RightCapacity: SLA-driven Cross-Layer Cloud Elasticity Management

YOUSRI KOUKI
Ascola Research Group, France
and
THOMAS LEDOUX
Ascola Research Group, France

Cloud computing paradigm has become the solution to provide good service quality and exploit economies of scale. However, the management of such elastic resources, with different Quality-of-Service (QoS) combined with on-demand self-service, is a complex issue. New challenges for elasticity management arise when people look deeper into the Cloud characteristics such as non-ignorable instance initiation time and full hour billing model. The main challenge for a SaaS provider is to determine the best trade-off between profit and end-user satisfaction. This paper proposes RightCapacity, an approach driven by Service Level Agreement (SLA) for optimizing the Cloud elasticity management (i.e., both elasticity at the application and at the infrastructure levels). We consider cross-layer (application-resource) Cloud elasticity. We model Cloud application using closed queueing network model taking into account the SLA concept and the Cloud economic model.

Our results show that RightCapacity successfully keeps the best trade-off between SaaS provider profit and end-user satisfaction. Using RightCapacity, the cost saving of as much as 30% can be achieved while causing the minimum number of violations, as small as 1%.

Keywords: Cloud Computing, Quality-of-Service (QoS), Service Level Agreement (SLA), Capacity Planning, Cross-layer Elasticity

1. INTRODUCTION

According to NIST [Hogan and al 2011], Cloud computing is a model for enabling on-demand network access to a shared pool of configurable computing resources as services. Elasticity, the ability of an application to adjust the infrastructure resources, is a key property of Cloud computing. Resources elasticity can be implemented in several ways in particular via auto-scaling i.e., increase seamlessly the used resources during demand spikes to maintain the Quality of Service (QoS), and decrease automatically during demand lulls to minimize costs. The QoS is formalized via Service Level Agreements (SLA). It specifies one or more Service Level Objectives (SLOs) to guarantee that the service quality is delivered to satisfy pre-agreed userséxpectations. In case of SLA violation, penalties are applied to the service provider in order to compensate the users for tolerating the service failure.

Thanks to auto-scaling service, dynamic resource scaling in Cloud computing environments is a fairly simple task. But doing it *right* in a way that maintains SLAs while minimizing service cost well is not so trivial a task unless the capacity planning takes into account non-ignorable instance initiation time [Mao and Humphrey 2012] and full hour billing model. With respect to the existing work, industrial initiatives such as Amazon Auto-Scaling[1], and scalr[2] and recent research works [Dutta et al. 2012; Shen et al. 2011] only focus at the infrastructure level by optimizing resource scaling, whereas in this paper we take into account both levels: application and infrastructure. In fact, only resource elasticity may not be enough to support Cloud challenges, more precisely

[1]http://aws.amazon.com/autoscaling/
[2]https://scalr.net/

Authorsáddresses: Yousri Kouki, *É*cole des Mines de Nantes, 44307 France, Email: Yousri.kouki@inria.fr; Thomas Ledoux, *É*cole des Mines de Nantes, 44307 France, Email: Thomas.ledoux@inria.fr.

non-ignorable instance initiation time. To address this issue, we introduce application elasticity where applications go through a degraded mode in order to absorb instance initiation time and minimize the infrastructure oscillations. Indeed, we consider Cloud application as white boxes to switch from one configuration to another at runtime. In particular, we distinguish between two configurations: normal mode and degraded mode. An example of degradation is to limit the number and level of details in the answers returned to the end-user. The main purpose of the application elasticity is to ensure the availability of service even in degraded mode.

This paper focuses on the SaaS provider point of view by emphasizing its producer-consumer role in the Cloud stack. The objective of the SaaS provider is to maximize its revenues and minimize its costs. Its revenues are determined through serving user queries according to QoS expectations, whereas its costs are due to hosting fees and SLA penalties that the SaaS provider has to pay in case of violations. In order to address this issue, we propose RightCapacity: a SLA-driven approach for optimizing Cloud elasticity management. We model Cloud services using closed queueing network model and we propose an extension of a Mean Value Analysis (MVA) [Reiser and Lavenberg 1980] algorithm to take into account SLA and Cloud economic model. Then, we propose a capacity planning approach allowing a SaaS provider to find for a given workload an optimal configuration (application and resources) to fulfill the SLA and to minimize the cost of the service.

The key contributions of this paper are as follows:

- Business policies: we propose business policies to keep capacity planning method under control and fit to Cloud economic model (i.e., full hour billing model),
- Cross-layer Cloud elasticity: we take into account both levels: application and infrastructure. We rely on application elasticity to absorb non-ignorable instance initiation time and minimize the infrastructure oscillations,
- Objective function: we define an objective function to optimize capacity planning for Cloud application.

