, R. Vwgx, H. G. X. Fq, O. O. B. Frp, Z. Jpdlo, F. R. P. Fdlbfrqj, H. W. F. Dqg, D. Dqg, Y. V. Qfkurql, D. Zhuh, and P. Lq, "A study on QoS/QoE Correlation Model in Wireless-network," pp. 3–8.

[53] S. Aroussi, T. Bouabana-Tebibel, and A. Mellouk, "Empirical QoE/QoS correlation model based on multiple parameters for VoD flows," *GLOBECOM - IEEE Global Telecommunications Conference*, vol. 2, no. 1, pp. 1963–1968, 2012.

[54] D. E. Thomas and B. Armstrong-Helouvry, "Fuzzy logic control-a taxonomy of demonstrated benefits," *Proceedings of the IEEE*, vol. 83, no. 3, pp. 407–421, 1995.

[55] K. J. Astrom, "Pid controllers: theory, design and tuning," *Instrument society of America*, 1995.

[56] H.-J. Zimmermann, *Fuzzy set theory and its applications*. Springer Science & Business Media, 2011.

[57] F. Valdez, P. Melin, and O. Castillo, "A survey on nature-inspired optimization algorithms with fuzzy logic for dynamic parameter adaptation," *Expert Systems with Applications*, vol. 41, no. 14, pp. 6459–6466, 2014.

[58] H. K. Khalil and J. Grizzle, *Nonlinear systems*, vol. 3. Prentice hall New Jersey, 1996.

[59] G. Soundararajan, C. Amza, and A. Goel, "Database replication policies for dynamic content applications," in *ACM SIGOPS Operating Systems Review*, vol. 40, pp. 89–102, ACM, 2006.

[60] H.-X. Li and H. Gatland, "A new methodology for designing a fuzzy logic controller," *IEEE Transactions on Systems, Man and Cybernetics,*, vol. 25, no. 3, pp. 505–512, 1995.

[61] Z.-Y. Zhao, M. Tomizuka, and S. Isaka, "Fuzzy gain scheduling of pid controllers," in *Control Applications, 1992., First IEEE Conference on*, pp. 698–703, IEEE, 1992.

[62] S. Das, I. Pan, S. Das, and A. Gupta, "A novel fractional order fuzzy pid controller and its optimal time domain tuning based on integral performance indices," *Engineering Applications of Artificial Intelligence*, vol. 25, no. 2, pp. 430–442, 2012.

[63] G. Fiengo, L. Glielmo, and S. Santini, "On-board diagnosis for three-way catalytic converters," *International Journal of Robust and Nonlinear Control*, vol. 11, no. 11, pp. 1073–1094, 2001.

[64] L. Leong *et al.*, "Gartner: Magic quadrant for cloud infrastructure as a service, worldwide report," 2016.

[65] D. Emma, A. Pescape, and G. Ventre, "Analysis and experimentation of an open distributed platform for synthetic traffic generation," in *Distributed Computing Systems, 2004. FTDCS 2004. Proceedings. 10th IEEE International Workshop on Future Trends of*, pp. 277–283, May 2004.

[66] "Amazon cloudwatch monitoring service." http://aws.amazon.com/cloudwatch/. Online; accessed Nov '15.

[67] C. Klein, M. Maggio, K.-E. Årzén, and F. Hernández-Rodriguez, "Brownout: Building more robust cloud applications," in *Proceedings of the 36th International Conference on Software Engineering*, pp. 700–711, ACM, 2014.

[68] "Worldcup98 web site access log." http://ita.ee.lbl.gov/html/contrib/WorldCup.html. Online; accessed Mar '16.

[69] H. Ghanbari, B. Simmons, M. Litoiu, and G. Iszlai, "Exploring alternative approaches to implement an elasticity policy," in *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pp. 716–723, IEEE, 2011.

Valerio Persico is a Post Doc at the Department of Electrical Engineering and Information Technology of University of Napoli Federico II. He has a PhD in computer engineering from the University of Napoli Federico II. His research interests fall in the area of networking and of IP measurements; in particular on cloud monitoring and measurement and path tracing. He has been awarded the Best Student Paper Award at CoNext 2013 for his paper Don't Trust Traceroute (Completely).

Domenico Grimaldi received the M.Sc. degree in control system engineering from the University of Napoli in 2014 with a thesis on model reference adaptive control applied to internal combustion engine control. Since 2015 he is reseach associate at the Department of Electrical Engineering and Information Technology of University of Napoli, Federico II. His research interests include analysis and control of cloud based systems with applications to resources managment in public clouds.

Alessandro Salvi received the M.Sc. degree in automation engineering and the Ph.D. degree in control systems engineering from the University of Napoli Federico II, Italy, in 2010 and 2014, respectively. In 2015 he won a Postdoctoral fellow at the Department of Electrical Engineering and Information Technology of the same University. His research interests include analysis and control of complex networks with applications to automotive engineering and power systems.

Stefania Santini is an Associate Professor of Automatic Control and Member of the Academic Senate of the University of Napoli Federico II (Department of Electrical Engineering and Information Technology). Her research interests include nonlinear control theory, analysis and control of hybrid and piecewise-smooth dynamical systems, time-delayed systems, cooperative control with applications that range from computational biology to mechanical systems, automotive engineering and transportation technologies. She is involved in many projects in cooperation with industry, including enterprises operating in the automotive field.

Antonio Pescapé [SM '09] is a Full Professor at the Department of Electrical Engineering and Information Technology of the University of Napoli Federico II (Italy). His research interests are in the networking field with focus on Internet Monitoring, Measurements and Management and on Network Security. Antonio Pescapé has co-authored over 180 journal and conference publications and he is co-author of a patent. For his research activities he has received several awards, comprising a Google Faculty Award, several best paper awards and two IRTF (Internet Research Task Force) ANRP (Applied Networking Research Prize).