A FRAMEWORK FOR QOS-AWARE EXECUTION OF WORKFLOWS OVER THE CLOUD

Moreno Marzolla[1] and Raffaela Mirandola[2]
[1]*Università di Bologna, Dipartimento di Scienze dell'Informazione, Mura A. Zamboni 7, I-40127 Bologna, Italy*
[2]*Politecnico di Milano, Dipartimento di Elettronica e Informazione, Piazza Leonardo da Vinci, I-20133 Milano, Italy*

Keywords: Cloud Computing, Workflow Engine, Self-management, SLA Management.

Abstract: The Cloud Computing paradigm is providing system architects with a new powerful tool for building scalable applications. Clouds allow allocation of resources on a "pay-as-you-go" model, so that additional resources can be requested during peak loads and released after that. In this paper we describe SAVER (qoS-Aware workflows oVER the Cloud), a QoS-aware algorithm for executing workflows involving Web Services hosted in a Cloud environment. SAVER allows execution of arbitrary workflows subject to response time constraints. SAVER uses a simple Queueing Network (QN) model to identify the optimal resource allocation; specifically, the QN model is used to identify bottlenecks, and predict the system performance as Cloud resources are allocated or released. Our approach has been validated through numerical simulations, whose results are reported in this paper.

1 INTRODUCTION

The emerging Cloud computing paradigm is rapidly gaining consensus as an alternative to traditional IT systems. Informally, Cloud computing allows computing resources to be seen as a utility, available on demand. Cloud services can be grouped into three categories (Zhang et al., 2010): Infrastructure as a Service (IaaS), providing low-level resources such as Virtual Machines (VMs); Platform as a Service (PaaS), providing software development frameworks; and Software as a Service (SaaS), providing whole applications. The Cloud provider has the responsibility to manage the resources it offers so that the user requirements and the desired Quality of Service (QoS) are satisfied.

In this paper we present SAVER (qoS-Aware workflows oVER the Cloud), a workflow engine provided as a SaaS. The engine allows different types of workflows to be executed over a set of Web Services (WSs). In our scenario, users negotiate QoS requirements with the service provider; specifically, for each type c of workflow, the user may request that the average execution time of the whole workflow should not exceed a threshold R_c^+. Once the QoS requirements have been negotiated, the user can submit any number of workflows of the different types. Both the submission rate and the time spent by the workflows on each WS can fluctuate over time.

SAVER uses an underlying IaaS Cloud to provide computational power on demand. The Cloud hosts multiple instances of each WS, over which the workload can be balanced. If a WS is heavily used, SAVER will increase the number of instances by requesting new resources from the Cloud. System reconfigurations are triggered periodically, when instances are added or removed where necessary.

SAVER uses a open, multiclass Queueing Network (QN) model to predict the response time of a given Cloud resource allocation. The parameters which are needed to evaluate the QN model are obtained by passively monitoring the running system. The performance model is used within a greedy strategy which identifies an approximate solution to the optimization problem minimizing the number of WS instances while respecting the Service Level Agreement (SLA).

The remainder of this paper is organized as follows. In Section 2 we give a precise formulation of the problem we are addressing. In Section 3 we describe the Queueing Network performance model of the Cloud-based workflow engine. SAVER will be fully described in Section 4, including the high-level architecture and the details of the reconfiguration algorithms. The effectiveness of SAVER have been evaluated using simulation experiments, whose

Marzolla M. and Mirandola R..
A FRAMEWORK FOR QOS-AWARE EXECUTION OF WORKFLOWS OVER THE CLOUD.
DOI: 10.5220/0003898702160221
In *Proceedings of the 2nd International Conference on Cloud Computing and Services Science* (CLOSER-2012), pages 216-221
ISBN: 978-989-8565-05-1

results will be discussed in Section 5. In Section 6 we review the scientific literature and compare SAVER with related works. Finally, conclusions and future works are presented in Section 7.

2 PROBLEM FORMULATION

SAVER is a workflow engine that receives workflows from external clients, and executes them over a set of K WS $1,\ldots,K$. At any given time t, there can be $C(t)$ different workflow types (or classes); for each class $c = 1,\ldots,C(t)$, clients define a maximum allowed completion time R_c^+. The number of workflow classes $C(t)$ does not need to be known in advance; furthermore, it is possible to add or remove classes at any time.