2. ANALYTICAL MODEL

One main challenge for SaaS provider is to provide a fine-tuned resource management, satisfying a major number of its clients while minimizing its resources cost. To address this problem, we present an analytical model for multi-tier Cloud applications to predict service performance. This model takes into account the concept of Service Level Agreement (SLA). It is based on a network of queues to represent how the tiers in a multi-tier application cooperate to process requests.

2.1 Service Level Agreement Model

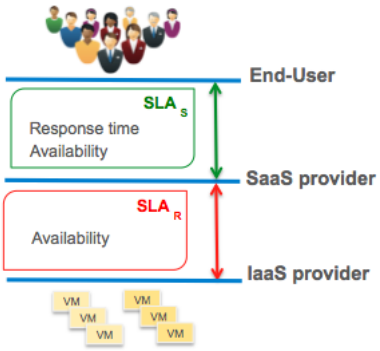


Figure 1. Actors and SLAs

In Cloud computing, the pay-per-use model enables service providers to offer their services to customers in different Quality-of-Service (QoS) levels. SLA is a negotiated agreement between a service provider and a customer where QoS parameters specify the quality level of service that the service provider has to guarantee. From architectural point of view, XaaS (Everything as a Service) layers are connected in a producer-consumer way, in which the upper layers depend on services provided by underlying layers, as it can be seen in Figure 1, providing SLA dependencies across the different Cloud layers [Kouki et al. 2011]. From SaaS providerś point of view, there are two layers of SLA:

- SLAS between End-user and SaaS provider,
- SLAR between SaaS provider and IaaS provider.

```
<csla:Obligations xmlns:csla="http://www.inria.fr/cslamodel">
  <csla:Guarantees>
    <csla:Guarantee guaranteeID= "G1" serviceID="S1">
      <csla:SLO sloID="PerformanceSLO" type ="simple">
        <csla:Simple metric="responseTime"  unit="second /request"
                     comparator="le"  threshold="5" fuzziness="1"/>
      </csla:SLO>
      <csla:SLO sloID="AvailabilitySLO"  type ="simple">
        <csla:Simple metric="abandonRate " unit="%"
                     comparator="le" threshold="1" fuzziness="0.2"/>
      </csla:SLO>
      <csla:SLO sloID="CompSLO"  type ="composite">
        <csla:Composite A="PerformanceSLO" operator="and"
                        B="AvailabilitySLO"/>
      </csla:SLO>
    </csla:Guarantee>
  </csla:Guarantees>
  <csla:Requirements>
    <csla:Requirement requirementID="R1" sloID="CompSL0">
     ...
    </csla:Requirement>
  </csla:Requirements>
  <csla:Confidences>
    <csla:Confidence confidenceID="C1" sloID="CompSLO" percentage ="95"
                     unit="allRequests"/>
  </csla:Confidences>
  <csla:Penalties>
    <csla:Penalty penaltyID="P1" condition="violation" sloID="CompSLO">
      <csla:Constant value ="0.3" unit="euro/violatedRequest"/>
    </csla:Penalty>
  </csla:Penalties>
</csla:Obligations>
```

Figure 2. Contract between End-user and SaaS provider.

Figure 2 presents an example described with the CSLA language [Kouki and Ledoux 2012a], a SLA language dedicated to Cloud computing. Due to the limitation of paper space, we only describe the obligations section of SLA, which constitutes input parameters in the analytical model for multi-tier Cloud applications. We defined a composite SLO that is composed (using ändöperator) of a performance SLO and a availability SLO: the response time must be less than 5 seconds/request with acceptable margin less than 1 second and the abandon rate of the service must be less than 1% with an acceptable margin less than 0,2%. SLOs guaranteed on at least 95% of requests to the SaaS service i.e. clauses must be meet for 95% of requests to the SaaS service. If more than 5% of requests to the SaaS service violates the SLOs, a penalty of 0,3 euro violated request is applied [3].

2.2 Application Configuration Model

Service Based Application (SBA) is characterized by the operating mode. A mode depends on which components are running, their properties (intra-component), and how these components are bound to each other (inter-component). Let I be the number of components $Com_1, \cdots, Com_I$. Each component Com_i has a fixed number of configurations $J(Conf_{Com,i,1}, \cdots, Conf_{Com,i,J})$. We distinguish between two operating mode: i) normal mode: all components are running in normal mode and ii) degraded mode: at least one component is running in degraded mode. An

[3]Generally, penalty value p depends on delay time: $p = alpha + beta$ dt. For the sake of simplicity, the penalty is fixed here to 0,3 \$euro\$ violated request.

example of degradation is to limit the number and level of details in the answers returned to the end-user.

2.3 Resource Configuration Model

The resources configuration of a SaaS application is characterized by the topological structure of an application. Let M be the number of tiers, $T_1, \cdots, T_M$. We use K_m to denote the number of replicas at tier $T_m (m = 1, \cdots, M)$. The MPL_m represents servers MPL (Multi-Programming Level) at tier T_m. MPL control consists in fixing a limit for the maximum number of clients allowed to concurrently access a server. We use initm to denote the instance initiation time at tier T_m. Let Pr_m be the price per instance-hour consumed for each instance at tier T_m. We use Scm to denote the software pricing and licensing costs at tier T_m.

