A feedback-control approach for resource management in public clouds

Domenico Grimaldi[1], Valerio Persico[1], Antonio Pescapé[1,2], Alessandro Salvi[1], Stefania Santini[1]
[1]University of Naples "Federico II" (Italy)
[2]NM2 srl (Italy)
dom.grimaldi@studenti.unina.it, {valerio.persico,pescape,alessandro.salvi,stefania.santini}@unina.it

***Abstract*—Nowadays, more and more the industry and market depend on cloud-based infrastructures for delivering IT services. To this aim cloud-based infrastructures are changing continuously, increasing their complexity especially for the management of cloud resources. Control and management of resources (e.g., virtual machines, VMs) are of paramount importance to adjust resources automatically allocated to an application and for delivering quality-assured services to final users. In this paper, we propose a feedback-based control approach for the management of VMs in the AWS EC2 public cloud. First, we evaluate the proposed *Gain Scheduling* policy against different workloads. Second, we provide results on the robustness of the proposed *Gain Scheduling* policy in presence of failures. Finally, we compare our approach to state-of-the-art control approaches for cloud resources. Our results indicate that the proposed control strategy guarantees, without the need of *a priori* information on system dynamics or complex estimations of the operating conditions, high performance with respect to both constant and time-varying workloads as well as in spite of sudden VM failures.**

I. Introduction

Nowadays, cloud computing (hereinafter, *cloud*) is a standard de facto in the IT world for providing services to final users [9]. 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. services on-demand, reduced costs, optimized hardware and software resources utilization, and performance flexibility. Therefore, companies use clouds for different purposes, such as running batch jobs, hosting web applications, or for backup and storage. The *pay-as-you-go* pricing model is the characteristic that more directly captures the appealing economic benefit to the customer [16]. Indeed, the absence of up-front capital expense allows capital to be redirected to core business investment. *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 – is a key characteristic of cloud systems and allows matching resources to workload much more closely. For instance a cloud customer can decide to start on-demand new application servers or allocate more storage capacity to them just when needed, and without any up-front provisioning. In this way, it is possible to dramatically raise the server utilization level that would be without cloud-based approaches extremely low (from 5% to 20% [35], [38]), due to typical *overprovisioning* needed to properly manage peak workload that can exceed the average by factors of 2 to 10 [16].

However, deciding *the right amount* of resources is not an easy task. Indeed, *appropriately* dimensioning resources to applications is a crucial issue in cloud computing, although elasticity allows cloud customers to acquire and release resources dynamically according to changing demands. Since applications may face large fluctuating loads [29], it would be desirable in presence of unplanned spike loads to free the cloud customers from the burden of deciding the resource allocation. In other words, it would be desirable to have an *automatic control* for adjusting resources allocated to an application automatically and based on its needs at any given time. This need is witnessed also by the recent scientific literature on control and management of cloud resources (and especially of virtual machines, VMs) that have to deliver quality-assured services to final users. *Auto-scaling* approaches (like threshold-based rules or policies) are very popular among cloud providers such as Amazon EC2 [2], or third-party tools like RightScale [4]: due to their simplicity, these policies are very appealing to cloud customers. However, setting thresholds is a per-application task, and requires a deep understanding of workload trends. Therefore, the management of cloud resources is a key challenge and a hot research topic [15], [26], [21], [41], [32], [6].

In this paper, we propose a feedback-based control strategy using a *Gain Scheduling* policy to dynamically allocate cloud resources (i.e., VMs) to users of public clouds so to guarantee a pre-specified Service Level Objective despite the presence of large fluctuating loads and VM failures, without the need of previsional models of the system behavior [11]. While in literature approaches for analyzing cloud resource management are based on simulation [12] or implemented in private clouds [27], [28], [30], [8], [7], [33], [17], in this work we implemented and tested the proposed control strategies on a real public cloud environment, the AWS EC2 public cloud [2]. For the sake of comparison and repeatability and to foster further research in this field, we publicly release to the community the code we have implemented[1].

To the best of our knowledge, the main advantages of the proposed approach with respect to the state of the art can be summarized as: *(i)* the proposed solution (Gain Scheduling) has been implemented and then extensively experimentally validated in the public cloud environment AWS EC2; *(ii)* the Gain Scheduling approach guarantees robustness with respect to synthetic and real-trace workloads characterized by a high rate of variability without requiring any *a priori* knowledge or information on the current workload or its on-line measurement/estimation; *(iii)* adaptive control gains have been optimized with an automatic optimal tuning procedure with respect to the integral squared error (ISE); *(iv)* the control

[1]http://www.comics.traffic.unina.it/cloud

signal is actuated through a nonlinear approach based on control aggressiveness, where the rate at which the controller allocate resources can be also opportunely chosen by the user; *(v)* we evaluated the performance of the proposed control approach against VMs failures.
The effectiveness of the approach has been also shown by an experimental comparison with fixed-gains controllers which have been proposed in the recent literature in the public Cloud environment [18].