We denote with $\lambda_c(t)$ the average arrival rate of class c workflows at time t. Since all WSs are shared between the workflows, the completion time depends both on arrival rates $\lambda(t) = \big(\lambda_1(t),\ldots,\lambda_{C(t)}(t)\big)$, and on the utilization of each WS.

To satisfy the response time constraints, the system must adapt to cope with workload fluctuations. To do so, SAVER relies on a IaaS Cloud which maintains multiple instances of each WS. We denote with N_k the number of instances of WS k; a system configuration $\mathbf{N}(t) = (N_1(t),\ldots,N_K(t))$ is an integer vector representing the number of allocated instances of each WS. When the workload intensity increases, additional instances are created to eliminate the bottlenecks; when the workload decreases, surplus instances are shut down and released.

The goal of SAVER is to minimize the total number of WS instances while maintaining the mean execution time of type c workflows below the threshold R_c^+, $c = 1,\ldots,C(t)$. Formally, we want to solve the following optimization problem:

$$\begin{aligned} \text{minimize} \quad & f(\mathbf{N}(t)) = \sum_{k=1}^{K} N_k(t) && (1)\\ \text{subject to} \quad & R_c(\mathbf{N}(t)) \le R_c^+ \quad \text{for all } c = 1,\ldots,C(t) \\ & N_i(t) \in \{1,2,3,\ldots\} \end{aligned}$$

where $R_c(\mathbf{N}(t))$ is the mean execution time of type c workflows when the system configuration is $\mathbf{N}(t)$.

3 PERFORMANCE MODEL

In this section we describe the QN performance model which is used to plan a system reconfiguration. We model the system using the open, multiclass QN model (Lazowska et al., 1984) shown in Fig. 1. Each server represents a single WS instance; thus, WS k is represented by N_k queueing centers, for each $k = 1,\ldots,K$. N_k can change over time, as resources are added or removed from the system.

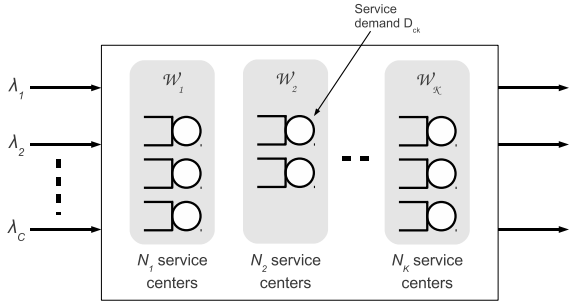


Figure 1: Performance model based on an open, multiclass Queueing Network.

In our QN model there are C different classes of requests: each request represents a workflow, thus workflow types are directly mapped to QN request classes. In order to simplify the analysis of the model, we assume that the inter-arrival time of class c requests is exponentially distributed with arrival rate λ_c.

The interaction of a type c workflow with WS k is modeled as a visit of a class c request to one of the N_k queueing centers representing WS k. We denote with $R_{ck}(\mathbf{N})$ the total time (*residence time*) spent by type c workflows on one of the N_k instances of WS k for a given configuration $\mathbf{N}$. The residence time is the sum of two terms: the *service demand* $D_{ck}(\mathbf{N})$ (average time spent by a WS instance executing the request) and queueing delay (time spent by a request in the waiting queue).

The *utilization* $U_k(\mathbf{N})$ of an instance of WS k is the fraction of time the instance is busy processing requests. If the workload is evenly balanced, then both the residence time $R_{ck}(\mathbf{N})$ and the utilization $U_k(\mathbf{N})$ are almost the same for all N_k instances.

4 SAVER ARCHITECTURE

SAVER is a reactive system based on the Monitor-Analyze-Plan-Execute (MAPE) control loop shown in Fig. 2. During the *Monitor* step, SAVER collects operational parameters by observing the running system. The parameters are evaluate during the *Analyze* step; if the system needs to be reconfigured (e.g., because the observed response time of class c workflows exceeds the threshold R_c^+, for some c), a new configuration is identified in the *Plan* step. We use the QN model described in Section 3 to evaluate different configurations and identify an optimal server allocation such that all QoS constraints are satisfied.

