SLA-aware Resource Management for Application Service Providers in the Cloud

Valeria Cardellini, Emiliano Casalicchio, Francesco Lo Presti, Luca Silvestri
Department of Computer Science, Systems and Production
University of Roma "Tor Vergata", Roma, Italy
Email:{cardellini,casalicchio,silvestri}@ing.uniroma2.it, lopresti@info.uniroma2.it

***Abstract*—In the today Internet of Services, one of the challenges of Application Service Providers (ASPs) is to fulfill the QoS requirements stated in the Service Level Agreements (SLAs) established with different consumers and to minimize the investment and management costs. Cloud computing is the promising solution for ASPs that increasingly demand for an elastic infrastructure. In this paper, we formulate the ASP resource management as an optimization problem and propose both reactive and proactive heuristic policies that approximate the optimal solution. The proposed policies leverage on information about the system performance history and can be applied at runtime because of their reduced computational time. Our experimental results show that some heuristics based on prediction approximate the exact knowledge of the workload.**

I. Introduction

In the today Internet of Services, one of the challenges of Application Service Providers (ASPs) is to fulfill the QoS obligations defined in the Service Level Agreements (SLAs) established with different consumers and to minimize the investment and management costs. To achieve these goals, the ASPs need an infrastructure capable to rapidly scale up in case of peak load, but flexible enough to avoid resources over-provisioning in case of scarce demand. Outsourcing of computational and storage resources is a solution to this problem and the Cloud computing paradigm [1] enables its implementation. Relying on Cloud computing solutions, the ASPs are relieved from the burden of setting up their own data center (totally or in part), and are enabled to instantiate an "elastic" set of resources that dynamically adapt to their needs [2], [3].

Among all the challenges that Cloud computing poses, in this paper we deal with the dynamic QoS provisioning problem. According to the layers of the Cloud stack (IaaS, PaaS, and SaaS), the dynamic QoS provisioning problem can be managed at infrastructure and platform levels (*e.g.,* [4], [5], [6], [7]) or at service level (*e.g.,* [8], [9], [10]). *Service provisioning* (or application-level resource provisioning) is related to mapping applications to application containers or applications directly to virtual machines (VMs) in order to satisfy a SLA agreed with the customers of the Cloud-based applications and to maximize revenues of the service provider offering the applications.

Our work, framed in the context of application-level resource provisioning, investigates policies and mechanisms that an ASP can use to manage the resources offered as a service by an IaaS provider.

Specifically, we propose an autonomic solution to dynamically manage resources taking into account both application QoS objectives and resource exploitation costs. We suppose that an ASP offering a Cloud-based application chooses to lease computational power from an IaaS provider and to size by its own the pool of allocated VMs. Therefore, the ASP has to find out the number of VMs that should be allocated to guarantee the SLA fulfillment at application level, while minimizing the allocation cost it pays for the computational resources over a given time horizon T.

We formulate the optimal VM allocation as a mixed integer linear optimization problem, where the objective function accounts for the VM allocation cost that will be charged in T, subject to the fulfillment of the SLA constraints agreed with the application customers. This approach has two potential drawbacks. First, the problem is inherently NP-hard. Second, in order to compute the optimal solution the system workload should be either

978-0-7695-4550-9/11 $26.00 © 2011 IEEE
DOI 10.1109/NCCA.2011.11

known over the entire interval T or at least estimated over such interval. To address these drawbacks, we also propose and compare proactive and reactive heuristics for resource provisioning which trade-off computational complexity with system efficiency. To evaluate the proposed policies we define a stochastic workload model that reproduces the real access patterns of Web application users. Differently from previous works that consider SLAs defined on average values of some performance metrics, our autonomic resource management approach contemplates an SLA based on the maximum percentage of violations of some Service Level Objective (SLO), *e.g.*, the response time, that the application customers can tolerate. The SLA model we propose is more indicated in a highly changing and unpredictable environment because it contractualizes the probability that SLOs are violated [11].

