Dynamic Resource Provisioning for Cloud-Based Gaming Infrastructures

MORENO MARZOLLA, STEFANO FERRETTI, and GABRIELE D'ANGELO, University of Bologna

Modern massively multiplayer online games (MMOGs) allow hundreds of thousands of players to interact with a large, dynamic virtual world. Implementing a scalable MMOG service is challenging because the system is subject to high workload variability, but nevertheless must always operate under very strict quality of service (QoS) requirements. Traditionally, MMOG services are implemented as large dedicated IT infrastructures with aggressive over-provisioning of resources in order to cope with the worst-case workload scenario. In this article we address the problem of building a large-scale, multitier MMOG service using resources provided by a Cloud computing infrastructure. The Cloud paradigm allows customers to request as many resources as they need using a pay-as-you-go model. We harness this paradigm by proposing a dynamic provisioning algorithm, which can resize the resource pool of a MMOG service to adapt to workload variability and maintain a response time below a given threshold. We use a queuing network performance model to quickly estimate the system response time for different configurations. The performance model is used within a greedy algorithm to compute the minimum number of servers to be allocated on each tier in order to satisfy the system response time constraint. Numerical experiments are used to validate the effectiveness of the proposed approach.

Categories and Subject Descriptors: C.4 [**Computer Systems Organization**]: Performance of Systems; K.6.2 [**Management of Computing and Information Systems**]: Installation Management—*Pricing and resource allocation*; C.2.4 [**Computer Communication Networks**]: Distributed Systems—*Distributed applications*

General Terms: Algorithms, Performance

Additional Key Words and Phrases: Cloud computing, dynamic scalability, massively multiplayer online games, performance modeling

ACM Reference Format:
Marzolla, M., Ferretti, S., and D'Angelo, G. 2012. Dynamic resource provisioning for Cloud-based gaming infrastructures. ACM Comput. Entertain. 10, 3, Article 4 (November 2012), 20 pages.
DOI = 10.1145/2381876.2381880 http://doi.acm.org/10.1145/2381876.2381880

1. INTRODUCTION

Modern massively multiplayer online games (MMOGs) are large-scale distributed systems serving millions of concurrent users which interact in real-time with a large, dynamic virtual world. An important characteristic of online games is their strict performance requirements, especially response time [Chen et al. 2006; Dick et al. 2005]: depending on the type of game, the response time to ensure a responsive play may range from tens of milliseconds for first- person shooter action games, to a few seconds for role-playing games. MMOGs are usually implemented as client-server architectures,

Authors' address: M. Marzolla, S. Ferretti, and G. D'Angelo, Università di Bologna, Dipartimento di Scienze dell'Informazione, Mura Anteo Zamboni 7, I-40127 Bologna, Italy; email: marzolla@cs.unibo.it; sferrett@cs.unibo.it; g.dangelo@unibo.it.

DOI 10.1145/2381876.2381880 http://doi.acm.org/10.1145/2381876.2381880

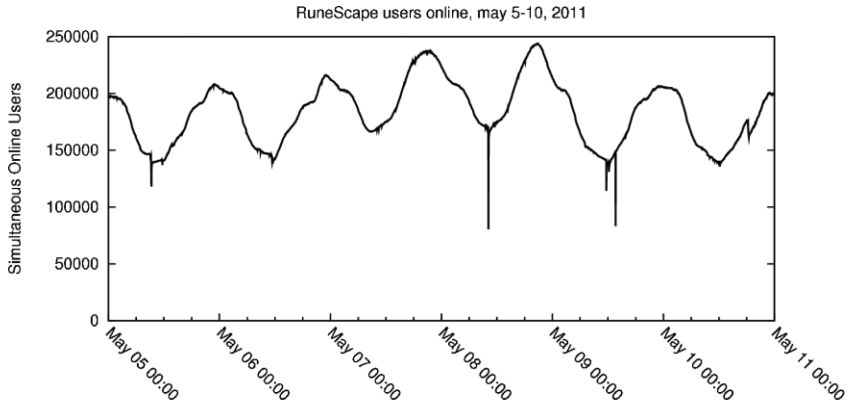

Fig. 1. Number of simultaneous RuneScape players during the period May 5 to May 10, 2011.

where the server is responsible for maintaining the global state of the virtual play field in response to users (clients) requests. Since the client-server network connection can easily become a significant source of latency, which the game service providers cannot control, MMOG services are often hosted on multiple, geographically distributed datacenters, so that each user can be redirected to the "faster" (i.e., best connected) one. Binding users to datacenters can be performed at run-time by taking into consideration the measured end-to-end connection quality; when network connections or datacenters become congested, users can be migrated to different servers. Multiple data centers also help to address scalability problems, because the virtual world can be partitioned or replicated across the available servers [Cai et al. 2002; Palazzi et al. 2006].