2.4 Service Workload Model

Service workload is characterized, on the one hand, by the workload amount (i.e., the number of clients that try to concurrently access a server), and on the other hand, by the workload mix (i.e., the nature of requests made by clients and the way they interleave). Let C be the number of classes of users. Each class c has a fixed number of users N_c with think time Z_c (i.e., the time between the reception of a response and the sending of the next request by a client). Let $S_{c,m}$ denote the service time of class C_m at tier m and $V_{c,m}$

denote request rate of class C_m at tier m. D_c is the inter-tier communication delays. We denote the total number of sessions by N, so $N = \sum_{c=1}^{C} N_c$

2.5 Queueing Network Model

We follow a queueing network approach, where a multi-tier system is modeled as an $M/M/c/K$ queue. SaaS services are modeled as closed loops to reflect the synchronous communication model that underlies these services. That is, a client waits for a request response before sending another request. We rely on Mean Value Analysis (MVA) [Reiser and Lavenberg 1980] for evaluating the performance of the queueing network.

We extend the MVA algorithm to take into account the following main features: multi-tier, multi-station, multi-class queueing network mode (see Figure 3). The algorithm takes the following set of parameters as inputs: (i) the application: I, $Conf_{Com,i} (i = 1, \cdots, I, j = 1, \cdots, J), (ii)$ the resources: $M, K_m, Pr_m, Sc_m, MPL_m, init_m (m = 1, \cdots, M), (iii)$ the workload: $C, N_c, Z_c, S_{c,m}, V_{c,m}, D_{c,m} (c = 1, \cdots, C, m = 1, \cdots, M)$ and $(iv) SLA_c (c = 1, ..., C)$. The algorithm computes the following metrics as output per-class $(c = 1, \cdots, C)$:

- σ_c: service latency/response time;
- α_c: service abandon rate;
- ω_c: service financial cost.

It consists of three parts. The first part calculates the admitted requests and rejected requests at each tier per class. Admission control is a key component for QoS delivery because it determines the extent to which resources are utilized. In this work, we consider a static class-based priority policy, which classifies requests according SLAs. Thus it is crucial to give priority to gold requests without causing starvation for other requests. The second part is dedicated to the prediction of service request latency and service abandon rate. Finally, the third and last part of the algorithm calculates the total cost ω of SaaS service in terms of servers that underlie the system and penalties. The total cost is defined by equation 1:

$$\omega = \sum_{m=1}^{M} Cost_{VM,m} + \sum_{m=1}^{M} Cost_{Software,m} + \sum_{c=1}^{C} Cost_{Penalty,c} \quad (1)$$

Where $Cost_{VM,m}$ is the cost of VM at tier m, $Cost_{Software,m}$ (Sc_m) is the software pricing and licensing costs at tier m and $Cost_{Penalty,c}$ is the cost of penalties of class c.

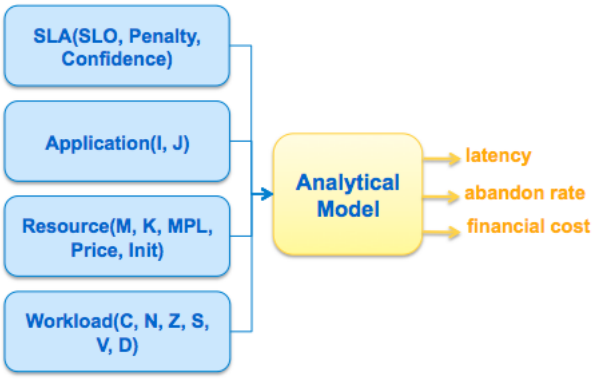


Figure 3. Analytical model.

The VM cost depends on the price of VM and rental duration (see equation 2).

$$Cost_{VM,m} = \sum_{k=1}^{K_m} Pr_{k,m}.duration_{k,m} \tag{2}$$

Prk,m is the price per instance-hour consumed for each VM instance at tier m. durationk,m is the sum of instance-hour of VM k at tier m. The penalty cost is based on confidence property and penalty model (see equation 3).

$$Cost_{penalty,c} = Penalty_c(\sigma_c, \alpha_c, SLA_c) \tag{3}$$

Where $Penalty_c$ is the penalty of violation of consumers of class c as mentioned in [Kouki and Ledoux 2012a].

3. CAPACITY PLANNING METHOD

Based on the analytic model previously described to predict Cloud service performance and availability, we present an approach for capacity planning. First, business policies are defined to control the capacity planning method. Second, a capacity planning method is developed to calculate the best configuration that guarantees the SLA constraints while minimizing the service cost according to business policies.