The paper is organized as follows. Sec. II provides an overall picture of the related literature and positions the paper accordingly. Sec. III describes the problem statement, while Sec. IV presents the proposed control strategy. Sec. V shows the obtained results, first describing the experimental testbed (Sec. V-A) then presenting the experimental results (Sec. V-B). Finally, Sec. VI ends the paper with conclusion remarks.

II. Related Work

Control approaches for automatically scaling resources that leverage cloud elasticity have recently attracted the interest of the scientific community. Due to system complexity some of the works do not address the problem in the real system, but propose the extensive use of cloud simulators to mimic the dynamic response of the cloud system under the proposed control action (e.g. [8], [7]). One of the main drawbacks of those approaches is that they rely on performance models instead of reality, hence results strongly depend on the reliability of the simulations. The experimental validation of cloud control strategies in a real environment has been often addressed by designing custom in-house testbeds such as server cluster or private cloud deployments [27], [28], [34], [30], [33], [17], [23]. In all these works different control approaches, such as, for example, Proportional-Integral (PI), adaptive deadbeat, adaptive threshold mechanism, feedback plus feed-forward, have been analyzed to investigate the ability of the proposed strategies to regulate the average response time of the web application latency or the average CPU load. For all the proposed approaches the control effectiveness has been only tested in the private experimental set-up and the control design is often based on the precise knowledge of the plant and on-line measurements of all the variable of interest. This information is exploited to derive mathematical models of the system performance and/or for the prediction of the incoming variable loads acting on the private cloud. For example, *Padala et al.* [33] 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 normalized performance, *Gambi and Toffetti* [17] exploited a Kriging model to predict job completion time as function of the number of VMs, the number of incoming requests, and the jobs enqueued at the master node and *Kalyvianaki et al.* [23] designed different SISO and MIMO controllers to determine the CPU allocation of VMs, relying on the prediction coming from on-line Kalman filters. A different attempt to cope with workload variability has been instead proposed in [11]. Here a very complex strategy combines an integral control action with a performance models (to predict workloads) and a statistical machine learning techniques to control internet data-centers. Obviously, the practical use of these approach is strongly limited from the correct on-line use of these estimation techniques. Indeed, in predictive (or proactive) approaches the prediction accuracy highly depends on the history window size, (i.e. the number of observations), and the adaptation window (i.e. the observation interval) [22]. Moreover, they require stable workloads to apply long learning [37].

In this paper, we aim to investigate the feasibility and effectiveness of a Gain Scheduling strategy for automatically scaling cloud resources in a real production environment. Note that only few attempts have made in the very recent literature to solve the problem of cloud resources control in public cloud for practical-use scenarios. For example *Gergin et al.* [18] propose a decentralized architecture for multi-tier application based on multiple fixed-gains Proportional-Integral-Derivative (PID) controllers and they test the strategy on Amazon EC2 cloud. Differently from our approach, the control strategy suffers from a lack of flexibility, since it relies on fixed parameters, tuned with a heuristic method, and the control action is able to allocate/deallocate just one VM at each time instant. This choice can be ineffective when the control has to counteract the unpredictable effects of high variable workloads and time-varying operating conditions.

III. Problem Statement

We aim at solving the following problem: properly dimensioning the *set of resources* allocated to an *application*, in order to achieve/guarantee the *desired performance level* i.e. a pre-specified Service Level Objective (SLO). By leveraging cloud elasticity, the cloud customer is able to decide at runtime the dimension of the resource set adopted and – consequently – the quantity of resources she pays for. In more details, we aim at designing a feedback-based control strategy to dynamically allocate cloud resources to users on the basis of the actual *system workload* and according to the SLO.

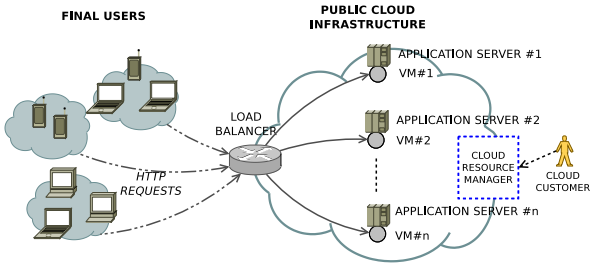

Fig. 1: Reference system architecture.

The cloud service model we refer to is the Infrastructure as a Service (IaaS) model, therefore the resource considered are VMs, that can be activated or terminated on customer's request.

TABLE I: Actors and terms.