Nevertheless, scalability problems still exist because the system is subject to high variability of the workload: as the number of concurrent users increases, so does the system response time. To keep the response time within appropriate levels, a resource overprovision policy is often adopted, which is based on statically allocating enough resources to cope with the worst-case scenario. This policy is inefficient because it can lead to a largely suboptimal utilization of the hosting environment resources. In fact, since the worst-case scenario rarely happens, a significant fraction of the allocated resources may remain unused at run-time.

To support this claim, we have monitored the number of online users of the RuneScape MMOG [Jagex Ltd. 2011]. RuneScape is a fantasy MMOG where players can travel across the fictional medieval realm of Gielinor. There is not a fixed storyline: players can decide to combat monsters, practice skills or interact competitively or cooperatively with other players. From the technical point of view, the RuneScape client code is written in Java and runs inside a browser. Players connect to servers which are located in different world regions to reduce the communication latency.

We collected the total number of online players as displayed on the http://www.runescape.com/ Web page using a sampling period of two minutes, during the period May to October 2011. Figure 1 shows a subset of the data,[1] from May 5 to May 10, 2011. A daily pattern is clearly visible; during peak hours, more than 200.000 players are connected to the system (the load is split across the regional servers), while about 110.000 players are active during off-peak hours. Hence the daily churn (number of players leaving/joining the system during the day) is about 100.000 users. For the dataset shown in Figure 1 the maximum number of online players is about 230.000 and the minimum is about 130.000. In this scenario, statical resource provisioning based

[1]The full dataset is available at http://pads.cs.unibo.it/.

on the average load results in system overloaded roughly half the time; provisioning for the worst-case results in a massive resource underutilization.

In recent years, the *Cloud computing* paradigm has emerged as an affordable way to cope with scalability issues. Specifically, Cloud computing allows customers to "rent" computing and storage resources, and only pay for what they actually use. Many Cloud providers employ the *utility computing* model, where computing, storage or application resources are billed like regular utility services (such as electricity).

Cloud computing can be very helpful in deploying large-scale MMOG services, for at least two reasons: (i) game service providers do not have to provision for peak load limit; and (ii) the MMOG service can be augmented to request more resources during peak periods, and release them when no longer needed.

In this article we propose a dynamic resource provisioning strategy for large-scale MMOG services implemented on top of Cloud infrastructures providing resources using the Infrastructure as a Service (IaaS) model. An IaaS Cloud provides low-level computing and storage capabilities where the customer can run arbitrary software, including operating systems and applications [Zhang et al. 2010]. IaaS Clouds make heavy use of virtualization techniques in order to fragment and allocate physical resources to customers. For example, computing power is usually provided as virtual machine (VM) instances executing on some physical server. The same physical server can host multiple VM instances, each instance exposing part of the capabilities of the server. Virtualization allows Cloud providers to offer a set of homogeneous (virtual) resources, characterized by a choice of key parameters (CPU speed, memory size, and so on) among which customers can choose. The underlying physical infrastructure can be made of heterogeneous resources, and is not directly exposed to the clients.

We consider a large-scale gaming service built over multiple, geographically distributed data centers, each one providing resources on demand using Cloud infrastructures. Each data center hosts a three-tier system, which handles one partition of the virtual world. The goal is to ensure that the system response time R at each datacenter is kept below a predefined threshold $R_{\max}$. As already observed, the value of $R_{\max}$ depends on the kind of game: action games, where fast interaction between players and the environment is essential, require very low response times (tens of milliseconds); strategy games, on the other hand, can deliver a satisfactory gameplay with response times on the order of a few seconds. Therefore, the game operator defines the value for $R_{\max}$ that provides the best experience to its users [Palazzi et al. 2006].

Thanks to the underlying Cloud, it is possible to add (remove) computing nodes to (from) any tier. Hence we satisfy the response time constraint by dynamically adding or removing server instances where necessary. From the point of view of the game service provider, the cost of a data center depends on the number of nodes allocated there. We thus aim at minimizing the total cost of a data center by minimizing the number of allocated nodes such that the response time satisfies the constraint $R < R_{\max}$.

We assume that the MMOG service is capable of transparently reconfiguring itself when the resource pool of each datacenter is altered. We also assume that only one datacenter is in charge of handling a given region of the virtual world in a specific game session. This is actually what happens in many real implementations of MMOGs [Jagex Ltd. 2011]. A consequence of such an approach is that, for instance, most issues related to the consistency management of the game state can be easily handled. In general, this does not hold when multiple servers working on the same virtual region of the same game session are geographically distributed. In fact, in this case other factors might influence the level of provided responsiveness, for example, the synchronization algorithm, the distribution of clients connected to distributed servers [Mauve et al. 2004; Palazzi et al. 2006].

To achieve the goal above, we enhance the gaming service hosted in each Cloud with two additional components, called *monitor* and *planner*. The monitor is a passive observer that collects run-time performance metrics; in particular, the monitor measures the current system response time R, and triggers the planner when the response time deviates from the threshold $R_{\max}$. The planner is responsible for computing the optimal (minimum) number of nodes to allocate at each tier so that the response time is maintained below the threshold.