3.1 Business policies

We propose to parameterize the capacity planning method by business policies. We distinguish two kinds of business policy: (i) pricing policies and (ii) scaling policies. Dynamic pricing policy, such as Amazon "spot pricing", is better to generate more revenue but more complicated to implement. In this paper, we consider a static service price model. Regarding scaling policies, we propose "Use as you Pay" (UaP) policy that check scale in (removal of resources) process. "UaP" policy consists to use the full-instance period before terminating any instance. In fact, Cloud is typically based on full hour billing model. For example, with Amazon EC2, pricing is per instance-hour consumed for each instance, thus each partial instance-hour consumed will be billed as a full hour! In this way, we use what we pay exactly. Further, this policy allows avoiding system oscillations particularly face to "On and Off"[4] workload. We can consider this policy as a proactive strategy. In fact, an instance waiting for the end of an instance-hour to terminate can be useful if there is an increasing workload. In addition we propose some parameters to fit both to the application type and its workload: (i) a calm period, a time during which no scaling

[4]http://watdenkt.veenhof.nu/2010/07/13/workload-patterns-for-cloud-computing/

decisions can be committed in order to avoid system oscillations and (ii) maximum capacity to keep scale out (addition of resources) under control.

3.2 Capacity Planning Method

RightCapacity takes into account both levels: application and infrastructure. In case of violation, we propose to degrade the Cloud application in order to absorb the incoming requests. In this paper, we consider a simple degradation of the application: degraded all components to absorb the maximum requests. How to choose the best components configurations according to workload demand is on-going work. After a calm period, if the SLA constraints are checked, the application returns in a normal mode. Otherwise, RightCapacity calculates resources capacity thanks to our capacity planning method. The application continues in degraded mode until all instances have been activated. The variables MPL_m and K_m $(m = 1, \cdots, M)$ are the unknowns of the capacity planning problem. Values of variables are taken from a domain, which is defined by a set of values. The constraints are the SLA and the business policies. The objective is to maximize the SaaS profit (i.e., minimize the cost of the service). The capacity planning method (see Figure 4) consists of two main algorithms: (i) calculate the demanded capacity and (ii) compute the optimal configuration that maximizes the objective function.

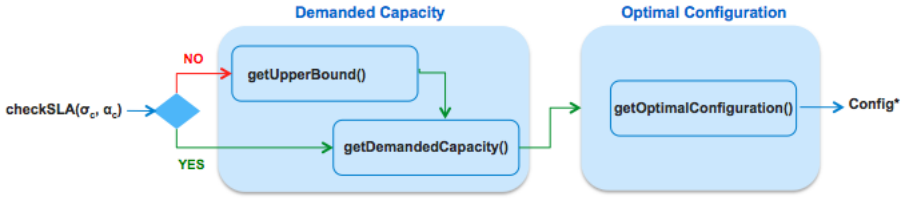


Figure 4. Capacity planning method.

3.2.1 *Demanded Capacity.* The first algorithm (demanded capacity) consists of two parts: (i) define the upper bound of variable K_m $(m = 1, \cdots, M)$; (ii) run a search until matching the demanded capacity. The first part first checks the SLA constraints. If the SLA constraints are checked, we consider current values of K_m $(m = 1, \cdots, M)$ as upper bound and the second part will be invoked. Whereas in case of violation, it increases the number of servers assigned to all tiers of the Cloud application. It then adjusts the MPL of servers to distribute client requests among server replicas at each tier. Afterwards, the resulting service configuration along with the workload is used to predict the latency, abandon rate and cost of the service. This process is repeated until an upper bound of K_m that meets the SLA is found. From the upper bound found of K_m, the second part estimates the demanded capacity. We define demanded capacity as the minimum value of K_m as mentioned in our previous work [Kouki and Ledoux 2012b]. Finally, the second algorithm of capacity planning method computes the optimal solution (MPL_m and K_m $(m = 1, \cdots, M)$) that maximizes the objective function Θ described below.

3.2.2 *Optimal Configuration.* We consider a SLA for class c described with the CSLA language (see Figure 3) that specifies service performance and availability constraints in the form of maximum response time and maximum abandon rate. Performance and availability preference of a Cloud service is defined in equation 4.

$$\rho(\sigma_c, \alpha_c) = (\sigma_c \leq \sigma_{max,c} + F_{\alpha,c}).(\alpha_c \leq \alpha_{max,c} + F_{\alpha,c}) \tag{4}$$

Where σ_c, α_c are respectively the current latency and abandon rate while $F_{\sigma,c}$, $F_{\alpha,c}$ are respectively the latency fuzziness and abandon rate fuzziness of the Cloud service for class c. Note that $\forall\sigma_c, \forall\alpha_c, \rho_c(\sigma_c, \alpha_c) \in \{0, 1\}$