An increasing number of research have recently proposed solutions to manage dynamically the VM allocation at application level. In [10], Cloud customers are empowered with their own dynamic controller that automatically adds/removes VMs when the CPU utilization exceeds a target range. In [7] the application level is able to access mechanisms and define policies to automatically size the number of instances of containers, thus reducing the application cost. A heuristic approach, based on a QN model, to guide resource allocation decisions is in [9]. The allocation of spot instances has been proposed in [8] to improve the SLA satisfaction; we observe that spot instances are an adequate solution only to provide non-critical Cloud-based applications. With respect to [12], in this paper we propose a completely revised formulation of the service provisioning problem, we improve the workload prediction model, and we provide a comprehensive experimental evaluation using a workload with long range dependencies and bursty arrivals.

Dynamic VM provisioning and management are also provided by commercial solutions, such as Amazon EC2 Auto Scaling and Elastic Beanstalk. The drawback of commercial solutions is that the application service provider is unaware of the used policies and how them can be tuned for its specific needs. Moreover, the service provider can not easily map the SLA requirements negotiated with its users into the SLA agreed with the IaaS provider.

The rest of the paper is organized as follows. In Section II we describe the system architecture of the ASP. In Section III we first define the SLA offered by the ASP to its customers, then present the formulation of the VM allocation as optimization problem. We propose the heuristics policies for VM allocation in Section IV and in section V we analyze the experimental results. We conclude in Section VI.

II. System Architecture

We consider an Application Service Provider (ASP) that offers a service accessible through a Web service interface (e.g., WSDL) or a Web interface (e.g., HTTP). To guarantee the offered service level, the ASP leases computational capacity from an IaaS provider. IaaS providers not only allow to instantiate, on-demand, an arbitrary number of VMs running the same system image, but they also offer a portfolio of services, that range from load distribution and performance monitoring to automatic VM allocation. To achieve a high degree of flexibility in allocating the computational resources minimizing leasing costs on one hand and to comply to customers requirements specified in the SLAs on the other, the components of a Cloud-based system can be organized according to the autonomic loop (e.g., [13], [14], [7]). Therefore, in this paper we suppose the ASP realizes an autonomic architecture in order to fully control the resource management.

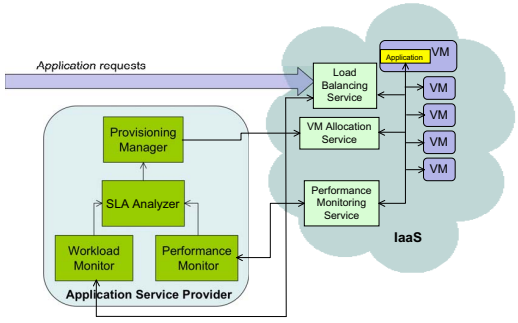


Fig. 1. System architecture of the ASP.

Figure 1 illustrates the autonomic system architecture that we envision for the ASP. On the right side of the figure, there are the IaaS components that provide some services to the ASP. Specifically,

the IaaS *Performance Monitoring Service* collects aggregated performance indexes related to the VMs that the ASP leases (*e.g.*, response time, network traffic, requests count) and provides them to the ASP. The IaaS *Load Balancing Service* distributes the incoming requests among the leased resources, maintaining user sessions. The IaaS *VM Allocation Service* is in charge to manage the allocation and deallocation requests for virtual machines leased by the IaaS provider.

