2011 International Conference on Parallel Processing

Virtual Machine Provisioning Based on Analytical Performance and QoS in Cloud Computing Environments

Rodrigo N. Calheiros*, Rajiv Ranjan†, and Rajkumar Buyya*

*Cloud Computing and Distributed Systems (CLOUDS) Laboratory
Department of Computer Science and Software Engineering
The University of Melbourne, Australia
Email: {rnc, rbuyya}@unimelb.edu.au

†CSIRO Information and Communication Technologies (ICT) Centre
Acton, ACT, Australia
Email: rranjans@gmail.com

***Abstract*—Cloud computing is the latest computing paradigm that delivers IT resources as services in which users are free from the burden of worrying about the low-level implementation or system administration details. However, there are significant problems that exist with regard to efficient provisioning and delivery of applications using Cloud-based IT resources. These barriers concern various levels such as workload modeling, virtualization, performance modeling, deployment, and monitoring of applications on virtualized IT resources. If these problems can be solved, then applications can operate more efficiently, with reduced financial and environmental costs, reduced under-utilization of resources, and better performance at times of peak load. In this paper, we present a provisioning technique that automatically adapts to workload changes related to applications for facilitating the adaptive management of system and offering end-users guaranteed Quality of Services (QoS) in large, autonomous, and highly dynamic environments. We model the behavior and performance of applications and Cloud-based IT resources to adaptively serve end-user requests. To improve the efficiency of the system, we use analytical performance (queueing network system model) and workload information to supply intelligent input about system requirements to an application provisioner with limited information about the physical infrastructure. Our simulation-based experimental results using production workload models indicate that the proposed provisioning technique detects changes in workload intensity (arrival pattern, resource demands) that occur over time and allocates multiple virtualized IT resources accordingly to achieve application QoS targets.**

I. Introduction

Cloud computing [1] is the latest evolution of computing, where IT resources are offered as services. The hardware and software systems that manage these services are referred to as Infrastructure as a Service (IaaS) and Platform as a Service (PaaS), while the actual applications managed and delivered by IaaS and PaaS are referred to as Software as a Service (SaaS).

The process of deployment and management of application services (SaaS) on ubiquitous Cloud infrastructures (such as Amazon EC2 [2], GoGrid[1], and Rackspace[2]) that expose their capabilities as a network of virtualized IT resources is known as Cloud Provisioning.

The process of provisioning in Clouds is a complex undertaking, as it requires the application provisioner to compute the best software and hardware configuration to ensure that QoS targets of application services are achieved, while maximizing the overall system efficiency and utilization.

Achieving QoS targets is important for meeting SLAs agreed with end-users and justifying the investment in Cloud-based deployments. However, this process is further complicated by the uncertain behavior of virtualized IT resources and network elements. At runtime, there may be unpredictable situations obstructing the smooth provisioning and delivery of application services such as:

- Estimation error: IT managers or SaaS owners can easily under or overestimate their needs because of lack of understanding of requirements due to complexities of Cloud-based IT resources and applications. As a result, it becomes extremely hard for IT managers to find the right combination of Cloud-based IT resources that can suitably fit current and anticipated application workload;
- Highly dynamic workload: An application service is used by large numbers of end-users, thus highly variable load spikes in demand can occur, depending on the day and the time of year, and the popularity of an application. Further the characteristic of workload [3] could vary significantly across application types (high performance, web hosting, and social networking). This causes serious problems while estimating the workload behavior (arrival pattern, I/O behavior, service time distribution, and network usage) and related resource requirements;
- Uncertain behavior: In a large-scale computing envi-

[1]http://www.gogrid.com
[2]http://www.rackspacecloud.com

0190-3918/11 $26.00 © 2011 IEEE
DOI 10.1109/ICPP.2011.17

ronment such as Cloud data centers, the availability, load, and throughput of Cloud-based IT resources and network links can vary in an unpredictable way. Transient conditions further complicate the process of determining aggregate resource requirements for provisioning an application. The provisioning problem is shown to be computationally intractable (i.e., NP-hard) [4]. If one is to add complexity and diversity of Cloud-based IT resources with the uncertain behavior of application workload, then the problem becomes much more challenging.

To counter the above complexities related to application provisioning on Clouds, this paper presents an adaptive approach with focus on automating routine management tasks, flexible delivery of virtualized IT resources and application when and where needed, and growing and shrinking of system capacity without: (i) over-provisioning; and (ii) having unacceptable impact on QoS targets. Applications and virtualized IT resources are automatically provisioned, pooled, and delivered on-demand according to business-driven policies.

Our provisioning technique tries to meet QoS targets, which are request processing time (response time) and service requests rejection, while preventing over-provisioning of IT resources. It optimizes the usage of IT resources, hence lowering consumption of physical, energy, and human resources while increasing efficiency and minimizing IT budgets.

We model the behavior and performance of different types of applications and IT resources to adaptively transform end user's service requests. We use analytical performance (queueing network system model) [5] and workload information to supply intelligent input about system requirements to an application provisioner, which improves the efficiency of the system. Such improvements ensure better achievement of QoS targets, while reducing costs due to improved utilization of IT resources. Workload information allows application provisioner to better understand workload demands in terms of resource needs, hence improving resilience to uncertainties and reducing estimation errors that may lead to unacceptable application performance and resource usage (over/under provisioning).

Analytical performance model allows the system to predict the effects of a provisioning schedule on target QoS. This model helps application provisioner to predict what mixes of infrastructures are most suited for a given application and when and how system capacity should be scaled up or down. Such models are simple and still efficient in delivering expected QoS because they are based on information available to the application provisioner. Therefore, aspects related to network connections between physical machines hosting the VMs, as well as other infrastructure-level aspects controlled by infrastructure providers are abstracted because this information is not disclosed to the application provisioner.

Our main contributions are:

1) an adaptive provisioning technique based on analytical performance and workload information for dynamically determining and capturing the relationship between application QoS targets and the allocation of individual IT resources. Our technique captures the complex behavior of applications including requests arrival rates and resource demands over time;
2) an analysis of two well-known application-specific workloads aimed at demonstrating the usefulness of workload modeling in providing feedback for Cloud provisioning;
3) a comprehensive simulation-driven analysis of the proposed approach based on realistic and well-known production environment workload models.