We introduce equation 5 for C class:

$$\rho_c = \prod_{c=1}^{C} \rho_c(\sigma_c, \alpha_c) \tag{5}$$

To check $capacity_{max}$, we define equation 6 where $capacity_{max}$ is the maximum threshold for capacity. Note that $\sum_{m=1}^{M} K_m, \chi \in \{0, 1\}$

$$\chi(M) = \sum_{m=1}^{M} K_m \leq capacity_{max} \tag{6}$$

We model the concept of "Use as you Pay" policy via equations 7, 8 and 9 that define the set of instances that can be terminated.

$$UaP_k(k) = (dr{<}(0, 95.fullT).roundUp(dr(k)/fullT)) \tag{7}$$

$$UaP_m(m) = (\sum_{k=1}^{K} UaP_k(k)) \tag{8}$$

$$UaP_M(M) = (\sum_{m=1}^{M} UaP_m(m)) \tag{9}$$

Where dr is the duration of instance k allocation, $fullT$ is the instance price model (e.g., hour for Amazon $EC2$), UaP_k, UaP_m and UaP_M are respectively the decision for the instance k, for the tier m and for all tier. Note that $\forall M, UaP(k) \in \{0, 1\}$. Also, we define the equation 10 to distinguish between scale out and scale in.

$$scale(M) = \begin{cases} 1 & \text{if } scaleOut \\ UaP_M(M) & \text{if } scaleIn \end{cases} \tag{10}$$

Based on performance and availability preference, cost of a Cloud application and business policies, the objective function of the Cloud service combines multi-criteria as follows:

$$\Theta(M, \rho_c, \omega) = \frac{M}{\omega}.\rho_c.scale_M(M).\chi \tag{11}$$

Where ω is the current cost of the Cloud application, and M is the number of tiers.

A high value of the objective function reflects the fact that, on the one hand, the Cloud application guarantees SLO for performance and availability and, on the other hand, the cost underlying the service is low. Values of variables $K_m(m = 1, \cdots, M)$ are taken from a domain which is between current values and demanded capacity. Then, we define the optimal configuration:(config*: $MPL_m, K_m(m = 1, \cdots, M)$) as the configuration that maximizes the objective function.

Depending on the value of the objective function, current capacity and demanded capacity, we define the optimal reconfiguration plan. If the value of the objective function is equal to "0" and ρ_C is equal to "1", there may be two cases: (i) in case of scale out, demanded capacity is greater than capacitymax, (ii) in case of scale in, no instance can be terminated. Thus, we keep the same configuration and in case of scale in, deferred plan is defined that is a scheduling of full-time-instance consumption notification. Else, if the value of the objective function is greater than "0", we define an immediate plan to scale horizontally (out or in) and scale vertically

(adjust $MPL_m(m = 1, ..., M)$). In addition, if the optimal configuration is greater then the demanded capacity (scale in), we define a deferred plan. Finally, if current configuration is equal to demanded capacity, the optimal reconfiguration plan is a simple adjustment of the admission control ($MPL_m(m = 1, ..., M)$).

4. PERFORMANCE EVALUATION

RightCapacity is well suited for any type of Cloud applications. This Section presents the performance results obtained from an extensive set of experiments.

4.1 Experimental Methodology

A number of simulation experiments have been conducted to evaluate the performance of the proposed approach for Cloud elasticity management. In order to predict Cloud service performance and availability, our model requires several parameters as inputs. In practice, the parameters of the workload mix ($Z_c, S_c, m, V_c, m, D_c, m(c = 1, ..., C, m = 1, ..., M)$) can be estimated by monitoring the application workload as mentioned in [Urgaonkar et al. 2007; Chen et al. 2008].

We rely on a business intelligence application to evaluate the impact on the performance of our proposed solution. The business intelligence application consists on several components, among them only three offer a degraded mode. The following modes are summarized in Table I.

Table I: Components modes

Components	***Normal mode***	***Degraded mode***
Search	Unlimited	Limited
Historical	Last 10 answers	Last 2 answers
Display	2D	1D

We consider 3-tier architecture[5] (M=3), distributed and replicated in the Cloud: (i) a load balancer tier that is responsible to distribute incoming application traffic across multiple application servers, (ii) an application server tier that implements business logic functionality, and (iii) a back-end database. There are multiple classes of users (C=3) in the business intelligence application, representing different SLAs (e.g., Gold, Silver or Bronze customers) and heterogeneous workloads (e.g., read requests, write requests or read-write requests). The SLA is based on performance (response time) and availability (abandon rate). The following SLAs are summarized in Table II.