On the left side of Fig. 1 there are the system components implemented and controlled by the ASP. The *Workload Monitor* and *Performance Monitor* periodically estimate and forecast, on the bases of collected historical data, the request arrival rate and the VMs performance level, including the application response time. The *SLA Analyzer* determines whether a new allocation strategy has to be planned because of violations of the SLO agreed with the application customers and, if necessary, triggers the *Provisioning Manager*. The latter is the core component of the envisioned autonomic system, because it plans provisioning decisions to react to (or anticipate future) environmental changes. It uses the estimated and forecasted system state information to parameterize the IaaS system model that is used as input to the provisioning policy to determine whether new VMs should be allocated, running VMs should be deallocated (or let expired), or no action should be taken at all. The Provisioning Manager executes the resource provisioning plan by sending the proper commands to the IaaS VM Allocation Manager. The VM allocation policies are presented in Sections III and IV.

III. VM Allocation Problem

In this section we present the VM allocation problem. We assume time to be logically divided into slots of equal length τ and that VM allocation/deallocation can only occur at slot boundaries. We also assume that the different performance measures, *e.g.*, expected response time, are computed and averaged over a period of time of one slot. Hereafter, for the sake of simplicity, we will assume that all time intervals are integer multiples of a slot.

We consider an ASP that aims to determine a VM allocation schedule over a time horizon of length T. The allocation should minimize the allocation cost paid by the ASP to the IaaS provider, while guaranteeing some SLO defined in the SLA offered by the ASP to its customers. The allocation schedule takes the form of a sequence $z_1, z_2, \ldots, z_i, \ldots$ where z_i is the number of VMs allocated at the beginning of time slot i. Without loss of generality, we assume that a VM can be allocated for a period which is a multiple of time intervals of length W_a (corresponding to $M_{W_a} = W_a/\tau$ time slots), after which it is automatically deallocated (unless implicitly renewed if a new VM is allocated at the same time).

A. SLA Definition

We assume that the SLA offered by the application service provider to its customers is a tuple $\langle R_{max}, W, V_{max} \rangle$ where: R_{max} is the maximum value allowed for the average application response time, *i.e.*, the SLO; W is the length of the SLA time window (we will denote by $M_W = W/\tau$ the corresponding number of slots); V_{max} is the maximum fraction of SLO violations allowed during W ($0 \leq V_{max} \leq 1$). The SLO is violated if $r_i > R_{max}$, where r_i is the expected application response time at time slot i. If we suppose the SLA is evaluated at each slot, the SLA is violated at time slot i if a fraction $V_i > V_{max}$ violations have been observed over the last time window W, *i.e.*, over the last M_W slots $\{i - M_W + 1, \ldots, i\}$, where $V_i = \frac{1}{M_W} \sum_{j=i-M_W+1}^{i} 1\{r_j > R_{max}\}$, $0 \leq V_i \leq 1$.

The ASP periodically monitors the performance of the virtual machines (specifically, their service rate μ) and the request arrival rate λ to the application running in the Cloud through the Workload Monitor and Performance Monitor components, as described in Sec. II. Let us suppose that the ASP system is also instrumented with a model to compute the application response time as a function of μ, λ, and the number x of allocated VMs. Considering that the fraction of SLO violations can be evaluated periodically by measuring how many times the request response time exceeds the R_{max} threshold, we can evaluate the number x of VMs that are needed to satisfy the SLA.

To this end, we model the IaaS system composed by the Load Balancing Service and the set of VMs as a network of $M/M/1$ queues. Under the hypothesis that the flow of incoming requests is

equally distributed among the set of x VMs, the expected application response time r is given by $r = \frac{1}{\mu - \lambda/x}$.

Using this simple model, it is easy to compute the number of VMs such that $r \leq R_{max}$ (in our model, we do not consider the delays introduced by the ASP and the Load Balancing Service).