Resources	*Virtual Machine cluster* hosting multiple servers (IaaS model)
Workload	*Arithmetic task execution* started by web requests generated by the final users
SL	*Average CPU load of the VM cluster* impacting tasks completion time and latency
Objective	*Keeping the VM-set average CPU load close to the SLO* to obtain expected performance and avoid revenue loss

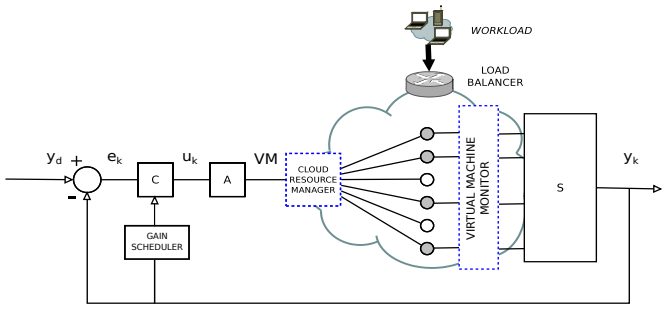

Fig. 2: The closed loop system.

The system utilization is measured by observing the CPU load of the active VMs. Without compromising generality, the SLO also is defined on the CPU load that is a valuable metric for our problem, as being representative of the QoS guaranteed by the system [18], [42]. Indeed, it is representative of the state of the VM and of the hosted application, by impacting on the performance perceived by the user, such as request-response latency or task completion-time. The final objective of the proposed control strategy is to keep the Service Level (SL) – defined as the average CPU load of the VM-set – close to the SLO. In fact, activating a number of VMs larger than the one strictly needed causes revenue loss to the cloud customer. Otherwise, if the set of VMs does not properly grow together with the workload, the performance perceived by the final user could dramatically fall down. In Tab. I the key points of the problem are summarized. As an instance of the described problem, we designed the architecture described in the following (see Fig. 1). We considered an application, whose interface exposes a set of heavy computation functionalities. The application back-end consists of a VM cluster. The cluster is composed of a set of VMs, connected to a load-balancer instructed to equally distribute the incoming requests across them. On each VM of the cluster, a server has been properly configured, in order to serve the requests generated by the final-users and forwarded by the load-balancer.

IV. Control and System Design

As anticipated in the previous sections, the control aim is to regulate the application's Service Level (SL) at a predefined level (SLO) as specified in the Service Level Agreement (SLA).

The control strategy is implemented according to the scheme represented in Fig. 2, where a Gain Scheduling PID (Proportional Integral Derivative) - block C in the figure - dynamically adjusts at each time interval k the actual number of the VMs running on a cloud data-center in order to meet application-level QoS goals while achieving the best performances with respect to variable workloads. Note that from the control perspective workloads can be seen as unknown and unmodelled exogenous disturbance and the control solutions should react to it.

To this aim the actual SL of the cloud application at time interval k, say $y_k = y(t_k)$, is evaluated on the base of the average CPU utilization and, hence, compared to the target y_d. The algorithm automatically adapts its control gains to drive the actual error $e_k = (y_d - y_k)$ to zero and to counteract workload variations. To actuate the control signal a scaling algorithm is designed to provide or terminate a different number of VMs according to the control aggressiveness.

A. Control Strategy

The proposed control strategy is based on a variable gains PID discrete-time controller that can be mathematically formalized as:

$$\begin{aligned} u_k = & \quad u_{k-1} + \tfrac{T_d(e_k,\Delta e_k)}{\Delta t} e_{k-2} \\ & + k_p(e_k,\Delta e_k)\left[-1 - \tfrac{2T_d(e_k,\Delta e_k)}{\Delta t}\right] e_{k-1} \\ & + k_p(e_k,\Delta e_k)\left[\left(1 + \tfrac{\Delta t}{T_i(e_k,\Delta e_k)} + \tfrac{T_d(e_k,\Delta e_k)}{\Delta t}\right) e_k\right] \end{aligned} \tag{1}$$

with

$$\begin{aligned} T_i(e_k,\Delta e_k) &= \frac{k_p(e_k,\Delta e_k)}{k_i(e_k,\Delta e_k)}, \\ T_d(e_k,\Delta e_k) &= \frac{k_d(e_k,\Delta e_k)}{k_p(e_k,\Delta e_k)}, \end{aligned} \tag{2}$$

being u_k the control action at time interval k; $e_k = y_d - y_k$ the closed-loop error (i.e. the difference of the desired output y_d and the measured output y_k); $\Delta e_k = e_k - e_{k-1}$ and Δt the sampling time. Moreover $k_p(e_k,\Delta e_k)$, $k_i(e_k,\Delta e_k)$, and $k_d(e_k,\Delta e_k)$ are control parameters selected such that the integral quadratic error (ISE) of the closed loop is a minimum. The gain scheduling algorithm adjust on-line and in real time the gains values according to the actual closed-loop error dynamics $(e_k,\Delta e_k)$.