The rest of the paper is organized as follows. Section II discusses the Cloud provisioning process. This is followed by a detailed description on the Cloud resource and SaaS application models in Section III. Section IV presents the proposed adaptive Cloud provisioning technique. Section V presents experimental methodology, setup, and the discussion of results. Section VI presents related works. Section VII draws on some important conclusion along with suggestions for future research.

II. Comprehensive Cloud Provisioning Approach

Cloud Provisioning [3] is the process of deployment and management of applications on Cloud infrastructures. It consists of three key steps: (i) *Virtual Machine Provisioning*, which involves instantiation of one or more Virtual Machines (VMs) that match the specific hardware characteristics and software requirements of an application. Most Cloud providers offer a set of general-purpose VM classes with generic software and resource configurations. For example Amazon EC2 supports 11 types of VMs, each one with different options of processors, memory, and I/O performance; (ii) *Resource Provisioning*, which is the mapping and scheduling of VMs onto physical Cloud servers within a cloud. Currently, most IaaS providers do not provide any control over resource provisioning to application providers. In other words, mapping of VMs to physical servers is completely hidden from application providers; and (iii) *Application Provisioning*, which is the deployment of specialized applications (such as ERP system, BLAST experiments, and web servers) within VMs and mapping of end-user's requests to application instances.

In this paper, we focus on VM Provisioning and Application Provisioning, because these are the steps that application service providers can control. The goal of Application Provisioning is ensuring an efficient utilization of virtualized IT resources, which can be achieved through the use of techniques such as load balancing and efficient mapping of requests, while the goal of VM Provisioning is to provide applications with sufficient computational power, memory, storage, and I/O performance to meet the level of QoS expected by end-users. The latter is achieved either by increasing/decreasing capacity of deployed virtual machines or by increasing/decreasing the number of application and VM instances.

III. Models

In this section, we present an overview of our system architecture, the assumptions, and the notations (Table I) that

TABLE I
NOTATIONS: SYSTEM AND APPLICATION MODELS.

Symbol	Meaning
	System
P	a set of cloud computing infrastructures
c_i	i-th cloud (IaaS/PaaS) data center from P
n	number of cloud data centers
s_j	j-th application instance (such as software library, executable, data, or functional component)
v_j	j-th virtual machine
m	number of virtual machine instances allocated to an application
	Application
G_s	an application workload
r_l	l-th end-user request for G_s
h	number of requests, tasks, or work units within a workload
t_l	arrival time of request r_l at the application provisioner
T_r	response time of an end-user request
T_s	negotiated maximum response time of an end-user request
λ	expected arrival rate of requests at application provisioner
λ_{s_i}	expected arrival rate of requests at an application instance
$Rej(G_s)$	rejection rate of requests by G_s

drive our design. We assume that applications are hosted within virtual machines to enable resource sharing on a Cloud infrastructure. It is possible that a multi-tier application will run on multiple VMs that span cross computing servers. However, in this work we assume that there is one-to-one mapping relationship between an application instance s_i and a VM instance v_j. Since there is one-to-one mapping, we refer to them interchangeably in the rest of this paper.

A. System model

Clouds organize data centers as networks of virtualized IaaS (computing servers, databases, and networks) and PaaS (load-balancers and auto scalers) so that providers are able to access and deploy applications (SaaS) from anywhere in the world on demand at competitive costs driven by QoS requirements. The Cloud computing system [6], P, is a set of Cloud infrastructures owned and maintained by 3rd party IaaS/PaaS providers such as Amazon EC2, Flexiscale[3], and GoGrid. More formally, $P = (c_1, c_2, \ldots, c_n)$, where $c_1, c_2, \ldots, c_n$ are data centers that are part of P.

An application deployment is composed of m VM instances $\{v_1, v_2, \ldots, v_m\}$, where m is either fixed or varies with time based on the current workload and performance requirements. Application instances are examples of SaaS software that can be owned by small and medium business enterprises (SMEs) and governments who choose to offer their applications via Clouds. Without loss in generality, we consider in this paper the case where applications (SaaS) and platforms (PaaS) are offered by one organization and Cloud-based IT resources are owned by a different organization.

[3]http://www.flexiant.com/products/flexiscale

B. Application or SaaS Model

The cloud application scenario considered in this paper relates to execution of certain kind of action or functionality, by an application element s_j to end-users. The action or functionality varies based on the application model. For example, a public computing [7] service such as Folding@home and SETI@home provides functionality for executing mathematical models in a given set of data, whereas a Web server is a service that delivers content, such as web pages, using the Hypertext Transfer Protocol (HTTP) over the Internet.

Eventual dependencies among requests are assumed to be handled at the user side. Therefore, from the point of view of s_j, requests for actions or functionalities are independent from each other. This is the case, for example, of processing of HTTP requests. Even though some information about state of sessions may be stored in the Cloud, dependencies (e.g., security exceptions and protocols for communication) are handled at the end-user side by its web browser.

Several fundamental workloads in science, physics, commerce, media, and engineering can be modeled as *embarrassingly parallel* or *Bag of Tasks (BoT)* workloads, whose tasks are also independent. Some popular examples in this domain are scientific applications such as BLAST, MCell, and INS2D, and also public computing applications such as Einstein@home, Folding@home, and SETI@home. Typically, these are compute intensive applications composed of independent tasks that can be modeled as requests for services sent by end-users to an application instance.

The web server and the executor of BoT workloads are examples of other types of application models. In both cases, a workload G_s is composed of h *independent requests* (i.e., there is no communication between requests and computation required by each request is not distributed) $\{r_1, r_2, \ldots, r_h\}$ that are received by application instances at times $\{t_1, t_2, \ldots, t_h\}$.

Moreover, all the software and data requirements for execution of a request is met by the VM running it, what means that execution of service requests does not require utilization of Cloud storage. Relaxation of this requirement is part of our future work.