B. Problem Formulation

We now describe how to determine the optimal virtual machine allocation over a time horizon T, starting from the problem constants and variables.

a) Constants:

- T is the time interval considered by the ASP for the VM allocation problem; M is the corresponding number of slots;
- W_a is the VM allocation time period; M_{W_a} is the corresponding number of slots;
- $\hat{\lambda}_i$ is the request arrival rate expected at time slot $i \in \{1, \ldots, M\}$;
- μ is the service rate of each VM. We assume that the service time is exponentially distributed and that all the VMs have the same performance characteristics and therefore the same service rate;
- $x_{i,\min}$ is the minimum number of VMs needed to guarantee a response time of $R_{\max}$. From the response time formula we readily obtain: $x_{i,\min} = \frac{\hat{\lambda}_i R_{max}}{\mu R_{max} - 1}$.
- $c_i \in \mathbb{R}$ is the cost to use a VM for W_a time slots when the allocation is operated at time slot $i \in [1, M]$. Usually, c_i is constant; however, different billing models where the allocation cost changes over time (*e.g.*, due to energy costs) can be considered.

b) Variables:

- $z_i \in \mathbb{N}$ is the number of VMs to be allocated at the beginning of time slot $i \in \{1, \ldots, M\}$;
- $x_i \in \mathbb{N}$ is the number of VMs available during the time slot $i \in \{1, \ldots, M\}$. We have,

$$x_i = \begin{cases} x_{i-1} + z_i & \text{if } 1 < i < M_{W_a} \\ x_{i-1} + z_i - z_{i-M_{W_a}} & \text{if } i \geq M_{W_a} \end{cases} \quad (1)$$

i.e., the number of VMs available during slot i is equal to: the number of VMs available during the previous slot (x_{i-1}) plus the number of new allocated VMs (z_i) minus the number of VMs whose allocation period just ended ($z_{i-M_{W_a}}$);

- ξ_i is the number of *additional* VMs required to ensure a response time equal to $R_{\max}$ during time slot i;
- $\xi = \max_{i=1,\ldots,M} \xi_i$ is the maximum of the ξ_i over the time horizon T;
- $y_i \in \{0, 1\}$ is a binary variable equal to 1 if there is a SLO violation at slot i, 0 otherwise.

c) Cost and Objective Function: Our goal is to minimize the allocation cost over the interval T. The VM allocation cost is $C(\mathbf{z}) = \sum_{i=1}^{M} c_i z_i$. Nevertheless, as objective function for the VM allocation problem, we consider the following more general objective function: $F(\mathbf{z}, \xi) = \sum_{i=1}^{M} c_i z_i + K\xi$ which allows us to explore the trade-off between allocation cost and user perceived performance, expressed as function of the degree of SLO violations. K is a suitable non-negative constant: if $K = 0$, $F(\mathbf{z}, \xi) = C(\mathbf{z})$; otherwise, $F(\mathbf{z}, \xi)$ is a weighted sum of the allocation cost and the overall level of SLO violation, here simply captured by the maximum number of additional VMs which would be required to ensure the SLO over T.

d) Optimization Problem: Under the assumption that r_i approximates the observed average response time of the set of x_i VMs, the optimal VM allocation $\mathbf{z} = (z_1, z_2, ..., z_M)$, which minimizes the objective function $F(\mathbf{z}, \xi)$ while guaranteeing the fulfillment of the SLA $\langle R_{max}, W, V_{max} \rangle$, is given by the solution of the following optimization problem:

$$\textbf{min } F(\mathbf{z}, \xi) = \sum_{i=1}^{M} c_i z_i + K\xi$$

subject to:

$$x_i = x_{i-1} + z_i, \ 1 < i < M_{W_a} \quad (2)$$

$$x_i = x_{i-1} + z_i - z_{i-M_{W_a}}, \ i \geq M_{W_a} \quad (3)$$

$$x_i + \xi_i \geq x_{i,\min}, \ i \in \{1, \ldots, M\} \quad (4)$$

$$\xi_i \leq y_i B, \ i \in \{1, \ldots, M\} \quad (5)$$

$$\xi_i \leq \xi, \ i \in \{1, \ldots, M\} \quad (6)$$

$$\frac{1}{M_W} \sum_{j=i-M_W+1}^{i} y_j \leq V_{max}, \ i \in \{M_W, \ldots, M\} \quad (7)$$

$$x_i \leq X_{max}, i \in \{1, \ldots, M\} \quad (8)$$

$$z_i, \ \xi_i, \ \xi \geq 0, i \in \{1, \ldots, M\} \quad (9)$$

$$x_i, z_i \in \mathbb{N}, i \in \{1, \ldots, M\} \quad (10)$$

(2)-(3) are just Eqs. (1) which relate the number of available VMs x_i with the number of allocated