Note that automatic and well assessed methods for the tuning of PI/PID control parameters are today available in order to reduce the time required by the tuning phase [10]. As usual, closed-loop dynamics strongly depend on the specific choice of the control gains. Indeed a wrong selection of these parameters may jeopardize the system performance, leading to instability in some critical cases. The tuning procedures, based on heuristic methods, may result to be time-consuming or not effective. Furthermore, often these methods do not provide any optimality of the solution, which is essential in the context of resource allocation. To overcome this limit here we exploit use an automatic optimal tuning procedure for control gains. The approach is based on a cost function to be minimized so to optimize the control gains over a finite control horizon during step responses of the systems in different operating conditions (30 in total). Since optimal gains tuned for one operating condition may not work well when the system is operated in another condition, the range of cloud operations has been dived into several different zones according to different workloads and time slots. The optimality criteria considered in the cloud control system design is with respect to the quality of the transient response, induced for example by sudden changing in the workload, such as percentage overshoot, rise time, settling time, etc. Such optimalities are achieved by choosing k_p, k_i and k_d to minimize the integral squared error (ISE) and the optimal values are hence determined using sequential quadratic

TABLE II: Gain Scheduler. Parameters ranges.

$k_p^{\min} = 90$	$k_p^{\max} = 120$
$k_i^{\min} = 0.001$	$k_i^{\max} = 0.05$
$k_d^{\min} = 0.001$	$k_d^{\max} = 0.002$
$\#VM_{\max}$	4

programming to minimize the cost function $J(k_p, k_i, k_d)$ defined as follows:

$$J(k_p, k_i, k_d) = \min_{k_p,k_i,k_d} \int_0^\infty [e(t)]^2 dt \simeq \sum_0^\infty [e_k]^2. \quad (3)$$

The positive function $J(k_p, k_i, k_d)$ in (3) is so that it assumes finite values for stable solutions of the cloud system under the control action (1) and smaller values of $J(k_p, k_i, k_d)$ corresponds somehow to a better tracking of the reference trajectory. The optimization procedure is initialized by selecting the gains with the classical closed-loop Ziegler and Nichols method [43]. Notice that the optimization problem discussed here is different from the one usually considered in optimal control theory where the controller structure is not *a priori* fixed. Here instead the structure of the controller is fixed to the PID structure and the aim is the parameters optimization.

Experimental observations disclose the large variability of public cloud dynamics under various operating conditions, time slots and workload variations that can not be predicted for public clouds in practical-use scenarios and that depend on the actual condition of the system. Hence, to achieve further robustness and effective disturbance rejection, the actual values of the control gains are on-line selected, among the ones estimated during the optimal tuning phase, through a *Gain Scheduler* that changes the controller parameters according to the actual error e_k and its rate Δe_k, i.e $k_p = k_p(e_k, \Delta e_k) \in [k_p^{\min}, k_p^{\max}]$; $k_i = k_i(e_k, \Delta e_k) \in [k_i^{\min}, k_i^{\max}]$; $k_d = k_d(e_k, \Delta e_k) \in [k_d^{\min}, k_d^{\max}]$ (see Tab. II for the parameter values). We remark that in order to not introduce large disturbances during on-line control adjustments linear interpolations are used to obtain smooth controller gain transition surfaces.

B. Actuation Policy

The correct implementation of the controller u_k in (1) requires the design of scaling algorithm to actuate the proportional, integral and derivative actions (see the actuation block A in Fig. 2). In our approach the number of VMs to be provisioned or terminated at a time interval k, say VM_k, is set according to the aggressiveness of the control signal. Note that the classical strategy of provisioning or terminating ± 1 VM at time (as done in [18]) can be ineffective in real scenarios, since it may be too slow in scaling up in the case of sudden peak loads or it may result in unwanted longer provisioning periods during scaling down. In our approach, the number of VMs to be provided or terminated at time interval k, say VM_k, depends on the actual value of the control signal u_k according to the following dead-zone with saturation:

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

where α and β determine how fast the controller adds and removes VMs and, as usual when dealing with nonlinear actuators, the amplitude of the dead-zone ϵ has been set so to prevent oscillations in the actuation signals [24]. Note that the actuation characteristic has been discretized with the classical sample-and-hold method.

C. System Monitoring

The feedback control-loop in Fig. 2 requires a sensor (S) to measure the value of the output variables. Here a monitoring subsystem, e.g Amazon's Cloudwatch service, provides information on how many VMs are currently running and their CPU utilization, say $CPU_{VM_j}(k)$ (note that the monitoring interval is set as 3 minutes). From these data the actual SL at time interval k is evaluated as:

$$y_k = SL_k = \frac{CPU_{VM_1}(k) + \ldots + CPU_{VM_M}(k)}{\#VM(k)} \quad (5)$$

where $\#VM(k)$ is the maximum number of VMs that are effectively running at time interval k and $CPU_{VM_j}(k)$ is the CPU utilization of the j-th VM running at time interval k (i.e. $VM_j(k)$ with $j = 1, \ldots, M$ being $M \leq \#VM_{\max}$).