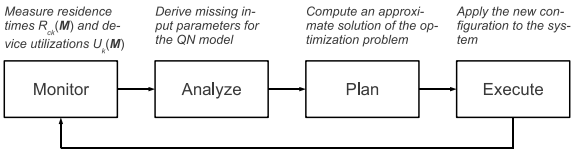


Figure 2: SAVER Control Loop.

$$U_k(\mathbf{N}) = \sum_{c=1}^{C} \lambda_c D_{ck}(\mathbf{N}) \quad (2)$$

$$R_{ck}(\mathbf{N}) = \frac{D_{ck}(\mathbf{N})}{1-U_k(\mathbf{N})} \quad (3)$$

$$R_c(\mathbf{N}) = \sum_{k=1}^{K} N_k R_{ck}(\mathbf{N}) \quad (4)$$

Figure 3: Equations for the QN model of Fig. 1.

Finally, during the *Execute* step, the new configuration is applied to the system: WS instances are created or destroyed as needed by leveraging the IaaS Cloud. Unlike other reactive systems, SAVER can plan complex reconfigurations, involving multiple additions/removals of resources, in a single step.

4.1 Monitoring System Parameters

The QN model is used to estimate the execution time of workflow types for different system configurations. To analyze the QN it is necessary to know two parameters: (*i*) the current arrival rate of type c workflows, λ_c, and (*ii*) the mean service demand $D_{ck}(\mathbf{M})$ of type c workflows on an instance of WS k, for the current configuration $\mathbf{M}$.

The parameters above can be computed by monitoring the system over a time interval of suitable duration T. The arrival rates λ_c can be estimated by counting the number A_c or arrivals of type c workflows which are submitted over the observation period; then λ_c can be defined as $\lambda_c = A_c/T$.

Measuring the service demands $D_{ck}(\mathbf{M})$ is more difficult because they must not include the time spent by a request waiting to start service. If the WSs do not provide detailed timing information (e.g., via their execution logs), it is possible to estimate $D_{ck}(\mathbf{M})$ from the measured residence time $R_{ck}(\mathbf{M})$ and utilization $U_k(\mathbf{M})$. We use the equations shown in Figure 3, which hold for the open multiclass QN model in Fig. 1. These equations describe known properties of open QN models (see (Lazowska et al., 1984) for details).

The residence time is the total time spent by a type c workflow with one instance of WS k, including waiting time and service time. The workflow engine can measure $R_{ck}(\mathbf{M})$ as the time elapsed from the instant a type c workflow sends a request to one of the N_k instances of WS k, to the time the request is completed. The utilization $U_k(\mathbf{M})$ of an instance of WS k can be obtained by the Cloud service dashboard (or measured on the computing nodes themselves). Using (3) the service demands can be expressed as

$$D_{ck}(\mathbf{M}) = R_{ck}(\mathbf{M})\,(1-U_k(\mathbf{M})) \quad (5)$$

Let $\mathbf{M}$ be the current system configuration; let us assume that, under configuration $\mathbf{M}$, the observed arrival rates are $\lambda = (\lambda_1, \ldots, \lambda_C)$ and service demands are $D_{ck}(\mathbf{M})$. Then, for an arbitrary configuration $\mathbf{N}$, we can combine Equations (3) and (4) to get:

$$R_c(\mathbf{N}) = \sum_{k=1}^{K} N_k \frac{D_{ck}(\mathbf{N})}{1-U_k(\mathbf{N})} \quad (6)$$

The current *total* class c service demand on all instances of WS k is $M_k D_{ck}(\mathbf{M})$, hence we can express service demands and utilizations of individual instances for an arbitrary configuration $\mathbf{N}$ as:

$$D_{ck}(\mathbf{N}) = \frac{M_k}{N_k} D_{ck}(\mathbf{M}) \quad (7)$$

$$U_k(\mathbf{N}) = \frac{M_k}{N_k} U_k(\mathbf{M}) \quad (8)$$

Thus, we can rewrite (6) as

$$R_c(\mathbf{N}) = \sum_{k=1}^{K} \frac{D_{ck}(\mathbf{M}) M_k N_k}{N_k - U_k(\mathbf{M}) M_k} \quad (9)$$

which allows us to estimate the response time $R_c(\mathbf{N})$ of class c workflows for any configuration $\mathbf{N}$, given information collected by the monitor for the current configuration $\mathbf{M}$.