QoS targets related to an application includes response time T_s and rejection rate $Rej(G_s)$ of requests. These parameters are important because they have direct effect on the user experience about the SaaS application. If the response time is too high or if requests are rejected, some users might desert the application permanently, this attrition and possible negative publicity can result in loss of a portion of the revenue stream (usage and advertisement cost). Therefore, meeting QoS for an application is critical. To achieve this, we propose an adaptive mechanism for provisioning applications, which is described in the next section.

IV. ADAPTIVE CLOUD PROVISIONING APPROACH

To tackle the problem of uncertain behavior, estimation error, and dynamic workload related to Cloud provisioning, we propose an adaptive provisioning mechanism. The high level architecture of our approach is shown in Figure 1. Different

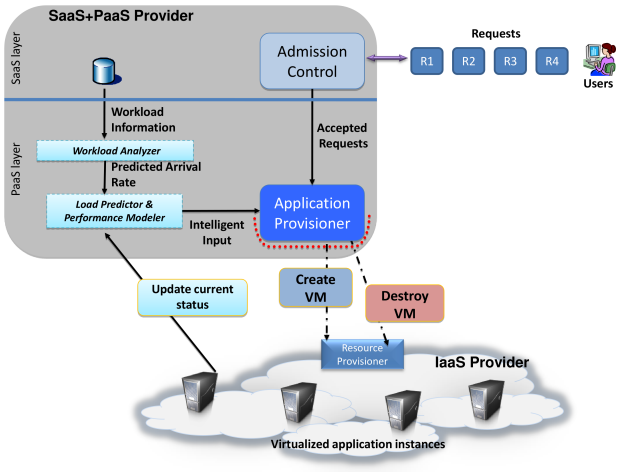

Fig. 1. Proposed mechanism for adaptive virtual machine provisioning.

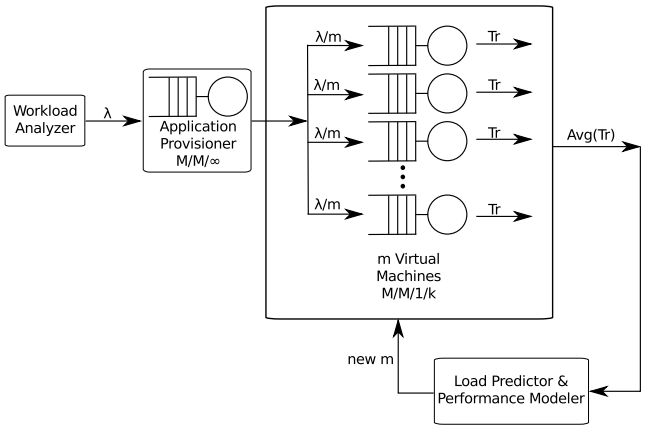

Fig. 2. Queueing model for the data center.

software components of the architecture are administered by the service provider. Its SaaS layer contains an admission control mechanism based on the number of requests on each application instance: if all virtualized application instances have k requests in their queues, new requests are rejected, because they are likely to violate T_s. Accepted requests are forwarded to the provider's PaaS layer, which implements the proposed system.

Mainly, the following components are critical to the overall functionality of the system: (i) *Application provisioner*, main point of contact in the system that receives accepted requests and provisions virtual machines and application instances based on the input from workload analyzer and from load predictor and performance modeler; (ii) *Workload analyzer*, which generates estimation of future demands for the application. This information is passed to the load predictor and performance modeler component; and (iii) *Load predictor and performance modeler*, which solves an analytical model based on the observed system performance and predicted load to decide the number of VM instances that should be allocated to an application. These three components are detailed in the rest of this section.

Our provisioning mechanism runs continuously to ensure that provisioning goals are met at all times. We set the following design goals for our provisioning approach:

- Automation: All decisions related to provisioning should be made automatically without human intervention;
- Adaptation: The application provisioner should adapt to uncertainties such as changes in workload intensity;
- Performance assurance: The resource allocation in the system can be dynamically varied for ensuring achievement of QoS targets.

A. Workload Analyzer

Workload analyzer is the component that is responsible for generating estimation (prediction) of request arrival rate. This information is used to compute the exact number of application instances required for meeting QoS targets and resource utilization goals. Prediction can be based on different information; for example, it can be based on historical data about resources usage, or based on statistical models derived from known application workloads.

Besides the particular method to estimate future load, the workload analyzer alerts the load predictor and performance modeler when service request rate is likely to change. This alert contains the expected arrival rate and must be issued before the expected time for the rate to change, so the load predictor and performance modeler has time to calculate changes in the system and the application provisioner has time to deploy or release the required VMs.

B. Load Predictor and Performance Modeler

Load predictor and performance modeler is responsible for deciding the number of virtualized application instances required to meet the QoS targets. This component models the system as a network of queues whose model parameters are obtained via system monitoring and load prediction models. Monitoring data can be obtained via regular monitoring tools or by Cloud monitoring services such as Amazon CloudWatch[4]. The queueing network model considered by the load predictor and performance modeler is depicted in Figure 2. End-users in the model are represented by the generated requests, whereas application provisioner and application instances are the processing stations for these requests.

Application provisioner is modeled to have a M/M/∞ request queueing station. On the other hand each virtualized application instance has a $M/M/1/k$ queue. Therefore, interarrival and service time distributions are exponentially distributed during each specific analysis interval.

The k parameter, queue size, is defined according to the negotiated service time (T_s) and execution time of a single request (T_r), according to Equation 1. If number of requests in a VM exceeds k, the request is rejected by the admission control system and thus not forwarded to the application

[4]http://aws.amazon.com/cloudwatch

provisioner. This guarantees that requests are either rejected or served in a time acceptable by clients.

$$k = \lfloor \frac{T_s}{T_r} \rfloor \quad (1)$$

The proposed model is built only with information that is made available about the infrastructure: because IaaS providers typically do not disclose information about specific hardware in use, as well as information about network connections and network topology in the data center, *load predictor and performance modeler cannot make assumptions about the low-level physical infrastructure*. Thus, these aspects are abstracted in the model.