V. Evaluation

In this section, we first describe the experimental testbed (Sec. V-A), and then show the results of our experimental campaigns (Sec. V-B).

A. Experimental Testbed

We set up a cloud application according to the problem statement described in Sec. III. The testbed is composed of two main blocks: (i) a dynamically resizable cloud resource cluster, made up of AWS EC2 VMs and instructed to run previously configured CPU-intensive tasks on user's request; (ii) a request generation block that implements user's request.

Cloud application deployment. 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. On each VM, a web server has been configured, in

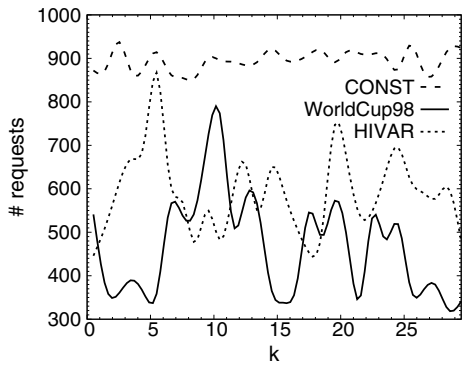

Fig. 3: Workload evolution over time: (i) constant workload, CONST (long-dashed line); (ii) WorldCup98 workload (solid line); (iii) highly variable workload, HIVAR (short-dashed line). The y-axis reports the cumulative number of requests for each time interval.

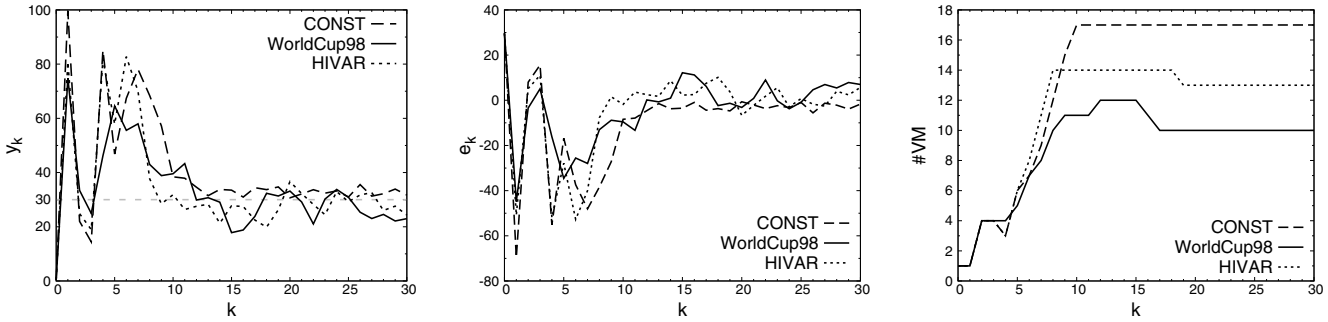

Fig. 4: Gain scheduling policy in the presence of different workloads. (a): y_k in (5), Service Level at the time interval k; (b): $e_k = (y_d - y_k)$, error of the actual Service Level at the time interval k w.r.t. the Service Level Objective; (c): $\#VM(k)$, number of VMs that are effectively running at time interval k.

order to serve the HTTP requests forwarded by the load-balancer. For each incoming request, the web-server has been instructed to perform a varying number of CPU-intensive tasks. For all our experimentations, we adopted general purpose *micro instances*, representing a feasible choice for our long experimental sessions thanks to their limited cost [2]. The choice of this particular kind of instances does not represent a limitation however, since the proposed control approach does not depend on the type of the VMs employed. 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* that is made available by the provider itself. In order to monitor the CPU load of each of the VMs within the cluster, we took advantage of *Cloudwatch* that is the monitoring service provided by Amazon [1].

Final-user emulation. For our experimentations, a geographically separated node was configured to produce web requests and to impose a workload on the system. The realistic workload has been generated by exploiting *Httpmon* [3] [25]. Httpmon is a HTTP request generator purposely designed for executing experiments related to computing capacity shortage avoidance in cloud computing. 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 the three different workloads depicted in Fig. 3. : *(i)* constant workload (CONST); *(ii)* WorldCup98 web server workload (WorldCup98) [5] (note that WorldCup98 is a meaningful benchmark extensively used in the cloud scientific literature[19], [39], [40], [20], [36], [31], [13]); *(iii)* highly variable synthetic workload (HIVAR) obtained by amplifying all the values and the frequencies of WorldCup98 workload as suggested in [18]. For each experiment, each workload request has a duration of 90 minutes (i.e., 30 $\times$ 3-minute cycles). The overall experimental activity (control tuning and extensive validation) amounted to 200 hours.