Table II: SLAs BI Application

Class	σ_{max} ***(sec/reqt)***	α_{max} ***(%)***	***Penalty*** **euro**
Gold	5	1	0,3
Silver	5	2	0,2
Bronze	5	3	0,1

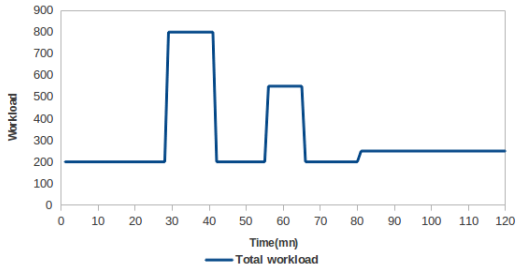
In our experiments, we examine RightCapacity with N=1000 customers (c1=200, c2=350 and c3=450). Two workload mixes were used: read-only interactions and write-only interactions. The following experiments were conducted with standard on-demand instances (EC2) . The price of instance running Linux (Region EU (Ireland)) is equal to 0.1318 euro. The preconfigured instance initiation time is equal to 2mn. With regard to business policies the calm period=1 mn and the capacitymax=10 instances that can be estimated using our prediction model.

[5]http://support.rightscale.com/03-Tutorials/02-AWS/02-WebsiteEdition

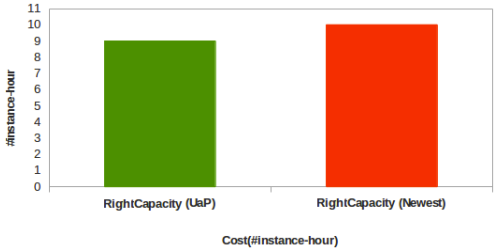
We compare two implementations of RightCapacity: i) RightCapacity (UaP) and ii) RightCapacity (Newest). “Newest” policy consists in terminate immediately the last instances allocated such as the standard scaling policies while ”UaP” use the full-instance period before terminating any instance. We evaluate the performance of our solution with representative workload patterns in the Cloud environment such as Unpredictable Bursting and ”On and Off”.

4.1.1 *Results.* RightCapacity produces the optimal configuration of the Cloud application as a result of the proposed capacity planning method. Results also show that thanks to the business policies and the cross-layer Cloud elasticity, the cost saving of as much as 30% can be achieved while causing the minimum number of violations, as small as 1%.

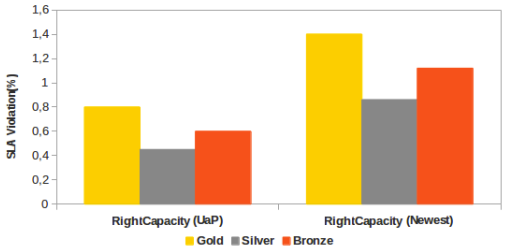
RightCapacity can be suited for any workload patterns to handle the trade-off between SaaS profit and end-user satisfaction levels. Even in the face to Unpredictable Bursting workload, RightCapacity reduces #instance hour to pay while minimizing SLA violations (see Figure 5). The “Use as you Pay” policy performs best to i) reduce



(a). Unpredictable Bursting Workload.



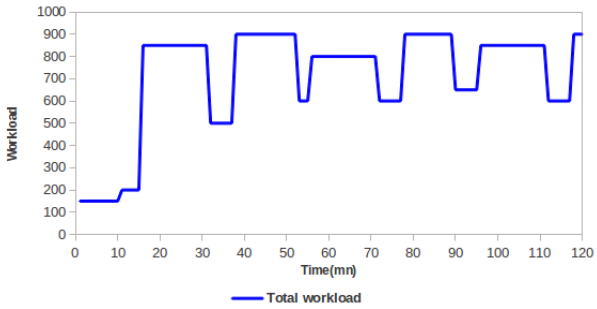
(b). Service cost.



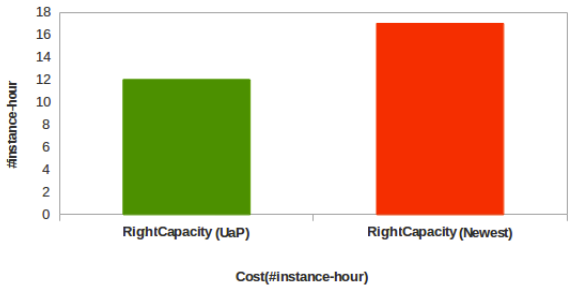
(c). SLA violations.

Figure 5: RightCapacity face to Unpredictable Bursting Workload

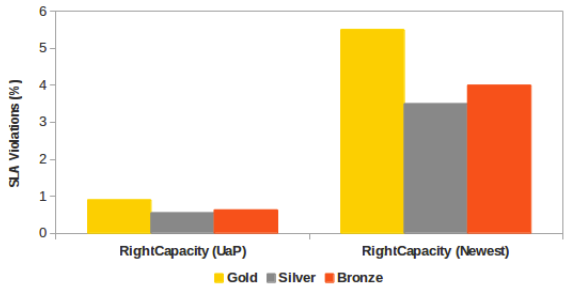
#instance hour to 9 only versus 10 for "Newest" Policy and ii) cause the minimum number of violations. The capacity planning method ensures the return to stability in a minimum time. The application elasticity ensures service availability even in degraded mode for the maximum number of requests during resources reconfiguration. In addition, RightCapacity adjusts the admission control per instance (MPL) automatically as per demand, which can absorb violations without increasing the cost. All this will influence the number of SLA violations. Figure 5(c) shows that RightCapacity(UaP) causes the minimum number of violations, as small as 1%.