Moreover, the proposed model assumes that application instances have the same hardware and software configuration, hence they deliver the same performance. It can be achieved in practice with proper configuration and management of VMs, either via a hypervisor such as Xen [8] or via higher level virtual environment managers such as OpenNebula [9] and VMware vSphere[5]. Virtual machines with different capacities might also be deployed in the system. In this case, the provisioner has to decide when to deploy VMs with different capacity, and this topic is subject of future research.

When workload analyzer updates the estimation of arrival rate, the load predictor and performance modeler checks whether current pool of virtualized application instances are sufficient to meet QoS. To make this verification, it first obtains current service times for each application instance, which is used along with the estimated arrival rate to predict the overall request response time, rejection rate, resource utilization, and maximum number of VMs. If response time or rejection is estimated to be below QoS, or if the utilization is predicted to be below a minimal utilization threshold, the number of VM instances serving applications is updated according to Algorithm 1.

To summarize, the system updates number of required virtualized application instances m depending on current QoS, current average request execution time, and resources utilization. Afterward, it models the system as a network of queues (see Figure 2) and calculates average T_r and $Rej(G_s)$ for a given value of m. If QoS or utilization rate is not met, m is recalculated and the process is repeated. Moreover, load predictor and performance modeler keeps track of minimum and maximum values of m that were tested, in order to avoid the system to try a number of virtualized application instances that either has been tested before or whose value is known to be not sufficient based on previous tested values. It prevents loops in the process.

Computing time of Algorithm 1 is dominated by the execution of the repeat loop between lines 4 and 22: all the operations inside and outside the loop are computed in constant time. Number of iterations in the loop depends on finding the number of required virtualized application instances: maximum number of application instances possible is dependent on

[5]http://www.vmware.com/products/vsphere

Algorithm 1: Adaptive VM provisioning.

Data: QoS metrics: T_s and $Rej(G_s)$
Data: T_m: monitored average request execution time
Data: k: application instance queue size
Data: λ: expected arrival rate of requests
Data: MaxVMs: maximum number of VMs allowed
Result: m: number of application instances able to meet QoS

1 $m \leftarrow$ current number of application instances;
2 $min \leftarrow 1$;
3 $max \leftarrow$ MaxVMs;
4 **repeat**
5 $oldm \leftarrow m$;
6 $\lambda_{s_i} \leftarrow \lambda/m$;
7 $Pr(S_k) \leftarrow$ expected rejection in a $M/M/1/k$ queue given λ_{s_i} and T_m;
8 $T_q \leftarrow$ expected response time in a $M/M/1/k$ queue given $Pr(S_k), \lambda_{s_i}$, and T_m;
9 **if** $Pr(S_k)$ *and* T_q *do not meet QoS* **then**
10 $m \leftarrow m + m/2$;
11 $min \leftarrow m + 1$;
12 **if** $m > max$ **then**
13 $m \leftarrow max$;
14 **end**
15 **else if** *utilization is below threshold* **then**
16 $max \leftarrow m$;
17 $m \leftarrow min + (max - min)/2$;
18 **if** $m \leq min$ **then**
19 $m \leftarrow oldm$;
20 **end**
21 **end**
22 **until** $oldm = m$;
23 **return** m;

both policy applied by the application provider and its previous negotiation with IaaS provider, and minimum number of virtualized application instances is updated during execution: if a given number of virtualized application instances m is not enough to meet QoS, minimum number of virtualized application instances is set as $m + 1$ and further search start from such a value. Space complexity of the algorithm is constant, because decision loop in the algorithm depends only on the maximum, minimum, and current number of virtualized application instances.

C. Application Provisioner

Requests that are accepted by the admission control mechanism of the SaaS provider are subject to application provisioning. They are forwarded to a virtualized application instance, which is able to process the request, following a round-robin strategy. In cases where expected service or response time has lower variability, this strategy may be enough to ensure an even load balance among virtualized application instances at a low monitoring cost. If this is not the case, IaaS provider-

supplied tools, such as Amazon Load-Balancer and GoGrid Controller may be used to help application provisioner keeping load among virtualized application instances balanced.

VM and application provisioning is performed by the application provisioner component based on the estimated number of application instances calculated by the load predictor and performance modeler: if utilization of data center resources is low, application provisioner is directed to destroy some application instances. In this case, the first virtualized application instances to be destroyed are the idle ones. If the number of idle virtualized application instances is smaller than the number of instances to be destroyed, the instances with smaller number of requests in progress are chosen to be destroyed. However, they are not immediately destroyed. Instead, they stop receiving further incoming requests and are destroyed only when running requests finish.

On the other hand, if the arrival rate is expected to increase or the QoS is not being met, application provisioner is directed to create more virtualized application instances. In this case, the application provisioner firstly looks for virtualized application instances selected to be destroyed but that still are processing applications and removes them from the list of instances to be destroyed until the number of required instances is reached. If the number of virtualized application instances created was found to be insufficient, new virtualized instance with the same characteristics (VM configuration) as the existing instances are requested to the data center's resource provisioner. Decision on which physical host receives the VM is made by the Cloud specific resource provisioner and is out of the scope of this paper.

Together, the three components of our adaptive Cloud provisioning mechanism—application provisioner, workload analyzer, and load predictor and performance modeler—are able to dynamically and proactively adapt number of virtual instances available to applications in such a way that service times and rejection rates are kept below a negotiated level at the same time resources are efficiently allocated to the service.

V. Performance Evaluation

In this section, we present the experiments aimed at evaluating the proposed adaptive Cloud provisioning mechanism. CloudSim [10] discrete-event Cloud simulation was used to model the Cloud environment. Next, we describe both the simulation set up and the workloads used for the evaluation.

A. Simulation Set Up

The simulated model is composed of one Cloud data center containing 1000 hosts. Each host has two quad-core processors and 16GB of RAM. This data center hosts the application instances and the mechanism. Simulation model also contains one broker generating requests representing several users and modeled after two common usage scenario for Cloud services, which are detailed in the next section.

Virtual machines for the applications require one core and 2GB of RAM. Because resource provisioning is not the focus of this work, we assume a simple load-balance policy for resource provisioning, where new VMs are created, if possible, in the host with fewer running virtualized application instances. Instances are assigned to an idle core from a physical host when they are created, so there is no time-sharing of CPUs between virtual machines.