B. Experimental results

The effectiveness of the proposed Gain Scheduling algorithm is shown in this section through representative experimental examples. In more details: *(i)* we evaluate the proposed Gain Scheduling policy against different workloads; *(ii)* we provide results on the robustness of the proposed Gain Scheduling policy in presence of failures; *(iii)* we compare our approach to state-of-the-art solutions (see Sec. II). We remark that robustness to disturbances and unmodeled dynamics is a fundamental issue in cloud management, since for the case of public cloud in practical-use scenario, the dynamical response of the system as well as the sudden workloads variations result to be unknown and unpredictable. Hence, one of key features of a control strategy to be experimentally evaluated is its ability to automatically compensate the undesired effects of loads variability on the system without the need of *a priori* information on system dynamics or complex measurements/estimations of the operating conditions.

Gain Scheduling performance in the presence of different workloads. We analyze the ability of the Gain Scheduling algorithm to regulate the SL at a predefined reference value $y_d = SLO = 30$ in the presence of the three different variable workloads described in Sec. V-A. Results in Fig. 4 confirm that in the cases where operating conditions are highly varying, the required control objectives are always achieved with short-term performance degradations and avoiding instabilities. Specifically, for all the different workloads, the system, starting with 1 allocated VM, is able to reach the desired target in about 10 time intervals (interval duration is 3 minutes) deciding the amount of VMs to allocate without the need of a prediction of the incoming load and of the service times (see Fig. 4a). Note that once the desired target is reached, the control algorithm guarantees good performance and the residual tracking error due to sudden variations in the operating conditions is negligible and never exceeds 10% as shown in Fig. 4b. The number of VMs to be provided during the experiments is depicted in Fig. 4c. As expected, according to the actuation policy described Sec. IV, the VM scaling-up rate depends on control aggressiveness. Furthermore, the maximum number of virtual machines to be allocated at the equilibrium point (i.e. when $y_k \approx y_d$) depends on the average values of the different workloads. To better analyze the effectiveness of the control strategy, the time history of the response time is reported in Fig. 5. Latency can be a key issue for some cloud applications that should act in real-time. In those cases users require low variability of the response time, whose average value depends on the specific application. Results in Fig. 5 confirm the control effectiveness and disclose the low variability in the response time despite the high variability of the operating conditions.

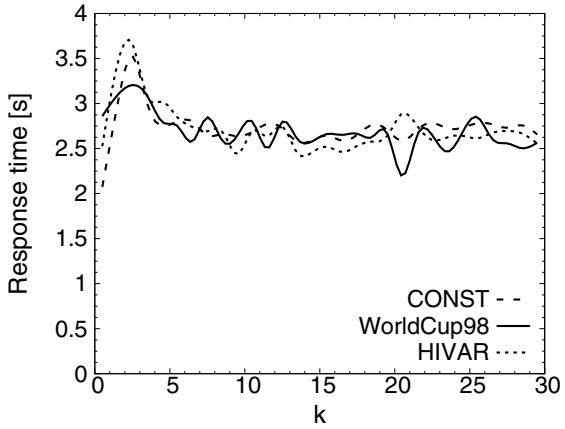

Fig. 5: Time history of the response time in the presence of the different workloads.

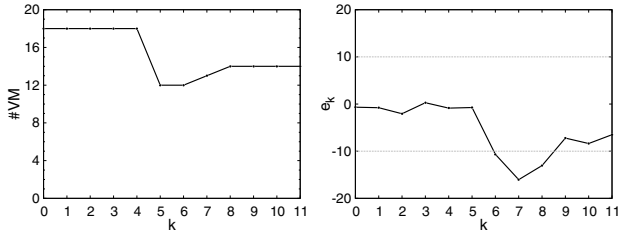

Fig. 6: Gain scheduling policy with constant workload and VM failures. (a): $\#VM(k)$, number of VMs that are effectively running at time interval k; (b): $e_k = (y_d - y_k)$, error of the actual Service Level at the time interval k w.r.t. the Service Level Objective.

Gain Scheduling robustness in the presence of VM failures. We consider the closed-loop system under the action of the CONST workload (see Fig. 6), being again the constant reference signal set as $y_d = 30$. The system is at the equilibrium point $y_k = y_d$ when we cause a hard failure to happen. $1/3$ of VMs that are running crash at time interval $k = 4$ (6 among the 18 that are working) as being suddenly terminated (see Fig. 6a). Due to this critical event, the tracking error increases (it reaches a percentage error of more than 10%) and, accordingly, the control action varies, on-line adapts its gains, and counteracts the effect of the failures. The tracking error is then again within $\pm 10\%$ bound at time interval $k = 9$. Note that during the entire failure/recovery period the tracking error is always less than $\pm 20\%$.