VMs z_i. Inequalities (4) are the SLO constraints: if $x_i \geq x_{i,\min}$, the SLO is satisfied and $\xi = 0$; if, instead, $x_i \leq x_{i,\min}$, there is a SLO violation and ξ is the number of additional VMs which would ensure a response time equal to $R_{\max}$. Inequalities (5) ensure that $y_i = 1$ whenever $\xi_i > 0$, *i.e.*, when there is a SLO violation, being B a large constant. (7) is the SLA constraint on the maximum number of SLO violations; the LHS corresponds to V_i, the number of SLO violations in $[i - M_W + 1, i]$. Finally, Eqs. 8 are the functional constraints: here we assume that the maximum number of VMs that can be allocated at any given time cannot exceed a given constant X_{max}. We observe that, while we included them in the constraints, it is not necessary to consider the integrality constraints for the variables z_i, which are implicitly enforced by the constraints (2)-(3) and the integrality constraints on the variables x_i.

The proposed optimization problem is a Mixed Integer Linear Programming (MILP) problem which can be solved via standard techniques. Since the complexity is exponential in the number of integer variables, the computational cost might turn to be prohibitive for online operations, unless we consider a shorter time interval T and/or a coarser time granularity, *i.e.*, a large τ. In the next section we present some heuristics for the VM allocation problem.

IV. Heuristic VM Allocation

In this paper, we propose both *reactive* and *proactive* heuristic policies for the VM allocation. Irrespectively from the policy, the heuristic VM allocation process is in Algorithm 1, where $\texttt{reqVM}_i$ is the number of required VMs and $\texttt{expVM}_{i-1}$ is the number of VMs whose billing period expires at the end of the current time slot.

In this paper, we propose the following heuristic allocation policies.

Exact knowledge (EK) assumes that the average arrival rate λ_i in the upcoming time slot i is known exactly at the beginning of that time slot and that the arrivals are uniformly distributed within the single time slots. The minimum number of VMs needed to meet the SLO in every slot is given by $x_{i,\min}$, defined in Section III. To prevent workload fluctuations that will exceed λ_i and therefore that could result in SLO violations, this policy overestimates the arrival rate as follows: $\hat{\lambda}_i = (1 + \alpha)\lambda_i$, where $\alpha > 0$.

Reactive 1 step early (r-1) measures the average arrival rate λ_{i-1} observed in the previous time slot $i - 1$ and sizes the set of VMs assuming that the estimated arrival rate in the following time slot will be $\hat{\lambda}_i = (1 + \alpha)\lambda_{i-1}$, where $\alpha > 0$. As in the *EK* policy, the number of needed resources in every slot is given by $x_{i,\min}$, taking into account only the constraint on the maximum response time R_{max}.

Proactive 1 step ahead, Y% (p1-Y), our proactive policy that uses a prediction algorithm based on Recursive Least Square (RLS) to forecast the workload in the next time slot of length τ. Specifically, the prediction algorithm uses an autoregressive process of order 2 $AR(2)$, whose weights and white noise process variance are continuously estimated using the RLS method [15]. In particular, the *p1-Y* policy predicts the one step ahead $Y\%$ upper bound of the request arrival rate using the RLS-based prediction. The number of needed VMs is given by $x_{i,\min}$, assuming that the estimated arrival rate is given by $\hat{\lambda}_i = \widetilde{\lambda}_i + a(Y)\hat{\sigma}^2(i)$ where: $\widetilde{\lambda}_i$ is the forecasted average request arrival rate, $\hat{\sigma}^2(i)$ is the white noise variance of the process and $a(Y)$ is equal either to 1.96 for the 95-percentile upper bound ($Y = 0.95$) or to 2.56 for the 99-percentile upper bound ($Y = 0.99$). Since the prediction is carried out only one step ahead, this policy is not capable to prevent SLA violations but only to guarantee that the SLO is honored in the current time slot.