Output metrics collected for each scenario are: average response time (T_r) of accepted requests; standard deviation of service times among all accepted requests; minimum and maximum number of virtualized application instances running in a single time; VM hours, which we define as the sum of the wall clock time of each instantiated application, from its creation to its destruction; number of requests whose response time violated QoS; percentage of requests rejected; and resources utilization rate, which we define as the rate between the sum of time actually spent by each virtualized application instance to process requests and the VM hours. Simulation of each scenario was repeated 10 times, and we report the average for each output metric.

Utilization of VM hours as a metric for VM utilization and cost allows us to provide a measurement for cost that is independent from pricing policies applied by specific IaaS Cloud vendors and thus makes the results more generally applicable.

B. Simulation Scenarios

As stated before, two usage scenarios for Cloud services were modeled in our experiments. The first scenario consists of a web service workload, which is composed of a large number of requests requiring small processing capacity (such as retrieval of web pages or on-line business transactions) from the application instances. This scenario is referred to as *web* throughout this section. The second scenario considers an application type that deals with processing of computationally intensive requests. In this scenario, the workload is composed of fewer requests, but T_r is larger when compared to the web workload. This case represents utilization of Cloud-based IT resources for scientific applications such as image rendering and protein folding. This scenario is referred to as *scientific* throughout this section.

Because in these experiments both workloads are based on models, we apply a time-based prediction model for them. More powerful techniques such as Quadratic Response Surface Model (QRSM) [11] and Autoregressive Moving Average with Exogenous Inputs Model (ARMAX) [12] can deliver good prediction of real workloads and execution traces. Exploring such workloads and prediction techniques is part of our future work. However, we are able to show that even the simple estimation presented in this section is able to improve QoS and resources utilization compared to static provisioning policies.

The proposed provisioning strategy is compared to static mechanisms of Cloud provisioning, where a fixed number of instances is made available to execute the same workloads. It allowed us not only to have a base strategy to compare our provisioning mechanism with, but also to evaluate how effective is our strategy in decreasing utilization of Cloud resources. Number of virtualized application instances used

TABLE II
MINIMUM AND MAXIMUM NUMBER OF REQUESTS PER SECONDS ON EACH WEEK DAY IN THE WEB WORKLOAD.

	Requests per second	
week day	**maximum**	**minimum**
Sunday	900	400
Monday	1000	500
Tuesday	1200	500
Wednesday	1200	500
Thursday	1200	500
Friday	1200	500
Saturday	1000	500

in the static policy was determined after execution of our provisioning strategy, so we were able to determine the biggest and the smallest static data center for each scenario. For the web workload, simulations were run for data centers with 50, 75, 100, 125, and 150 virtualized application instances. For the scientific workload, data centers have 15, 30, 45, 60, and 75 instances. In all the cases, the same admission control from the adaptive provisioning is used, therefore requests are either served on time or rejected. Output collected is the same for both static provisioning policies and adaptive provisioning policy simulations.

Next, we detail the generation process of each workload.

1) Web Workload: In the web workload, G_s is modeled after a simplified version of the traces of access to English Wikipedia pages published by Urdaneta *et al.* [13]. Incoming request rate in this workload varies depending on the day of the week and the hour of the day. In our simplified version of the traces, there is a 12-hour difference between the peak (at noon) and the trough (at midnight) in number of arriving requests, and requests are received by the data center in intervals of 60 seconds. Minimum and maximum arrival rates for each week day are presented in Table II. The average number of requests r for a specific time of the day is computed according to Equation 2, and the standard deviation is 5% of such a value.

$$r = R_{min} + (R_{max} - R_{min}) sin(\frac{\pi t}{86400}) \quad (2)$$

In Equation 2, R_{min} and R_{max} are respectively the minimum and maximum number of requests of the considered day of the week, given by Table II, and t is the time in seconds since the start of the current day. A day has 86400 seconds, and this is the denominator in the fraction in Equation 2.

Figure 3 presents the average number of requests received by the service provider during one week simulations generated according to the method described in this section.

Workload analyzer predicts requests arrival rate for the web workload by dividing each day into six periods. The six periods considered are 11:30 a.m. to 12:30 p.m. (peak activity in terms of requests arrival), 12:30 p.m. to 4 p.m. and 4 p.m. to 8 p.m. (decreasing activity), 8 p.m. to 2 a.m. (lowest activity period), and 2 a.m. to 7 a.m., and 7 a.m. to 11:30 a.m. (increasing activity).

Each request requires 100 ms to be processed in an idle server. To insert heterogeneity in the request processing time, we added a uniformly-generated value between 0% and 10%

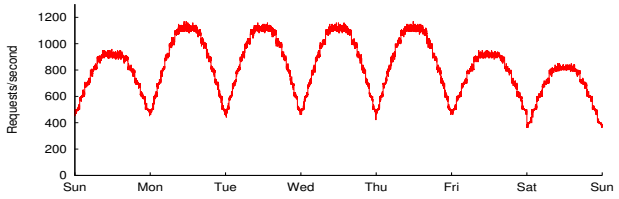

Fig. 3. Average number of requests received by the data center in the web workload.

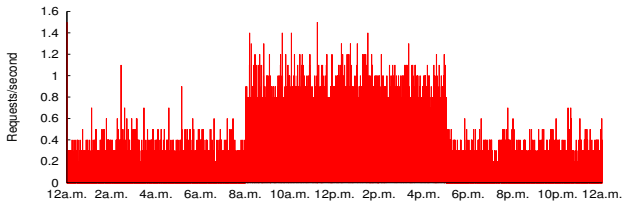

Fig. 4. Average number of requests received by the data center in the scientific workload.

to the processing time for each request. Maximum response time was set to 250 ms and maximum rejection was set to 0%, i.e., the system is required to serve all requests. Minimum utilization of resources was set to 80%. Simulation of this scenario consists in one week of requests being submitted to the data center, starting at Monday 12 a.m.