(a). On and Off Workload.



(b). Service cost.



(c). SLA violations

Figure 6: RightCapacity face to On and Off Workload

Face to "On and Off" workload (see Figure 6), RightCapacity reaches the maximum saving cost and the minimum SLA violations. It can be observed in Figure 6(b) that during the variation of demands, RightCapacity(UaP) minimize cost to 12 instance hour only versus 17 instance ? hour for RightCapacity(Newest) . Figure 6(b) shows that "Use as you Pay" policy generates

less number of SLA violations than "Newest" policy. In fact, "Use as You Pay" policy allows avoiding system oscillations and implicitly SLA violations. The RightCapacity(UaP) cost saving of as much as 30% can be achieved while SLA violations are always kept below 1%.

4.2 Discussion

RightCapacity takes into account non-ignorable instance initiation time and full hour billing model: (i) the application elasticity absorbs non-ignorable instance initiation time and (ii) the "Use as you Pay" policy performs well to fit to full hour billing model. In addition the capacity planning method ensures the return to stability in a minimum time.

Cloud elasticity is a double-edged solution. In fact, the capacity planning method cannot differentiate between valid workload and malicious one. A user can execute a distributed denial of service attack against the Cloud service (add more and more instances infinitely). In our case, RightCapacity kept under control by imposing a maximum capacity value that can be estimated using our analytical model.

4.3 RELATEDWORK

In traditional computing environments, resources have to be purchased in advance via ad-hoc provisioning. With the revolutionary promise of computing as a utility, Cloud computing has the potential to transform how IT services are delivered and managed. The great advantage of Cloud computing is elasticity. Recent work [Buyya et al. 2011] focus on approach for optimizing providers' environments while guaranteeing consumers' SLAs. We survey here those work directly related to SLA-oriented capacity planning management for Cloud computing environments. There are two kind of categories: (i) heuristic-based approaches [Wu et al. 2011; Freitas et al. 2011; Emeakaroha et al. 2010]: these heuristics do not provide guarantees on the QoS; (ii) model based approaches such as queuing theory based models [Urgaonkar et al. 2007; Chen et al. 2008; Arnaud and Bouchenak 2011] and game theory based models [Teng and Magoules 2010; Ardagna et al. 2011].

Wu et al. [Wu et al. 2011] propose resource allocation algorithms for SaaS providers who want to minimize infrastructure cost and SLA violations. Three cost driven algorithms are proposed. The ProfminVMminAvaiSpace algorithm optimizes cost savings better when compared to the other proposed algorithms. These algorithms do not ensure guarantees. The use of game theory for the resource allocation problem is investigated in [Teng and Magoules 2010]. The paper models the service provisioning problem as a Generalized Nash game, and proposes an algorithm for the run-time management and allocation of IaaS resources to competing SaaS.

[Arnaud and Bouchenak 2011] presents MoKa, a system for adaptive control of Internet services to guarantee performance and availability objectives and to minimize cost. Moka is an approach for building self-adaptive Internet services through utility-aware capacity planning and provisioning. This approach is based in single class MVA whereas, in our work, we consider multi-class MVA. Y. Chen et al [Chen et al. 2008] proposes an SLA decomposition approach that combines performance modeling with performance profiling to solve this problem by translating high level goals to more manageable low-level sub-goals. Nevertheless, our work differs on SLA and cost model.

Y. Jiang et al. [Jiang et al. 2012] proposes a new method for Cloud capacity planning with the goal of fully utilizing the physical resources, as we believe this is one of the emerging problems for Cloud providers. To solve this problem, they present an integrated system with intelligent Cloud capacity prediction.

All these approaches have contributed significantly to determine the best trade-off between profit and customer satisfaction levels. However, none presents a method taking into account non-ignorable instance initiation time and full hour billing model. In comparison, RightCapacity relies on application elasticity to absorb non-ignorable instance initiation time and Use as You Pay policy to fit to full hour billing model. In summary, our contribution differs from related work in the following aspects:

4.4 CONCLUSION

Elasticity is a key feature of Cloud computing paradigm. The management of such elastic resources according to different workloads and different Quality-of-Service (QoS) end-userś expectations is a complex issue. In order to determine the best trade-off between profit and end-user satisfaction, recent work only focus at the infrastructure level by optimizing resource scaling, whereas in this paper we take into account both levels: application and infrastructure. RightCapacity produces the optimal configuration (application and resources) as a result of the proposed capacity planning method. Results also show that thanks to the business policies and the cross-layer Cloud elasticity, the cost saving of as much as 30% can be achieved while causing the minimum number of violations, as small as 1%.