Since the planner must operate at run-time, it is extremely important that a new configuration is computed quickly. To do so, we use a greedy strategy in which one node is added (or removed) at a time from a suitably chosen tier. The planner uses a queueing network (QN) performance model to identify the tier to alter, and to estimate the system response time after each change; the parameters needed to analyze the performance model are those collected by the monitor. Thanks to the QN model, the planner can efficiently compute complex reconfiguration changes, which involve the addition or removal of multiple nodes from different tiers. In fact, it is well known that adding more servers to the bottleneck tier only is not guaranteed to improve the overall system performance, as the bottleneck may shift to other tiers.

This article is structured as follows. In Section 2 we compare our approach with the relevant literature. In Section 3 we describe the high-level architecture of the MMOG services we consider, and precisely formulate the dynamic provisioning problem as an optimization problem. Section 4 describes our proposed dynamic reconfiguration algorithm. The effectiveness of the solution is evaluated in Section 5 by means of numerical experiments. Finally, conclusions and future research directions are illustrated in Section 6.

2. RELATED WORK

Despite the fact that Cloud computing in general is an active research topic, to the best of our knowledge, the problem of QoS provision in Cloud computing environments is only recently receiving attention. In this section we briefly review a few papers on this topic that show some analogy with our approach to development of QoS-aware Cloud-based applications.

Two kind of approaches have been considered in the literature: model-based and measurement-based. Model-based solutions use performance models to drive the adaptation step. In [Li et al. [2009] the authors describe a method for achieving resource optimization at run-time by using performance models in the development and deployment of the applications running in the Cloud. Their approach is based on a layered queuing network (LQN) performance model that predicts the effect of simultaneous changes (such as resource allocation/deallocation) to many decision variables (throughputs, mean service delays, etc.). In Ranjan et al. [2002] the authors consider an approximation of a multitier architecture as a $G/G/N$ queueing center with general interarrival time, general service time distribution, and N identical servers, under heavy load. In Urgaonkar et al. [2008] a general, k-tier system is modeled as a chain of $G/G/1$ queueing centers. We also mention a recent work that addresses the same topic considered here, that is, dynamic resource provisioning in MMOG infrastructures. In Nae et al. [2010] the authors first introduce a combined processor, network and memory load model specifically tailored to MMOG architectures, which is used together with a neural-network based predictor in order to anticipate fluctuations without the need to accurately monitor them in real-time.

Our approach differs from those mentioned above because it is not limited to MMOG systems, but can also be easily applied to generic multitier services with an arbitrary number of tiers (see Section 6 for details).

As to measurement-based approaches, they basically consist in periodically monitoring the QoS provided by the Cloud and react to its performances by tuning the amount of resources exploited for hosting the service. For instance, in Ferretti et al. [2010], a reconfiguration approach is exploited that dynamically adds/releases resources devoted to support a given service, based on the amount of SLA violations that occur during the service utilization, in order to avoid that the rate of these violations surpass a predetermined threshold. The work proposed in this article copes with the same problem. However, our solution is based on a novel and different approach with respect to Ferretti et al. [2010]. In fact, in the former, the authors consider a single-tier service and propose a purely reactive system in which a reconfiguration is triggered based on the experienced QoS violations. On the other hand, in this article we consider a multitier MMOG architecture for which identifying a new configuration, which satisfies the QoS constraint, is more challenging. The reason is that in multitiered applications it is necessary to identify both the tier(s) from which resources should added/removed, and also compute how many resource instances to add/remove. To this aim, we propose a model-based approach based on a simple heuristic that uses a queuing network performance model to quickly estimate the system response time for different configurations.

Other work focuses mostly on issues related to the definition and monitoring of the SLAs in a Cloud computing environment, and does not address issues of QoS enforcement and resource optimization. In Spillner and Schill [2009] the authors present a methodology for adding or adjusting the values of nonfunctional properties in service descriptions, depending on the service run-time behavior, and then dynamically deriving adjusted SLA template constraints. In contrast, in our proposal, the SLA constraint is given, and the service must be modified at run-time to provide the necessary QoS level. Issues related to the SLA monitoring are presented in Korn et al. [2009]. In that paper the authors introduce the notion of service level management authority (SLMA), a third independent party that monitors the SLA and facilitates the interaction between the Cloud vendor and the end customer. This approach differs from ours, as in the solution we propose the monitoring facilities are implemented by a component of our middleware platform rather than by an external entity. (However it is worth noting that due to the modularity of our architecture, one could investigate the possibility of integrating a SLMA in our solution).

3. PROBLEM FORMULATION AND SYSTEM MODEL

We consider a large-scale distributed infrastructure to support MMOGs, as shown in Figure 2. The system has three types of actors: (i) users (game players), who interact with a virtual world by controlling software avatars; (ii) the *MMOG service*, which is responsible for maintaining the game state and executing all necessary interactions between players and the virtual world; and (iii) the *resource provider*, who is responsible for providing computational and storage resources on demand to the operator of the MMOG service.