2) Scientific workload: The scientific workload consists of submission of requests for execution of computationally intensive tasks in a service provider. Arrival of requests is modeled based on the workload for Bag-of-Tasks (BoT) grid applications defined by Iosup *et al.* [14]. According to this workload model, interarrival time of a BoT job in peak time (between 8 a.m. and 5 p.m.), and number of requests received in a 30 minutes period in off-peak time follow Weibull distributions with parameters (4.25, 7.86) and (1.79, 24.16), respectively. In the latter case, we assume that jobs arrive in equal intervals inside the 30 minutes period. Because in the BoT model a request may represent more than one task to be executed, we multiplied the number of arriving requests in a given time by the BoT size class, which is defined in the model as a Weibull distribution with parameters (1.76, 2.11). Figure 4 presents the average number of requests received by the service provider on each second during one day simulations.

Workload analyzer predicts arrival rate for this scenario as follows. For the peak time, the mode of the interarrival time (7.379 seconds) is used to estimate arrival rate, whereas the mode for the size class (which results in 1.309 tasks per BoT job) is used to estimate number of requests on each interarrival. Because these parameters are based on Weibull distributions, the system is subject to high variability in both interarrival interval and number of tasks. High task arrival rates may lead to service rejections. To compensate it, estimated number of tasks is increased by 20%, and this value is used to estimate arrival rate. During off-peak time, arrival rate is estimated based on the mode of the daily cycle, as defined by

the workload (15.298 requests per 30 minutes interval). This value is multiplied by a factor of 2.6 so eventual arrival rates that are bigger than the estimated do not compromise QoS.

In this workload, each request requires 300 seconds to be processed in an instance. Similarly to the method applied for the web workload, we added variability in the request processing time by increasing the processing time for each request by a uniformly-generated value between 0% and 10%. Maximum acceptable response time is 700 s and maximum rejection is 0%. Minimum utilization of resources is 80%. Because the scientific workload presents a daily behavior, in opposite to the web workload that presents a weekly behavior, simulation of the scientific scenario consists in one day of requests, starting at 12 a.m. and generated according to the workload described in this section.

C. Results

Next, experimental results are presented. In figures 5 and 6, our proposed mechanism is labeled as *Adaptive*, whereas static provisioning policies are labeled as *Static-**, where * represents the number of VMs in the specific scenario.

1) Web Workload: Figure 5 shows the results of the experiment for the web workload. In average, each simulation of the scenario generated 500.12 million requests in the one-week simulation time. Our adaptive provisioning mechanism avoided rejection of requests by dynamically increasing number of virtualized application instances in peak times and reducing it in off-peak times. Number of virtualized application instances active in the data center varied between 55 and 153 (278.2%) with our policy, and rejection rate was insignificant. Number of instances created by our policy in peak time (153) is bigger than the actually required, as a static data center with 150 VMs also avoids rejections in peak time. Number of virtualized application instances created can be decreased by allowing some requests to be rejected.

The number of hours of VMs required by our policy for one-week execution is equivalent to keeping 111 virtualized application instances active 24/7, even though this number of instances would not be able to cope with peak demand: with 125 statically allocated application instances to a data center, there are still 2% of rejections.

If the system were statically provisioned and able to cope with the peak demand, as provisioned by our mechanism, it would require approximately 150 instances 24/7, but in this case total utilization of resources would be below 60%. Therefore, our proposed system meets QoS and reduces host VM utilization in 26% (in terms of number of hours) at the same time that it completely avoids rejection of requests. Moreover, the system was also able to keep utilization rate of resources above the negotiated 80%.

Another interesting trend that Figure 5(b) shows is that even data centers with high rejection rates may have low utilization of resources. This is because the number of deployed application instances is more than enough to handle off-peak traffic, therefore there are idle resources in some periods of the day, but these resources are not enough to serve peak time demand.

2) Scientific Workload: Figure 6 shows the results of the experiment for the scientific workload. In average, each simulation of the scenario generated 8286 requests in one-day simulation time. In line with the web experiment, our mechanism could meet negotiated QoS of requests by dynamically increasing number of application instances in peak times and reducing it in off-peak times. Number of application instances active in the data center varied from 13 VMs in the off-peak time to 80 in peak time.

The number of hours of VMs required for one-day execution with our policy is equivalent of keeping 40 virtualized application instances active 24 hours. However, in this case number of instances is not enough in peak time, as observed for the case of 45 machines statically allocated, which has a rejection rate of 31.7%. If the system were statically provisioned and able to cope with the peak demand, it would require 75 instances available 24 hours with total data center utilization of only 42%. Therefore, in the scientific scenario our proposed system reduces VM utilization (in terms of number of hours) in 46% at the same time that it completely avoids rejection of requests and keeps service time in negotiated levels.

As Figure 6 shows, utilization of the data center with the use of our adaptive policy was slightly below the required level (78% rather than 80%). Moreover, number of requests in peak time was also over estimated, as in the previous experiment. Both effects in this workload were caused by the prediction policy adopted by load predictor and performance modeler, which deliberately over estimates arrival rate (by the use of factors, as explained in Section V-B2) to compensate for sudden increase in number of requests caused by the probability distribution that represents the workload. As in the previous case, this can be avoided if some rejection of requests is allowed in the system.

VI. Related Work

Quiroz *et al.* [3] propose a mechanism for dynamic VM provisioning in IaaS data centers based on clustering. In such a work, it is necessary not only to determine the number of virtualized application instances but also their types. In our approach, type of instance is not part of the problem, thus deployed instances can always be used to serve requests.

Zhu and Agrawal [15] propose a dynamic mechanism for VM provisioning based on control theory considering user budget. However, such an approach considers reconfiguration of available virtual instances (increase or decrease their capacity) and not increasing/decreasing number of instances for a customer, conversely to our approach that applies the latter approach for VM provisioning.

Bi *et al.* [16] propose a model for provisioning multi-tier applications in Cloud data centers based on queueing networks. However, such a model does not perform recalculation of number of required VMs based on expected load and monitored performance as does our approach.