4.2 Finding a New Configuration

In order to find an approximate solution to the optimization problem (1), SAVER starts from the current configuration $\mathbf{M}$, which may violate some response time constraints, and executes Algorithm 1. After collecting device utilizations, response times and arrival rates, SAVER estimates the service demands D_{ck} using Eq. (5).

Then, SAVER identifies a new configuration $\mathbf{N} \succ \mathbf{M}$[1] by calling the function ACQUIRE() (Algorithm 2). The new configuration $\mathbf{N}$ is computed by greedily adding new instances to bottleneck WSs. The QN model is used to estimate response times as instances are added: no actual resources are instantiated from

[1] $\mathbf{N} \succ \mathbf{M}$ iff $N_k \geq M_k$ for all $k = 1, \ldots, K$, the inequality being strict for at least one value of k

Algorithm 1: The SAVER Algorithm.

Require: R_c^+: Maximum response time of type c workflows
1: Let $\mathbf{M}$ be the initial configuration
2: **loop**
3: Monitor $R_{ck}(\mathbf{M})$, $U_k(\mathbf{M})$, λ_c
4: **for all** $c := 1,\ldots,C$; $k := 1,\ldots,K$ **do**
5: Compute $D_{ck}(\mathbf{M})$ using Eq. (5)
6: $\mathbf{N} := \mathsf{Acquire}(\mathbf{M},\lambda,\mathbf{D}(\mathbf{M}),\mathbf{U}(\mathbf{M}))$
7: **for all** $c := 1,\ldots,C$; $k := 1,\ldots,K$ **do**
8: Compute $D_{ck}(\mathbf{N})$ and $U_k(\mathbf{N})$ using Eq. (7) and (8)
9: $\mathbf{N}' := \mathsf{Release}(\mathbf{N},\lambda,\mathbf{D}(\mathbf{N}),\mathbf{U}(\mathbf{N}))$
10: Apply the new configuration $\mathbf{N}'$ to the system
11: $\mathbf{M} := \mathbf{N}'$ {Set $\mathbf{N}'$ as the current configuration $\mathbf{M}$}

Algorithm 2: $\mathsf{Acquire}(\mathbf{N},\lambda,\mathbf{D}(\mathbf{N}),\mathbf{U}(\mathbf{N})) \to \mathbf{N}'$.

Require: $\mathbf{N}$ System configuration
Require: λ Current arrival rates of workflows
Require: $\mathbf{D}(\mathbf{N})$ Service demands at configuration $\mathbf{N}$
Require: $\mathbf{U}(\mathbf{N})$ Utilizations at configuration $\mathbf{N}$
Ensure: $\mathbf{N}$ New system configuration
1: **while** $\big(R_c(\mathbf{N}) > R_c^+$ for any $c\big)$ **do**
2: $b := \arg\max_c \left\{ \dfrac{R_c(\mathbf{N}) - R_c^+}{R_c^+} \;\middle|\; c = 1,\ldots,C \right\}$
3: $j := \arg\max_k \{ R_b(\mathbf{N}) - R_b(\mathbf{N}+\mathbf{1}_k) \mid k = 1,\ldots,K \}$
4: $\mathbf{N} := \mathbf{N} + \mathbf{1}_j$
5: **Return** $\mathbf{N}$

the Cloud service at this time.

The configuration $\mathbf{N}$ returned by the function ACQUIRE() does not violate any constraint, but might contain too many WS instances. Thus, SAVER invokes the function RELEASE() (Algorithm 3) which computes another configuration $\mathbf{N}' \prec \mathbf{N}$ by removing redundant instances, ensuring that no constraint is violated. To call procedure RELEASE() we need to estimate the service demands $D_{ck}(\mathbf{N})$ and utilizations $U_k(\mathbf{N})$ with configuration $\mathbf{N}$. These can be easily computed from the measured values for the current configuration $\mathbf{M}$.

After both steps above, $\mathbf{N}'$ becomes the new current configuration: WS instances are created or terminated where necessary by acquiring or releasing hosts from the Cloud infrastructure. See (Marzolla and Mirandola, 2011) for further details.