Algorithm 1 Heuristic VM allocation

1: $\texttt{reqVM}_i$ =`heuristicAlg(log)`; {`heuristicAlg(log)` estimates the number of VMs needed to avoid SLO violations in the time slot i using one of the heuristic policies}
2: **if** $\texttt{reqVM}_i > x_{i-1}$ **then**
3: $x_i = \texttt{reqVM}_i$;
4: **else if** $\texttt{reqVM}_i < x_{i-1}$ **then**
5: $x_i = min\{(x_{i-1} - \texttt{reqVM}_i), \texttt{expVM}_{i-1}\}$; {$\texttt{expVM}_{i-1}$ = number of VMs whose billing period expires at the end of time slot $i-1$ }
6: **end if**

V. Experimental Results

To evaluate the performance of the proposed heuristic policies versus the optimal VM allocation we proceed as follows. First, we define a stochastic workload model that presents long range dependencies and bursty behavior. Second, we define an appropriate set of performance metrics to evaluate and compare the proposed policies. Finally, we present a large set of simulation experiments to evaluate the sensitivity of the heuristic policies to their tunable parameters and to compare their performance. The setting of the main parameters used in the simulation experiments is shown in Tab. I.

TABLE I
VALUE SETTING FOR THE MAIN SYSTEM MODEL PARAMETERS

Parameter	Notation	Value
Time horizon	T	168 hours
Length of VM allocation time	W_a	1 hour
Time slot	τ	5 min.
Service rate of each VM	μ	10 req/sec
Allocation cost	c_i	0.1 $
Maximum response time	R_{max}	0.5 sec
Maximum fraction of SLO violations	V_{max}	$\{0.1, 0.25, 0.35\}$
Length of SLA time window	W	$\{0.5W_a, W_a, 2W_a\}$
Weight of the overall level of SLO violations	K	0 - 10^5

To evaluate our policies we consider a stochastic workload model that reproduces the time dependency and bursty characteristics of real workloads. Specifically, using standard techniques, we first model an excerpt of the 1998 FIFA World Cup traces [16] with a discrete time Markov chain (DTMC). Then, through the obtained model, we generate a synthetic trace that has the same stochastic properties of the original trace. To evaluate the heuristic policies we used 20 instances of a long workload lasting 168 hours (Fig. 2). The 20 workload instances are obtained by changing the random number sequences used to generate the stochastic variables. We evaluate the optimal and heuristic policies against the following metrics.

- The overall *allocation cost* C (measured in $ and over T) defined in Sec. III-B. In our experiments, we observed that the randomly generated workload can have a highly variable intensity (see Fig. 2) that results in VM allocation costs with a high variance. Therefore, to take the worst cases into account we consider the 95-percentile of C.
- The *SLA satisfaction factor (SSF)* over the time horizon T defined as: $SSF = 1 - Pr\{V_i > V_{max}\}$, $0 \leq SSF \leq 1$, where $Pr\{V_i > V_{max}\} = \frac{1}{M - M_{W_a} + 1} \sum_{i=M_{W_a}+1}^{M} 1\{V_i > V_{max}\}$ and V_i is the fraction of SLO violations defined in Sec. III-A.
- The *fraction of SLO violations* V_i at time slot i. The latter metric allows us to evaluate the behavior of the optimal allocation policy under different combinations of tuning parameters. While the optimal allocation policy achieves $SSF = 1$ by definition, the number of SLO violations can range from 0 to V_{max} with a non negligible impact on the allocation cost.