Chieu *et al.* [17] propose a reactive algorithm for dynamic VM provisioning of PaaS and SaaS applications, whereas our

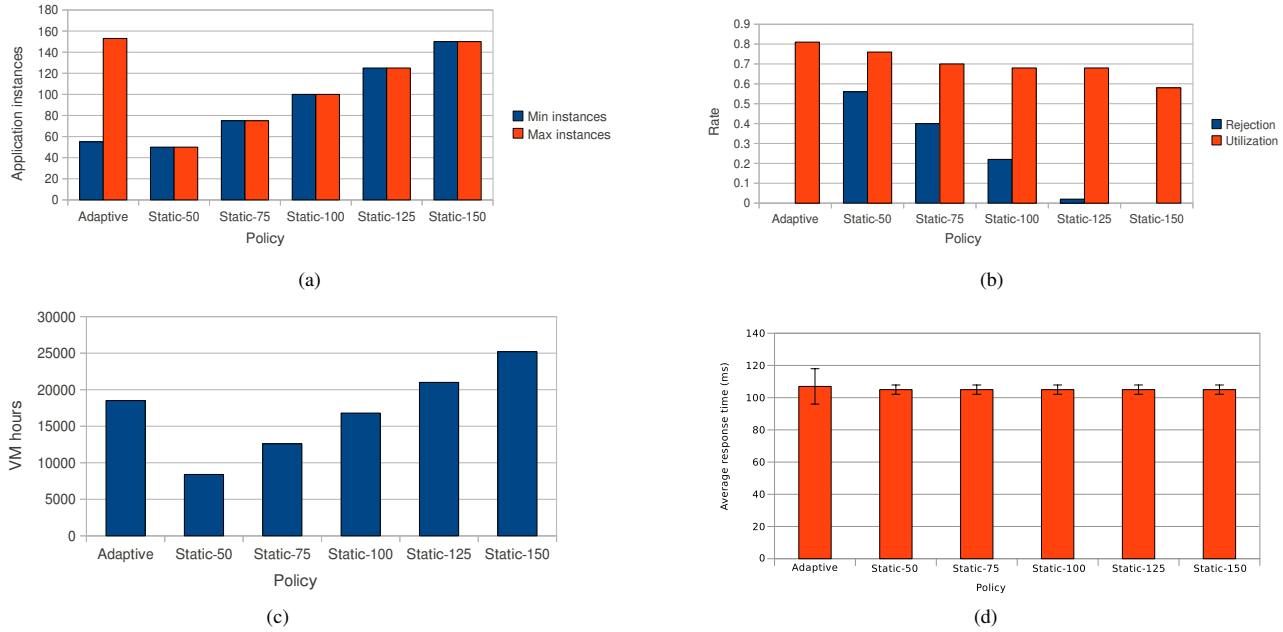

Fig. 5. Results for the Wikipedia workload (web) scenario (a) number of virtualized application instances (b) data center utilization and requests rejection rates (c) VM hours (d) average response time and standard deviation. Admission control mechanism in place in all scenarios successfully prevented QoS violations.

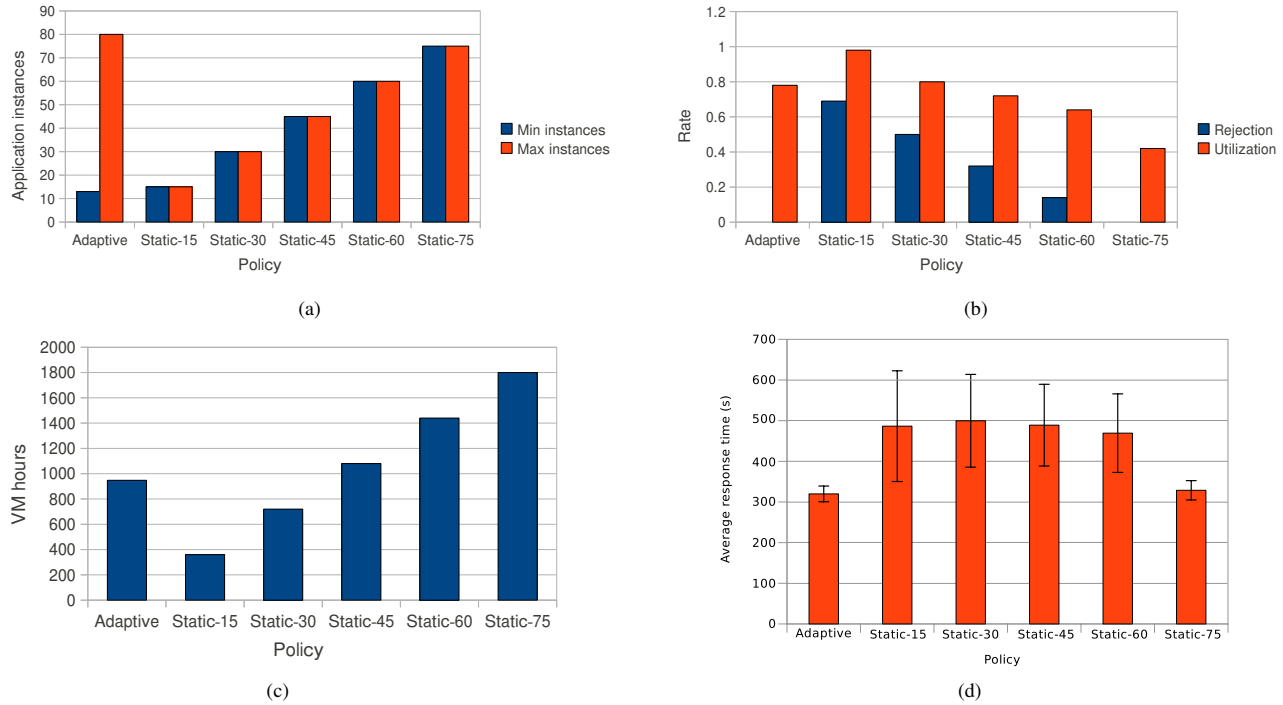

Fig. 6. Results for the Grid Workload Archive's BoT workload model (scientific) scenario (a) number of virtualized application instances (b) data center utilization and requests rejection rates (c) VM hours (d) average response time and standard deviation. Admission control mechanism in place in all scenarios successfully prevented QoS violations.

approach is proactive in the sense that number of instances is changed based on the expected arrival rate of requests.