5 NUMERICAL RESULTS

We performed a set of numerical simulation experiments to assess the effectiveness of SAVER. We consider all combinations of $C \in \{10,15,20\}$ workflow types and $K \in \{20,40,60\}$ Web Services. Service demands D_{ck} have been randomly generated, in such a

Algorithm 3: $\mathsf{Release}(\mathbf{N},\lambda,\mathbf{D}(\mathbf{N}),\mathbf{U}(\mathbf{N})) \to \mathbf{N}'$.

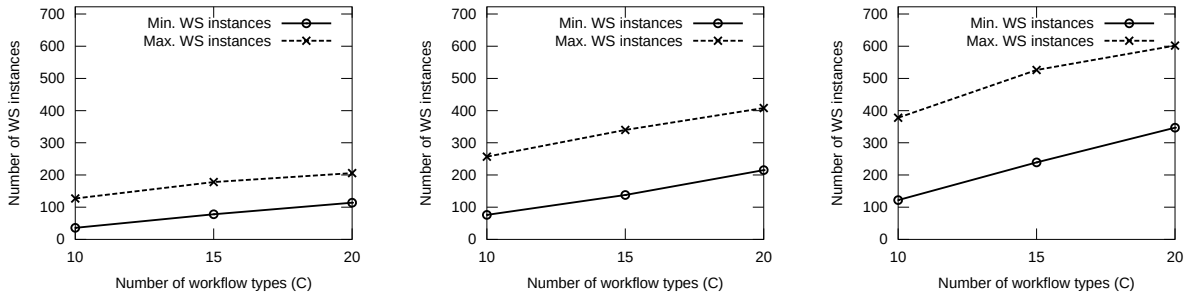
Require: $\mathbf{N}$ System configuration
Require: λ Current arrival rates of workflows
Require: $\mathbf{D}(\mathbf{N})$ Service demands at configuration $\mathbf{N}$
Require: $\mathbf{U}(\mathbf{N})$ Utilizations at configuration $\mathbf{N}$
Ensure: $\mathbf{N}'$ New system configuration
1: **for all** $k := 1,\ldots,K$ **do**
2: $Nmin_k := N_k \sum_{c=1}^{C} \lambda_c D_{ck}(\mathbf{N})$
3: $S := \{k \mid N_k > Nmin_k\}$
4: **while** $(S \neq \emptyset)$ **do**
5: $d := \arg\min_c \left\{ \dfrac{R_c^+ - R_c(\mathbf{N})}{R_c^+} \;\middle|\; c = 1,\ldots,C \right\}$
6: $j := \arg\min_k \left\{ R_c(\mathbf{N}-\mathbf{1}_k) - R_c^+ \;\middle|\; k \in S \right\}$
7: **if** $\big(R_c(\mathbf{N}-\mathbf{1}_j) > R_c^+$ for any $c\big)$ **then**
8: $S := S \setminus \{j\}$ {No instance of WS j can be removed}
9: **else**
10: $\mathbf{N} := \mathbf{N} - \mathbf{1}_j$
11: **if** $\big(N_j = Nmin_j\big)$ **then**
12: $S := S \setminus \{j\}$
13: **Return** $\mathbf{N}$

way that class c workflows have service demands which are uniformly distributed in $[0, c/C]$. Thus, class 1 workflows have lowest average service demands, while type C workflows have highest demands. The system has been simulated for $T = 200$ discrete steps $t = 1,\ldots,T$. Arrival rates $\lambda(t)$ at step t have been generated according to a fractal model, starting from a randomly perturbed sinusoidal pattern to mimic periodic fluctuations; each workflow type has a different period.

The minimum and the maximum number of resources allocated by SAVER during the simulation run are illustrated in Figure 4. Figure 5 shows the total number of WS instances allocated over the whole simulation run by SAVER (labeled *Dynamic*) with the number of instances statically allocated by overprovisioning for the worst-case scenario (labeled *Static*). The results show that SAVER allocates between 64%–72% of the instances required by the worst-case scenario. As previously observed, if the IaaS provider charges a fixed price for each instance allocated at each simulation step, then SAVER allows a consistent reduction of the total cost, while still maintaining the negotiated SLA.