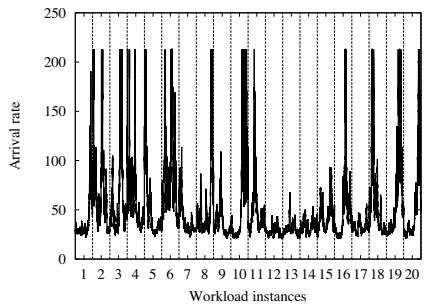


Fig. 2. The 20 instances of workload used in the experiments: vertical dotted lines separate the runs, being the length of each run 168 hours.

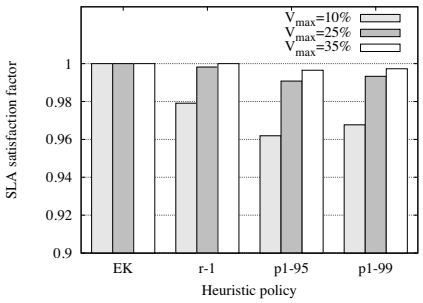
To evaluate the behavior of the heuristic policies we set: $T = 168$ h, $R_{max} = 0.5$ sec, $W_a = 1$ h, $W/W_a = \{0.5, 1, 2\}$, and $V_{max} \in \{0.1, 0.25, 0.35\}$.

We first analyze the sensitivity of the *EK* and *r-1* heuristics to the α parameter. Table II shows the corresponding results. The *EK* strategy, that assumes to know exactly the workload at time slot i, obtains no violation at the same cost of the *r-1* policy, because the same values of the α parameter are used to over-estimate the request arrival rate. For *r-1*, $\alpha = 0.1$ turns out to be the best setting because it allows to achieve the lowest cost-performance ratio, i.e., $C/(1 - Pr\{V > V_{max}\})$; therefore, we will use it for the comparison.

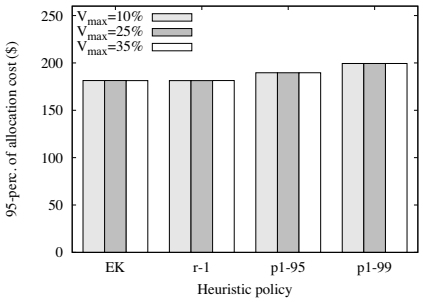
TABLE II
EK AND *r-1* HEURISTICS: SENSITIVITY TO α

Heu.	α	C	SLA satisfaction factor		
		95-%	V_{max}=.1	V_{max}=.25	V_{max}=.35
	0.1	181.4	0.9791	0.9982	1
r-1	0.2	196.55	0.962	1	1
	0.3	212.47	0.9984	1	1
	0.1	181.31	1	1	1
EK	0.2	196.51	1	1	1
	0.3	212.34	1	1	1

After the choice of values of the tuning parameters, we compare the different heuristic allocation policies, as shown in Fig. 3.



(a) SLA satisfaction factor



(b) Allocation cost

Fig. 3. Comparison of heuristics.

The *EK* policy, that at the beginning of each time slot assumes to knows the average arrival rate for that slot, outperforms all the other heuristics in terms of SLA satisfaction and allocation cost. The *r-1* is the second best allocation policy offering, in the worst case, a SLA satisfaction factor of the 98% at the same cost of the *EK*. A not expected result is that the proactive heuristics (*p1-95* and *p1-99*) suffer in predicting both the arrival and termination of bursts. When the workload is characterized by intense fluctuations, the slow convergence of the prediction algorithm first results in SLO violations (when the burst of requests arrives) and then in resource over-provisioning (when the burst terminates). This behavior determines a lower SLA satisfaction factor and a higher allocation cost with respect to the reactive heuristic policy *r-1*.