Lee *et al.* [18] propose a queueing network to model SaaS mashup applications whose goal is to maximize profit or reduce costs of the SaaS provider by finding an optimal number of instances for the application. Rodero-Merino *et al.* [19] propose a system called Claudia, where provisioning is based on performance indicators and elasticity rules defined by users. In both approaches number of instances vary reactively to incoming request rate, whereas our model proactively applies adaptive provisioning to deliver negotiated QoS to requests whose request arrival rate varies along with the time.

Jung *et al.* [20] propose the Mistral system, which performs management at data center host level to manage power consumption of resources and performance of applications. However, this approach requires access to the physical infrastructure, which typical IaaS providers do not provide to consumers. Therefore, Mistral is suitable to be applied in scenarios where both infrastructure and application are offered by the same provider, while our approach can be both applied in the same case or in cases where IaaS and PaaS/SaaS providers are different organizations.

VII. Conclusions and Future Work

Although adoption of Cloud computing platforms as application provisioning environments has several benefits, there are still complexities obstructing the smooth provisioning and delivery of application services in such environments.

To counter those complexities related to application provisioning over Clouds, this paper presented an adaptive provisioning mechanism for delivery of resources to SaaS applications. The mechanism uses analytical performance (queueing system model) and workload information to drive decisions of an application provisioner. The proposed approach is able to model the infrastructure using only information that IaaS providers make available to customers and monitoring data from running VMs. The goal of the model is to meet QoS targets related to service time and rejection rate of requests and utilization of available resources.

Simulation-based experimental results using production workload models indicated that the proposed provisioning technique can detect changes in workload intensity (arrival pattern and resource demands) that occur over time and allocate resources accordingly to achieve application QoS targets.

As future work, modeling and decision-making processes used by the mechanism will be improved to support not only changes in number of VMs but also changes in each VM capacity. Moreover, we intend to improve the queueing model to allow modeling composite services and access to Cloud storage. We also want to adapt more comprehensive prediction techniques (such as QRSM [11] and ARMAX [12]) to handle prediction for arbitrary service workloads.

Finally, we will extend the model to support other QoS parameters such as deadline and incentive/budget to ensure that high-priority requests are served first in case of intense competition for resources and limited resource availability. For the latter scenario, we will also address the problem of SLA management for trade-offs of QoS between different requests, potentially with different priorities and incentives, in order to effectively manage QoS violations.

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 Generation Computer Systems*, vol. 25, no. 6, pp. 599–616, 2009.

[2] J. Varia, "Best practices in architecting cloud applications in the AWS Cloud," in *Cloud Computing: Principles and Paradigms*. Wiley, 2011.

[3] A. Quiroz, H. Kim, M. Parashar, N. Gnanasambandam, and N. Sharma, "Towards autonomic workload provisioning for enterprise grids and clouds," in *Proceedings of the 10th IEEE/ACM International Conference on Grid Computing (GRID'09)*, 2009.

[4] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[5] D. A. Menascé, V. A. Almeida, and L. W. Dowdy, *Performance by Design: Computer Capacity Planning by Example*. Prentice Hall, 2004.

[6] Y. C. Lee and A. Zomaya, "Rescheduling for reliable job completion with the support of clouds," *Future Generation Computer Systems*, vol. 26, no. 8, pp. 1192–1199, 2010.

[7] D. P. Anderson, "Public computing: Reconnecting people to science," in *Proceedings of the Conference on Shared Knowledge and the Web*, 2003.

[8] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the art of virtualization," in *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP'03)*, 2003.

[9] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster, "Virtual infrastructure management in private and hybrid clouds," *IEEE Internet Computing*, vol. 13, no. 5, pp. 14–22, 2009.

[10] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. D. Rose, and R. Buyya, "CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, 2011.

[11] R. H. Myers, D. C. Montgomery, and C. M. Anderson-Cook, *Response Surface Methodology*. Wiley, 2009.

[12] J. V. Candy, *Model-based Signal Processing*. Wiley, 2006.

[13] G. Urdaneta, G. Pierre, and M. van Steen, "Wikipedia workload analysis for decentralized hosting," *Computer Networks*, vol. 53, no. 11, pp. 1830–1845, 2009.

[14] A. Iosup, O. Sonmez, S. Anoep, and D. Epema, "The performance of bags-of-tasks in large-scale distributed systems," in *Proceedings of the 17th International Symposium on High Performance Distributed Computing (HPDC'08)*, 2008.

[15] Q. Zhu and G. Agrawal, "Resource provisioning with budget constraints for adaptive applications in cloud environments," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing (HPDC'10)*, 2010.

[16] J. Bi, Z. Zhu, R. Tian, and Q. Wang, "Dynamic provisioning modeling for virtualized multi-tier applications in cloud data center," in *Proceedings of the 3rd International Conference on Cloud Computing (CLOUD'10)*, 2010.

[17] T. C. Chieu, A. Mohindra, A. A. Karve, and A. Segal, "Dynamic scaling of web applications in a virtualized cloud computing environment," in *Proceedings of the 6th International Conference on e-Business Engineering (ICEBE'09)*, 2009.

[18] Y. C. Lee, C. Wang, A. Y. Zomaya, and B. B. Zhou, "Profit-driven service request scheduling in clouds," in *Proceedings of the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid'10)*, 2010.

[19] L. Rodero-Merino, L. M. Vaquero, V. Gil, F. Galán, J. Fontán, R. S. Montero, and I. M. Llorente, "From infrastructure delivery to service management in clouds," *Future Generation Computer Systems*, vol. 26, no. 8, pp. 1226–1240, 2010.

[20] G. Jung, M. A. Hiltunen, K. R. Joshi, R. D. Schlichting, and C. Pu, "Mistral: Dynamically managing power, performance, and adaptation cost in cloud infrastructures," in *Proceedings of the 30th International Conference on Distributed Computing Systems (ICDCS'10)*, 2010.