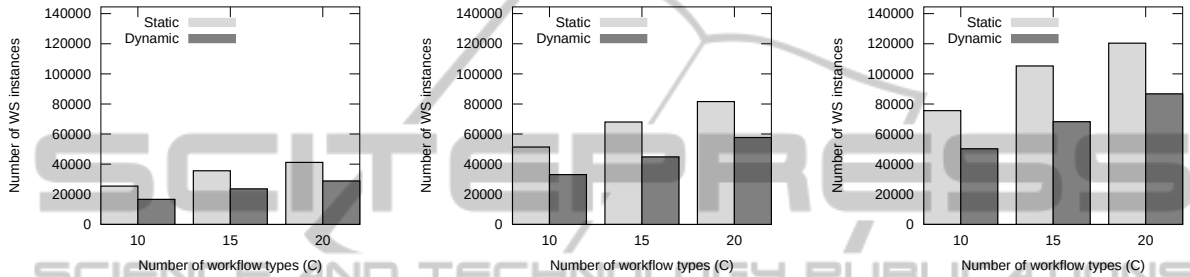
6 RELATED WORKS

Several research contributions have previously addressed the issue of optimizing the resource allocation in cluster-based service centers; some of them



(a) $K = 20$ (b) $K = 40$ (c) $K = 60$

Figure 4: Minimum and maximum number of WS instances allocated during the simulation runs.



(a) $K = 20$ (b) $K = 40$ (c) $K = 60$

Figure 5: Total number of WS instances allocated during the whole simulation runs (lower is better).

use control theory-based feedback loops (Litoiu et al., 2010; Kalyvianaki et al., 2009), machine learning techniques (Kephart et al., 2007; Calinescu, 2009), or utility-based optimization techniques (Urgaonkar et al., 2007; Zhu et al., 2009). When moving to virtualized environments the resource allocation problem becomes more complex because of the introduction of virtual resources (Zhu et al., 2009). Several approaches have been proposed for QoS and resource management at run-time (Li et al., 2009; Litoiu et al., 2010; Ferretti et al., 2010; Jung et al., 2010; Huber et al., 2011; Yazir et al., 2010).

(Li et al., 2009) describes a method for achieving optimization in Clouds by using performance models all along the development and operation of the applications running in the Cloud. The proposed optimization aims at maximizing profits by guaranteeing the QoS agreed in the SLAs taking into account a large variety of workloads. A layered Cloud architecture taking into account different stakeholders is presented in (Litoiu et al., 2010). The architecture supports self-management based on adaptive feedback control loops, present at each layer, and on a coordination activity between the different loops. Mistral (Jung et al., 2010) is a resource managing framework with a multi-level resource allocation algorithm considering reallocation actions based mainly on adding, removing and/or migrating virtual machines, and shutdown or restart of hosts. This approach is based on the usage of Layered Queuing Network (LQN) performance model. It tries to maximize the overall utility taking into account several aspects like power consumption, performance and transient costs in its reconfiguration process. In (Huber et al., 2011) the authors present an approach to self-adaptive resource allocation in virtualized environments based on online architecture-level performance models. The online performance prediction allow estimation of the effects of changes in user workloads and of possible reconfiguration actions. Yazir *et al.* (Yazir et al., 2010) introduces a distributed approach for dynamic autonomous resource management in computing Clouds, performing resource configuration using through Multiple Criteria Decision Analysis.

With respect to these works, SAVER lies in the research line fostering the usage of models at runtime to drive the QoS-based system adaptation. SAVER uses an efficient modeling technique that can then be used at runtime without undermining the system behavior and its overall performance.

Ferretti *et al.* (Ferretti et al., 2010) describe a middleware architecture enabling a SLA-driven dynamic configuration, management and optimization of Cloud resources and services. The approach is purely reactive and considers a single-tier application, while SAVER works for an arbitrary number of WSs and uses a performance model to plan complex reconfigurations in a single step.

Finally, Canfora *et al.* (Canfora et al., 2005) describe a QoS-aware service discovery and late-binding mechanism which is able to automatically adapt to changes of QoS attributes in order to meet the SLA. The binding is done at run-time, and depends on the values of QoS attributes which are monitored by the system. It should be observed that in SAVER we consider a different scenario, in which each WS has just one implementation which however can be instantiated multiple times. The goal of SAVER is to satisfy a specific QoS requirement (mean execution time of workflows below a given threshold) with the minimum number of instances.