The MMOG service maintains state information about a single virtual universe, or *metaverse*. In order to ensure scalability, many existing MMOG implementations partition the metaverse across multiple servers [Kumar et al. 2008]. As the size and complexity of game increase, it is reasonable to split the metaverse across multiple data centers, each one handling a partition. We assume that the virtual playfield is partitioned into nonoverlapping (or only partially overlapping) zones [Cai et al. 2002]. Human players control virtual avatars which move and interact inside a zone; avatars are not bound to a single zone, but can move freely over the whole virtual world.

The MMOG service operator maintains the state of each zone through a *zone controller*, which is a software component responsible for handling all interactions between players and/or the virtual world inside a single zone. To enhance scalability and

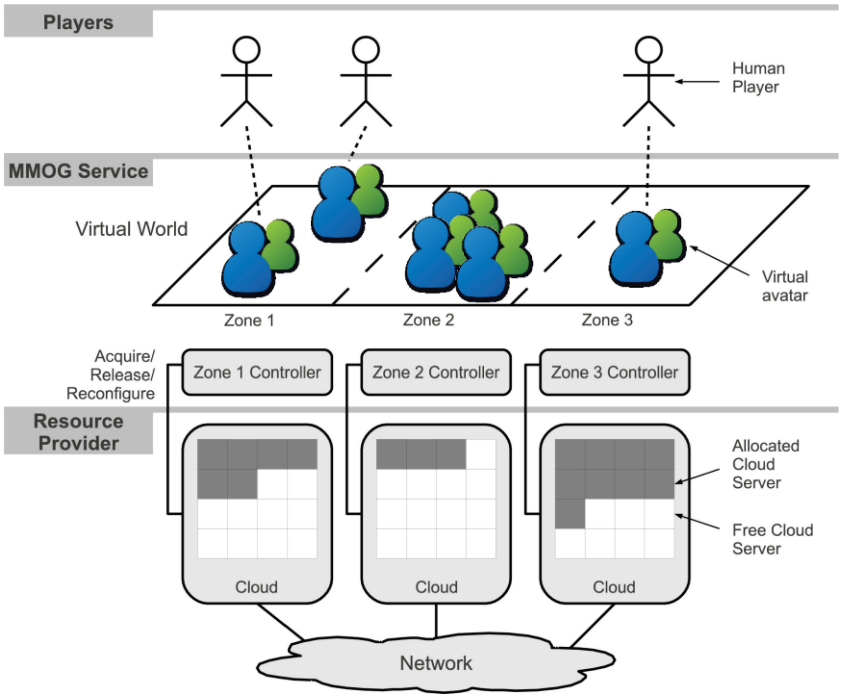

Fig. 2. High-level architecture of a distributed Cloud-based gaming infrastructure.

distribute the workload, each zone is handled by a separate hardware infrastructure hosted on different data centers. Given that communication between datacenters may incur significant delays, it is important that interactions across neighboring zones is minimized. For example, each partition may hold a collection of "islands," such that all interactions happen within the collection, while players can jump from one "island" to another (possibly joining a new server on a different datacenter). We assume that the MMOG service makes crossing a partition (zone) a seamless operation.

Depending on the (virtual) mobility pattern of each player, some areas of the metaverse may become crowded, while others may become less populated. In order to cope with this variability, each zone controller is hosted on resources provided and operated by a IaaS Cloud infrastructure. The Cloud provider is in general a separate entity that rents computational and storage resources to the customers on a pay-as-you go model [Zhang et al. 2010]. This means that the game operator can request additional servers and/or storage space at any time, and release them when no longer needed. Thus, the game operator can request more resources when the workload on a zone increases in order to keep the response time perceived by players below a predefined maximum value. When the workload decreases, the game operator can release surplus resources in order to reduce costs.

We now describe in detail the structure of a zone controller. We consider the multi-tier architecture described in Hsiao and Yuan [2005], which is shown in Figure 3. The *firewall* represents the entry point and allows traffic filtering. The first layer contains a set of *gateways*, which are responsible for handling basic gaming protocol checking and verification. The *cell servers* are responsible for managing the virtual world and its evolution; each server controls a small area inside the virtual zone assigned to the corresponding data center. Finally, the *database servers* are used to store persistent