Comparison against fixed gain controllers. We compare the Gain Scheduling (GS) strategy with respect to fixed gain controllers Proportional (P), Proportional-Integral (PI), and Proportional-Integral-Derivative (PID). Since the fixed gain performance strongly depends on the choice of the control gains values, to perform a fair comparison among the different strategies, gains have been all tuned with the optimal procedure described in Sec. IV. To compare the behavior of the different controllers different metrics can be used. The best controller keeps low error-based metrics. In order to better evaluate how the control meets strict SLOs (the requirement is the

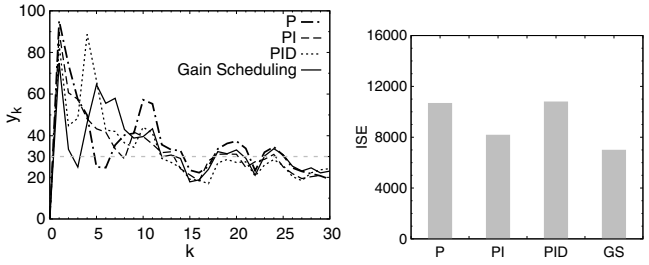

Fig. 7: Gain Scheduling (GS) vs. fixed gain controllers (P, PI, PID) with FIFA98 Workload. (a): Time history of the current Service Level, y_k in (5) ; (b): Integral of Squared Errors - ISE

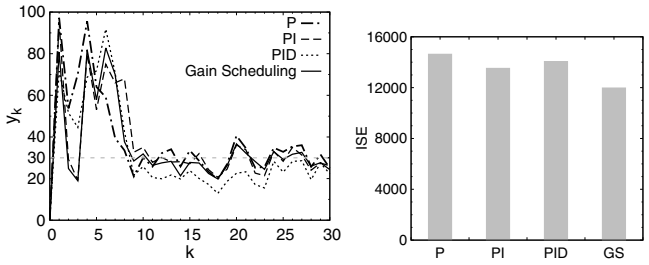

Fig. 8: Gain Scheduling (GS) vs. fixed gain controllers (P, PI, PID) with HIVAR Workload. (a): Time history of the current Service Level, y_k in (5) ; (b): Integral of Squared Errors - ISE.

constant value $y_d = 30$) during transients induced by workload variations (WorldCup98 and HIVAR), here we exploit the Integral of Squared Errors (ISE) [24]. Results in Fig. 7 and Fig. 8 confirm the ability of Gain Scheduling in guaranteeing the required objective while reacting in conditions of crowding and loads variability due to its ability in adapting at runtime to different conditions.

VI. Conclusion

We have proposed a feedback-based control approach (leveraging Gain Scheduling policy) for the management of VMs in the AWS EC2 public cloud. We have evaluated the proposed policy against different workloads and we have shown its robustness in presence of VM failures. Also, we have compared our policy to state-of-the-art control approaches for cloud resources. Our results indicate that our proposal guarantees high performance, without the need of *a priori* information on system dynamics or complex measurements/estimations of the operating conditions. In our ongoing work we are implementing other sophisticated control polices (e.g., model based ones) and we are testing application-based allocation policies driven by traffic identification [14].

ACKNOWLEDGEMENTS

This work is partially funded by the MIUR projects: PLATINO ($PON01\_01007$), SMART HEALTH ($PON04a2\_C$), and SIRIO ($PON01\_02425$) and art. 11 DM 593/2000 for NM2 srl (Italy). The experimental work in this paper was also supported by a grant provided by Amazon AWS in Education.

REFERENCES

[1] Amazon cloudwatch monitoring service. http://aws.amazon.com/cloudwatch/. Online; accessed Mar '15.

[2] Amazon web services website. http://aws.amazon.com/. Online; accessed Mar '15.

[3] Cloud control project, httpmon github webpage. https://github.com/cloud-control/httpmon. Online; accessed Mar '15.

[4] Rightscale. http://www.rightscale.com/. Online; accessed Mar '15.

[5] Worldcup98 web site access log. http://ita.ee.lbl.gov/html/contrib/WorldCup.html. Online; accessed Mar '15.

[6] G. Aceto, A. Botta, W. de Donato, and A. Pescapè. Cloud monitoring: A survey. *Computer Networks*, 57(9):2093 – 2115, 2013.

[7] A. Ali-Eldin, M. Kihl, J. Tordsson, and E. Elmroth. Efficient provisioning of bursty scientific workloads on the cloud using adaptive elasticity control. In *Proceedings of the 3rd Workshop on Scientific Cloud Computing Date*, ScienceCloud '12, pages 31–40. ACM, 2012.