7 CONCLUSIONS

In this paper we presented SAVER, a QoS-aware algorithm for executing workflows involving Web Services hosted in a Cloud environment. SAVER selectively allocates and deallocates Cloud resources to guarantee that the response time of each class of workflows is kept below a negotiated threshold. This is achieved though the use of a QN performance model which drives a greedy optimization strategy. Simulation experiments show that SAVER can effectively react to workload fluctuations by acquiring/releasing resources as needed.

Currently, we are working on the extension of SAVER, exploring the use of forecasting techniques as a mean to trigger resource allocation and deallocation proactively. More work is also being carried out to assess the SAVER effectiveness through a more comprehensive set of real experiments.

REFERENCES

Calinescu, R. (2009). Resource-definition policies for autonomic computing. In *Proc. Fifth Int. Conf. on Autonomic and Autonomous Systems*, pages 111–116, Washington, DC, USA. IEEE Computer Society.

Canfora, G., Di Penta, M., Esposito, R., and Villani, M. L. (2005). Qos-aware replanning of composite web services. In *Proceedings of the IEEE International Conference on Web Services*, ICWS '05, pages 121–129, Washington, DC, USA. IEEE Computer Society.

Ferretti, S., Ghini, V., Panzieri, F., Pellegrini, M., and Turrini, E. (2010). Qos-aware clouds. In *Proceedings of the 2010 IEEE 3rd International Conference on Cloud Computing*, CLOUD '10, pages 321–328. IEEE Computer Society.

Huber, N., Brosig, F., and Kounev, S. (2011). Model-based self-adaptive resource allocation in virtualized environments. In *SEAMS '11*, pages 90–99, New York, NY, USA. ACM.

Jung, G., Hiltunen, M. A., Joshi, K. R., Schlichting, R. D., and Pu, C. (2010). Mistral: Dynamically managing power, performance, and adaptation cost in cloud infrastructures. In *ICDCS*, pages 62–73. IEEE Computer Society.

Kalyvianaki, E., Charalambous, T., and Hand, S. (2009). Self-adaptive and self-configured cpu resource provisioning for virtualized servers using kalman filters. In *ICAC'09*, pages 117–126. ACM.

Kephart, J. O., Chan, H., Das, R., Levine, D. W., Tesauro, G., III, F. L. R., and Lefurgy, C. (2007). Coordinating multiple autonomic managers to achieve specified power-performance tradeoffs. In *ICAC'07*, page 24. IEEE Computer Society.

Lazowska, E. D., Zahorjan, J., Graham, G. S., and Sevcik, K. C. (1984). *Quantitative System Performance: Computer System Analysis Using Queueing Network Models*. Prentice Hall.

Li, J., Chinneck, J., Woodside, M., Litoiu, M., and Iszlai, G. (2009). Performance model driven qos guarantees and optimization in clouds. In *CLOUD '09*, pages 15–22, Washington, DC, USA. IEEE Computer Society.

Litoiu, M., Woodside, M., Wong, J., Ng, J., and Iszlai, G. (2010). A business driven cloud optimization architecture. In *Proc. of the 2010 ACM Symp. on Applied Computing*, SAC '10, pages 380–385. ACM.

Marzolla, M. and Mirandola, R. (2011). A Framework for QoS-aware Execution of Workflows over the Cloud. arXiv preprint abs/1104.5392.

Urgaonkar, B., Pacifici, G., Shenoy, P., Spreitzer, M., and Tantawi, A. (2007). Analytic modeling of multitier internet applications. *ACM Trans. Web*, 1.

Yazir, Y. O., Matthews, C., Farahbod, R., Neville, S., Guitouni, A., Ganti, S., and Coady, Y. (2010). Dynamic resource allocation in computing clouds using distributed multiple criteria decision analysis. In *CLOUD '10*, pages 91–98. IEEE Computer Society.

Zhang, Q., Cheng, L., and Boutaba, R. (2010). Cloud computing: state-of-the-art and research challenges. *J. of Internet Services and Applications*, 1:7–18.

Zhu, X., Young, D., Watson, B. J., Wang, Z., Rolia, J., Singhal, S., McKee, B., Hyser, C., Gmach, D., Gardner, R., Christian, T., and Cherkasova, L. (2009). 1000 islands: an integrated approach to resource management for virtualized data centers. *Cluster Computing*, 12(1):45–57.