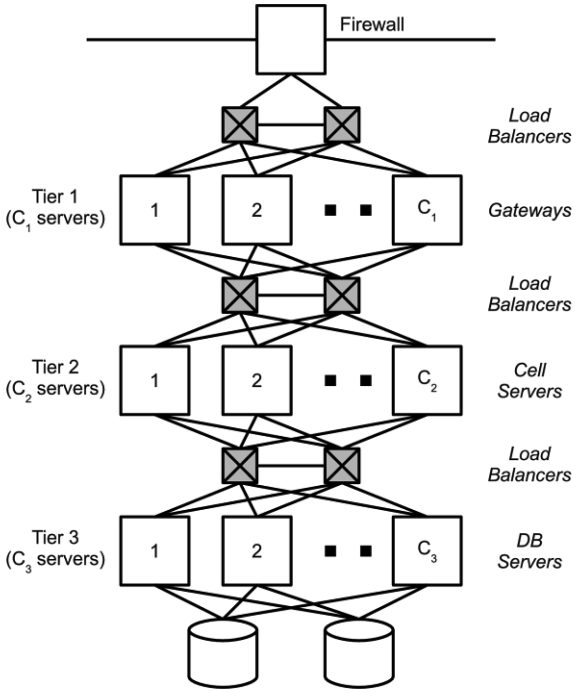

Fig. 3. Multitier architecture of the MMOG server hosted by a single data center.

game state information. *Load balancers* are used to evenly distribute requests among servers across layers.

It is reasonable to expect that all servers within the same layer are homogeneous, that is, have approximately the same configuration (CPU, memory, disk) and the same performance. While our dynamic reconfiguration algorithm could be extended to cope with nonhomogeneous layers, this scenario would be of little practical interest in this context, since sophisticated load balancing techniques would be required to distribute the load in such a way that slower machines do not become bottlenecks. Besides, as observed in Section 1, Cloud providers allow customers to request server instances with specific characteristics (CPU speed, number of cores, memory size, disk space), and hence the homogeneity assumption is the norm rather than the exception. Of course, we allow different layers to require different server configurations; for example, the DB servers' layer may require servers with larger and faster disks to sustain a larger I/O rate.

As observed above, the MMOG service is subject to workload variability because game players can join and leave the system at any time, and can migrate from one zone to another. In this article we address the problem of ensuring that the average response time experienced by a player does not exceed some predefined maximum value. To solve this problem we propose a model-based algorithm for dynamic reconfiguration of the MMOG service; our algorithm uses a simple QN model to identify the bottleneck tier(s), and compute the number of servers to add/remove. The reconfiguration algorithm is executed by each zone controller independently from each other.

Formally, the *system configuration* C is defined as a vector with three elements $\boldsymbol{C} = (C_1, C_2, C_3)$ where C_i represents the number of hosts allocated to tier $i = 1, 2, 3$. Thus, C_1 is the number of gateways, C_2 the number of cell servers, and C_3 the number of

DB servers which are currently instantiated. The aim of the reconfiguration algorithm is to compute the configuration $\mathbf{C}$ with minimum total number of servers such that the (estimated) system response time is less than $R_{\max}$. Formally, we aim at solving the following optimization problem:

$$\begin{array}{ll} \text{minimize} & (C_1 + C_2 + C_3) \\ \text{subject to} & R(\boldsymbol{C}) < R_{\max} \\ & \boldsymbol{C} = (C_1, C_2, C_3) \\ & C_i \in \{1, 2, \ldots, \} \quad i = 1, 2, 3 \end{array} \tag{1}$$

where $R(\mathbf{C})$ is the estimated system response time with configuration $\boldsymbol{C} = (C_1, C_2, C_3)$. In general, $R(\mathbf{C})$ does not depend on the configuration $\mathbf{C}$ only, but also on other parameters, as will be discussed in the next section.

Since the total number of hosts allocated at each Cloud data center is proportional to the total cost of the resources provided by that datacenter, by reducing the total number of hosts we also reduce the cost of the MMOG infrastructure.

4. RECONFIGURATION ALGORITHM

In this section we describe the details of the reconfiguration algorithm. We enhance the gaming service shown in Figure 3 with two additional components, called *monitor* and *planner*. Each data center has its own monitor and planner, and each data center executes the autonomic reconfiguration algorithm described in the following, independently of the others.

The monitor is a passive observer which collects run-time performance metrics; in particular, the monitor measures the system response time R. When R deviates from $R_{\max}$, the monitor triggers the planner, which computes the optimal (minimum) number of nodes to allocate at each tier so that the response time is maintained below the threshold. The new configuration is computed by finding an approximate solution to the optimization problem (1).

Since the planner must operate at run-time, it is extremely important that a new configuration is computed quickly. To do so, we use a greedy strategy in which one node is added (or removed) at a time from a suitably chosen tier. In general, it might be necessary to add (or remove) multiple hosts from different tiers with a single reconfiguration step; furthermore, the identification of a new configuration must be done efficiently in order to quickly adapt to the workload fluctuations. This rules out the simple solution in which hosts are added or removed by trial and error, and the impact of each new configuration is directly measured on the running system.

The planner uses a QN performance model to identify the tier to alter, and to estimate the system response time after each change; the parameters needed to analyze the performance model are those collected by the monitor. Thanks to the QN model, the planner can efficiently compute complex reconfiguration changes, which involve the addition or removal of multiple nodes from different tiers.