[8] A. Ali-Eldin, J. Tordsson, and E. Elmroth. An adaptive hybrid elasticity controller for cloud infrastructures. In *Network Operations and Management Symposium (NOMS), 2012 IEEE*, pages 204–212, April 2012.

[9] 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*, 53(4):50–58, Apr. 2010.

[10] K. J. Aström and T. Hägglund. Pid controllers: theory, design and tuning. 1995.

[11] P. Bodık, R. Griffith, C. Sutton, A. Fox, M. Jordan, and D. Patterson. Statistical machine learning makes automatic control practical for internet datacenters. In *Proceedings of the 2009 conference on Hot topics in cloud computing*, pages 12–12, 2009.

[12] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience*, 41(1):23–50, 2011.

[13] A. Chandra, W. Gong, and P. Shenoy. Dynamic resource allocation for shared data centers using online measurements. In *Quality of Service–IWQoS 2003*, pages 381–398. Springer, 2003.

[14] A. Dainotti, A. Pescapè, and C. Sansone. Early classification of network traffic through multi-classification. In *Traffic Monitoring and Analysis - Third International Workshop, TMA 2011, Vienna, Austria, April 27, 2011. Proceedings*, pages 122–135, 2011.

[15] E. Feller, L. Rilling, and C. Morin. Snooze: A scalable and autonomic virtual machine management framework for private clouds. In *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (Ccgrid 2012)*, CCGRID '12, pages 482–489, Washington, DC, USA, 2012. IEEE Computer Society.

[16] A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, and I. Stoica. Above the clouds: A berkeley view of cloud computing. *Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS*, 28:13, 2009.

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

[18] 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)*, pages 574–579. IEEE, 2014.

[19] 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*, pages 716–723. IEEE, 2011.

[20] 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*, pages 9–16. IEEE, 2010.

[21] J. Hu, J. Gu, G. Sun, and T. Zhao. A scheduling strategy on load balancing of virtual machine resources in cloud computing environment. In *Parallel Architectures, Algorithms and Programming (PAAP), 2010 Third International Symposium on*, pages 89–96, Dec 2010.

[22] 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*, pages 95–104. ACM, 2014.

[23] 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, pages 117–126. ACM, 2009.

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

[25] 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*, pages 700–711. ACM, 2014.

[26] H. A. Lagar-Cavilla, J. A. Whitney, A. M. Scannell, P. Patchin, S. M. Rumble, E. de Lara, M. Brudno, and M. Satyanarayanan. Snowflock: Rapid virtual machine cloning for cloud computing. In *Proceedings of the 4th ACM European Conference on Computer Systems*, EuroSys '09, pages 1–12, New York, NY, USA, 2009. ACM.

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

[28] 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*, pages 13–18. ACM, 2009.

[29] 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*, 12(4):559–592, 2014.

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

[31] H. Mi, H. Wang, G. Yin, Y. Zhou, D. Shi, and L. Yuan. Online self-reconfiguration with performance guarantee for energy-efficient large-scale cloud computing data centers. In *Services Computing (SCC), 2010 IEEE International Conference on*, pages 514–521. IEEE, 2010.

[32] M. Mishra, A. Das, P. Kulkarni, and A. Sahoo. Dynamic resource management using virtual machine migrations. *Communications Magazine, IEEE*, 50(9):34–40, September 2012.

[33] 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, pages 13–26. ACM, 2009.

[34] S.-M. Park and M. Humphrey. Self-tuning virtual machines for predictable escience. In *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 356–363. IEEE Computer Society, 2009.

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

[36] N. Roy, A. Dubey, and A. Gokhale. Efficient autoscaling in the cloud using predictive models for workload forecasting. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*. IEEE, 2011.

[37] 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*, pages 677–684, June 2013.

[38] L. Siegele. *Let it rise: A special report on corporate IT*. Economist Newspaper, 2008.

[39] B. Simmons, H. Ghanbari, M. Litoiu, and G. Iszlai. Managing a saas application in the cloud using paas policy sets and a strategy-tree. In *Proceedings of the 7th International Conference on Network and Services Management*, pages 343–347. International Federation for Information Processing, 2011.

[40] B. Urgaonkar, P. Shenoy, A. Chandra, P. Goyal, and T. Wood. Agile dynamic provisioning of multi-tier internet applications. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 3(1):1, 2008.

[41] H. N. Van, F. Tran, and J.-M. Menaud. Sla-aware virtual resource management for cloud infrastructures. In *Computer and Information Technology, 2009. CIT '09.*, volume 1, pages 357–362, Oct 2009.

[42] A. Zhang, P. Santos, D. Beyer, and H. Tang. Optimal server resource allocation using an open queueing network model of response time. *HP laboratories Technical Report, HPL2002301*, 2002.

[43] J. G. Ziegler and N. B. Nichols. Optimum Settings for Automatic Controllers. *Transactions of ASME*, 64:759–768, 1942.