In the next future, we plan to address some limitations of our approach by extending the model to handle requests priority setting. In addition, we are currently conducting a study on application reconfiguration in order to improve elasticity management. Finally, we intend to define dynamic pricing policies to improve end-user satisfaction levels.

4.5 ACKNOWLEDGMENTS

This work is supported by the MyCloud project (ANR-10- SEGI-010).

REFERENCES

Ardagna, D., Panicucci, B., and Passacantando", M. 2011. A game theoretic formulation of the service provisioning problem in cloud systems. In *Proceedings of the 20th international conference on World wide web (WWW11)*. New York, NY, USA.

Arnaud, J. and Bouchenak, S. 2011. Performance and dependability in service computing, chapter performance, availability and cost of self-adaptive internet services. IGI Global.

Buyya, R., Garg, S. K., and Calheiros, R. N. 2011. Sla-oriented resource provisioning for cloud computing: Challenges, architecture, and solutions. In *Proceedings of the 2011 IEEE International Conference on Cloud and Service Computing (CSC 2011)*. Hong Kong, China.

Chen, Y., Iyer, S., Liu, X., Milojicic, D., and Sahai, A. 2008. Translating service level objectives to lower level policies for multi-tier services. cluster compute. Cluster Compute.

Dutta, S., Gera, S., Verma, A., and Viswanathan, B. 2012. Smartscale: Automatic application scaling in enterprise clouds. In *IEEE 5th International Conference on Cloud Computing (CLOUD)*.

Emeakaroha, V., Brandic, I., Maurer, M., and Breskovic, I. 2010. Sla-aware application deployment and resource allocation in clouds. The Second IEEE CloudApp 2011. Germany.

Freitas, A. L., Parlavantzas, N., and Pazat, J. L. 2011. Cost reduction through sla-driven self-management. IEEE Ninth European Conference on Web Services ECOWS.

Hogan, M. and al. 2011. Nist cloud computing standards roadmap, version 1.0.

Jiang, Y., Perng, C., Li, T., and Chang, R. 2012. Self-adaptive cloud capacity planning. International Conference on Service Computing (SCC), Hawaii, USA.

Kouki, Y. and Ledoux, T. 2012a. Csla: a language for improving cloud sla management. International Conference on Cloud Computing and Services Science, (CLOSER) 2012, Porto, Portugal.

Kouki, Y. and Ledoux, T. 2012b. Sla-driven capacity planning for cloud applications. IEEE International Conference on Cloud Computing Technology and Science, (CloudCom) 2012, Taipei, Taiwan.

Kouki, Y., Ledoux, T., and Sharrock, R. 2011. Cross-layer sla selection for cloud services. International Symposium on Network Cloud Computing and Applications, NCCA 2011, Toulouse, France. 2011.

Mao, M. and Humphrey, M. 2012. A performance study on the vm startup time in the cloud. IEEE 5th International Conference on Cloud Computing (CLOUD), 2012.

Reiser, M. and Lavenberg, S. 1980. Mean-value analysis of closed multichain queueing networks. J. ACM 27.

Shen, Z., Subbiah, S., Gu, X., and Wilkes, J. 2011. Cloudscale : Elastic resource scaling for multi-tenant cloud systems. In *Proceedings of the 2nd ACM Symposium on Cloud Computing (SOCC)*.

Teng, F. and Magoules, F. 2010. A new game theoretical resource allocation algorithm for cloud computing. In Advances in Grid and Pervasive Computing.

Urgaonkar, B., Pacifici, G., Shenoy, P., and Mike, S. 2007. Analytic modeling of multitier internet applications. ACM Trans. Web.

Wu, L., Garg, S. K., and Buyya, R. 2011. Sla-based resource allocation for software as a service provider (saas) in cloud computing environments. In *Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*.

Yousri Kouki received an engineer degree in computer science in 2010 and master degree in network and distributed systems in 2011 from the National School of Computer Science (ENSI), Tunisia. He is currently a PhD student in ASCOLA team (INRIA-CNRS group), France. His research focus on Service Level Management for Cloud computing. Particularly, he works on Service Level Agreement (SLA) management, capacity planning and auto-scaling.

Thomas Ledoux is a senior associate professor at cole des Mines de Nantes and member of the ASCOLA team (INRIA-CNRS group). He held a PhD from University of Nantes in 1998. He headed several national projects for the ASCOLA team. He has served on a number of conference program committees and is a member of the board of the Regional Doctoral School STIM. Thomas's main research interests include Component-Based Software Engineering (CBSE), Domain Specific Languages (DSL) for autonomic computing, green computing and cloud computing.