4.1. The Monitor

The monitor is a passive observer, which collects run-time statistics on a single datacenter. Specifically, the monitor collects the following parameters:

—the average system response time R;
—the average system throughput X (rate at which requests, i.e., game events generated by clients connected to that node, are processed);
—the tier utilizations $\boldsymbol{U} = (U_1, U_2, U_3)$, where U_i is the utilization of tier i.

Since these parameters may be subject to high variability, it is necessary to apply suitable smoothing functions to the raw data before they can actually be used. A simple approach is to collect multiple samples, and compute a moving average over a time window of length W. For a comprehensive treatment of the state change detection problem; see Gustafsson [2000].

The system administrators must define two additional thresholds: R_{low} and R_{high}, such that $R_{\text{low}} < R_{\text{high}} < R_{\text{max}}$. The monitor checks whether the average response time R computed over the last time window is less than R_{low} or greater than R_{high} . In either case, the planner is invoked. If $R < R_{\text{low}}$, the planner tries to deallocate resources from underutilized tiers; if $R > R_{\text{high}}$, the planner adds resources to the bottleneck tiers in order to reduce the system response time before it hits the threshold R_{max} (details are given in the next section).

The values of R_{low}, R_{high}, and W are, in general, system- and application-dependent, and impact both the frequency of reconfiguration and the sensitivity of the reconfiguration algorithm. If R_{low} is small, over-provisioning may happen, because unused resources are relinquished only when $R < R_{\text{low}}$, which may rarely happen. Similarly, if R_{high} is large, violations of the "hard" response time limit R_{max} may happen before the system has the opportunity to react. Finally, low values of W imply that the system can react quickly to surges in the number of concurrent users, at the cost of increasing the frequency of reconfigurations, and thus increasing the overhead of the adaptation process. As a rule of thumb, we empirically found that setting $R_{\text{high}} = 0.9R_{\text{max}}$ and $R_{\text{low}} = 0.8R_{\text{max}}$ is a reasonable setting; the value of W should be chosen according to the workload fluctuation speed. For services that exhibit the daily pattern typical of human activities with a period of approximately 24 hours, values of W in the range 5 to 20 minutes are appropriate.

The control loop just described is shown in Algorithm 1. The system administrator must choose an initial configuration $\mathbf{C}$ (line 1), after which an infinite loop is used to collect monitoring data and reconfigure the system when necessary. The functions Acquire and Release are provided by the planner component, and are described in the next section.

ALGORITHM 1: QoS-Aware Reconfiguration Algorithm

Require: $R_{\text{low}} < R_{\text{high}} < R_{\text{max}}$: Thresholds

```
1:  Let C be the initial configuration
2:  loop
3:      Monitor the system and compute R; X; U
4:      C' := C
5:      if (R > R_high) then
6:          C' := Acquire(C; U; X; R)
7:      else if (R < R_low) then
8:          C' := Release(C; U; X; R)
9:      end if
10:     if a new configuration C' ≠ C has been found then
11:         Apply the new configuration C' to the system
12:         C := C'
13:     end if
14: end loop
```

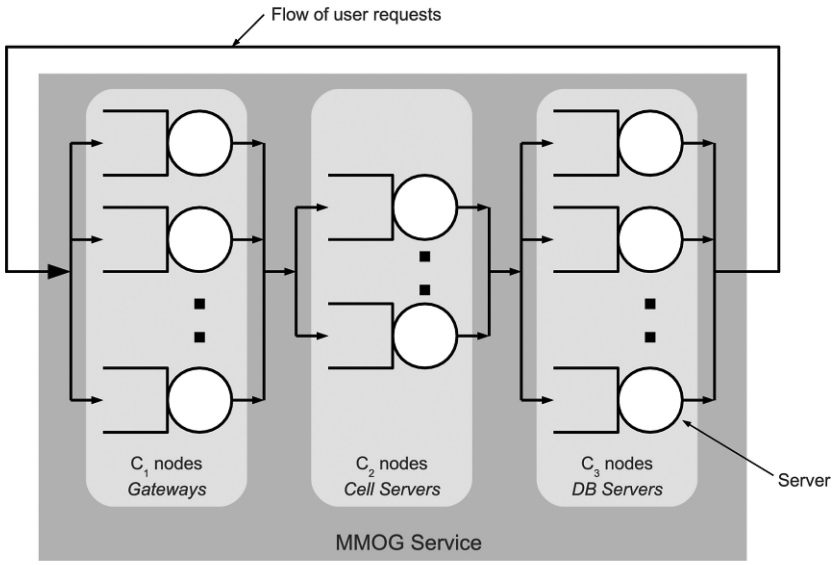

Fig. 4. Queueing model of a datacenter.

4.2. The Planner

The planner is responsible for identifying a new configuration $\boldsymbol{C} = (C_1, C_2, C_3)$ as a solution of the optimization problem 1. The planner uses an QN performance model to estimate the system response time of different configurations.