VI. CONCLUSIONS

In this paper, we proposed an autonomic solution that enables service providers offering a Cloud-based application to handle the dynamic resource provisioning at application level by taking into account both application QoS objectives and resource exploitation costs.

The experimental evaluation gives some useful insight into the optimal allocation policy and the heuristics. First, we observed that the performance of the optimal policy depends on the setting of the parameters K, W, and W_a. Second, we observed that the behavior of the *r-1* and *p1-Y* heuristics is comparable in terms of SLA satisfaction, while the *r-1* heuristic outperform the other policies in term of allocation cost. The lessons learned is that while the system behavior can be approximated with a simple model, the *r-1* resource management strategy is the most appropriate choice that allows to obtain, in the worst case, a SLA satisfaction factor only two percent points less than the Exact Knowledge solution.

In our future work, we plan to extend the system model and to consider more non functional requirements in the SLA definition.

REFERENCES

[1] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Gener. Comput. Syst.*, vol. 25, no. 6, pp. 599–616, 2009.

[2] R. Moreno-Vozmediano, R. S. Montero, and I. M. Llorente, "Elastic management of cluster-based services in the cloud," in *Proc. ACDC '09*. ACM, 2009, pp. 19–24.

[3] T. Dornemann, E. Juhnke, and B. Freisleben, "On-demand resource provisioning for BPEL workflows using Amazon's Elastic Compute Cloud," in *Proc. CCGRID '09*, 2009.

[4] Y. Song, H. Wang, Y. Li, B. Feng, and Y. Sun, "Multi-tiered on-demand resource scheduling for vm-based data center," in *Proc. CCGRID '09*, 2009, pp. 148–155.

[5] E. Kalyvianaki, T. Charalambous, and S. Hand, "Self-adaptive and self-configured cpu resource provisioning for virtualized servers using kalman filters," in *Proc. ICAC '09*. ACM, 2009.

[6] H. Nguyen Van, F. Dang Tran, and J.-M. Menaud, "SLA-aware virtual resource management for Cloud infrastructures," in *Proc. of CIT '09*, 2009.

[7] M. Litoiu, M. Woodside, J. Wong, J. Ng, and G. Iszlai, "A business driven cloud optimization architecture," in *Proc. of 2010 ACM Symposium on Applied Computing*, 2010.

[8] A. Andrzejak, D. Kondo, and S. Yi, "Decision model for cloud computing under SLA constraints," in *Proc. MASCOTS '10*, 2010, pp. 257–266.

[9] Y. Hu, J. Wong, G. Iszlai, and M. Litoiu, "Resource provisioning for cloud computing," in *Proc. CASCON '09*. ACM, 2009.

[10] H. C. Lim, S. Babu, J. S. Chase, and S. S. Parekh, "Automated control in cloud computing: challenges and opportunities," in *Proc. ACDC '09*. ACM, 2009, pp. 13–18.

[11] R. Powers, M. Goldszmidt, and I. Cohen, "Short term performance forecasting in enterprise systems," in *Proc. KDD '05*. ACM, 2005, pp. 801–807.

[12] E. Casalicchio and L. Silvestri, "Medium/long term SLA provisioning in cloud-based service providers," in *Proc. of 2nd ICST Int'l Conf. on Cloud Computing*, 2010.

[13] I. Brandic, "Towards self-manageable cloud services," in *Proc. COMPSAC '09*, vol. 2, 2009, pp. 128–133.

[14] S. Ferretti, V. Ghini, F. Panzieri, M. Pellegrini, and E. Turrini, "Qos-aware clouds," in *Proc. of 3rd IEEE Int'l Conf. on Cloud Computing*, 2010, pp. 321–328.

[15] S. Haykin, *Adaptive Filter Theory*, 3rd ed. Prentice-Hall, 1996.

[16] M. Arlitt and T. Jin, "A workload characterization study of the 1998 world cup web site," *IEEE Network*, vol. 14, no. 3, pp. 30 –37, May 2000.