A data center is modeled using the single-class, product-form closed QN shown in Figure 4. For any configuration $\boldsymbol{C} = (C_1, C_2, C_3)$, the model has $(C_1 + C_2 + C_3)$ service centers organized in three tiers with C_1, C_2, and C_3 servers each, respectively. A fixed population of N requests continuously circulate in the system, N being the total number of players currently connected to the system. We allow the value of N to change over time, as users join and leave the system.

Each server is approximated as a -/G/1—PS center with general service time and processor sharing (PS) service discipline. The only requirement on the service time distribution is that it must have rational Laplace transform; this requirement is not very restrictive, since it includes all distributions which can be expressed as a network of exponential stages; exponential, hyperexponential, and hypoexponential service time distributions all have rational Laplace transform. The PS service discipline closely approximates the scheduling policies of actual servers, where requests are processed in round-robin fashion, each receiving service for a small quantum of time. Under these assumptions, the QN belongs to the class of BCMP networks that have *product-form solution* [Baskett et al. 1975, Balsamo 2000], which basically means that average performance measures (response times, utilizations, and so on) can be computed efficiently. This is important because the model must be analyzed at run-time, so we favor computational efficiency over model accuracy. Also consider that precise performance estimations not only require an accurate model, but also detailed and accurate model parameters, which can be acquired only with an invasive and time-consuming monitoring activity.

We assume that the workload can be balanced across the servers of the same tier, such that all nodes have, on average, approximately equal utilization. We remark that this assumption is not strictly necessary, since the QN model can be analyzed with arbitrary parameters. However, our dynamic scalability algorithm relies on the assumption that

Table I. Symbols Used in this Article

R	Measured system response time
X	Measured system throughput
U_i	Measured utilization of tier i
$R_{\max}$	Maximum allowed response time
R_{high}	Upper response time limit
R_{low}	Lower response time limit
N	Number of users online
C_i	Number of nodes at tier i
D_i	Estimated Total service demand of tier i
U_i	Estimated average utilization of nodes at tier

the load on the bottleneck tier can be reduced by spreading the requests over a larger number of servers. This requirement is not strict, that is, we allow unbalance to happen; however, it is reasonable to assume that on average all servers on the same tier have similar utilization. This is in fact the case for existing MMOGs based on a multitiered architecture [Cai et al. 2002; Hsiao and Yuan 2005]. If the MMOG service is unable to spread the load evenly, then "hot spots" may arise and the system will eventually hit scalability limits, no matter how many resources are allocated.

To analyze the network of Figure 4, the following additional parameters are needed: (i) an estimation of the number N of requests currently in the system; and (ii) the aggregate service demand at each tier.

The number of concurrent requests N (which can be seen as the number of active players that periodically generate game events during the game evolution) can be computed from the measured system response time R and throughput X using Little's law [Little 1961] as:

$$N = XR \tag{2}$$

The service demand D_i of tier i is the cumulative time spent by a request in any server of tier i. According to the utilization law [Lazowska et al. 1984], the service demand on an individual server of tier i can be expressed as U_i/X. If the utilization and throughput were measured when the system configuration was $\mathbf{C} = (C_1, C_2, C_3)$, then the total service demand at tier i is

$$D_i = C_i \times U_i / X, \quad i = 1, 2, 3. \tag{3}$$

Table I summarizes all symbols used in this article.

For product-form QNs, the mean value analysis (MVA) algorithm [Reiser and Lavenberg 1980] can be used to compute individual device utilizations and system response time on a closed network, with n requests given the average queue lengths with $n-1$ requests. Thus, starting from an empty network, MVA computes solutions for populations $1, 2, \ldots, N$. A closed network with N requests and K queueing centers can be analyzed using the general MVA algorithm in time O(N K). However, since all servers in each tier are equivalent, the network of Figure 4 can be analyzed in time O(N) using the specialized MVA implementation of Algorithm 2.

If we are not interested in exact performance values, it is possible to compute upper and lower asymptotic bounds on the system throughput and response time using the balanced system bounds (BSB) algorithm [Zahorjan et al. 1982]. The computational complexity is greatly reduced, as the network of Figure 4 can be analyzed in time O(1) (independent of the number of requests N). It has already been observed [Marzolla and Mirandola 2010] that for dynamic reconfiguration applications, the MVA algorithm only provides marginal advantages over the simpler and more computationally efficient

ALGORITHM 2: MVA(**C; D; N**) $\rightarrow \langle \mathbf{U}; R \rangle$

Require: **C** configuration to analyze
Require: $\mathbf{D} = (D_1; D_2; D_3)$ total service demands of tiers
Require: N number of active users
Ensure: $\mathbf{U} = (U_1; U_2; U_3)$ estimated utilization of tiers
Ensure: R estimated system response time

1: **for** k : = 1 **to** 3 **do**
2: $Q_k := 0$ {Mean queue length at any tier k server}
3: end for
4: **for** n := 1 **to** N **do**
5: **for** k : = 1 **to** 3 **do**
6: $R_k := D_k(1 + Q_k)$
7: **end for**
8: $R_s = \sum_{k=1}^{3} R_k C_k$ {System response time}
9: $X_s := n/R_s$ {System Throughput}
10: **for** k := 1 to 3 **do**
11: $Q_k := X_s \times R_k$
12: **end for**
13: **end for**
14: **for** k : = 1 **to** 3 **do**
15: $U_k := X_s \times D_k$
16: **end for**
17: **Return** $\langle \mathbf{U}; R \rangle$

computation of BSB. Therefore, we estimate the system response time as the average value of the upper and lower bounds provided by the BSB algorithm.

Algorithm 3 computes upper and lower bounds on the system throughput (X^+ and X^- respectively) and upper and lower bounds on the system response time (R^+ and R^- respectively). The response time R is then estimated as the average value of the upper and lower bounds ($R^+ + R^-$)/2 (line 9). The parameter $\mathbf{D} = (D_1, D_2, D_3)$ represents the total service demand vector of tiers 1–3 as computed using Eq. (3), using the measurements performed by the monitor.

Acquiring new resources. When the measured response time R is greater than R_{high}, we need to allocate new resources to improve the system responsiveness. Let **C** be the current configuration at the time the planner is invoked. The new configuration **C'** is computed by the procedure Acquire() shown in Algorithm 4. The procedure uses a simple greedy strategy to iteratively define $\mathbf{C}'$ starting from **C**. At each iteration, the QN model is used to estimate the utilization of all tiers, and the system response time (line 5). The bottleneck tier $b \in \{1, 2, 3\}$ is identified as the tier whose nodes have higher utilization (line 7). Then, a single host is added to the bottleneck tier (line 8). These steps are repeated until the estimated response time is less than $(R_{\text{high}} + R_{\text{low}})/2$; at that point, the configuration $\mathbf{C}'$ becomes the new system configuration: new hosts are allocated from the Cloud service, and all tiers of the MMOG server are reconfigured accordingly.

Releasing resources. When the measured response time R is less than R_{low}, we try to release hosts. Let **C** be the current configuration at the time the planner is invoked. The new configuration $\mathbf{C}'$ is computed by procedure Release() shown in Algorithm 5. Again, procedure Release() uses an iterative greedy strategy to compute $\mathbf{C}'$ from **C**. At each iteration we consider the set $S \in \{1, 2, 3\}$ of tiers with more than one node (line 4);

ALGORITHM 3: BSB($\mathbf{C}$; $\mathbf{D}$; N) $\rightarrow$ $\langle \mathbf{U}; \mathrm{R} \rangle$

Require: $\mathbf{C}$ configuration to analyze
Require: $\mathrm{D} = (D_1; D_2; D_3;)$ total service demands of tiers
Require: N number of active users
Ensure: $\mathbf{U} = (U_1; U_2; U_3)$ estimated utilization of tiers
Ensure: R estimated system response time

1: $D_{\max} := \max\{D_1/C_1; D_2/C_2; D_3/C_3\}$ {Maximum demand of a server}
2: $D_{\text{tot}} := (D_1 + D_2 + D_3)$ {Total service demand on all servers}
3: $D_{\text{ave}} := D_{\text{tot}}/(C_1 + C_2 + C_3)$ {Average service demand of a server}
4: $X^- := N/(D_{\text{tot}} + (N-1) \times D_{\max})$ {Lower bound on system throughput}
5: $X^+ := \min\{1/D_{\max}; N/(D_{\text{tot}} + (N-1) \times D_{\text{ave}})\}$ {Upper bound on system throughput}
6: $R^- := \max\{N \times D_{\max}; D_{\text{tot}} + (N-1) \times D_{\text{ave}}\}$ {Lower bound on system response time}
7: $R^+ := D_{\text{tot}} + (N-1) \times D_{\max}$ {Lower bound on system response time}
8: $X := (X^+ + X^-)/2$ {Estimated system throughput}
9: $R := (R^+ + R^-)/2$ {Estimated system response time}
10: **for** i : = 1 to 3 **do**
11: $U_i := X \times D_i$
12: **end for**
13: **Return** $\langle \mathbf{U}; R \rangle$

ALGORITHM 4: Acquire($\mathbf{C}$; $\mathbf{U}$; $\mathbf{X}$; $\mathbf{R}$) $\rightarrow$ $\mathbf{C}'$

Require: $\mathbf{C}$ Current system configuration
Require: $\mathbf{U}$ Measured utilizations
Require: X; R Measured system throughput and response time
Ensure : $\mathbf{C}'$ New system configuration

1: $\mathbf{C}' := \mathbf{C}$
2: Compute N using Eq. (2)
3: Compute $\mathbf{D} = (D_1; D_2; D_3)$ using Eq. (3)
4: **repeat**
5: $\langle \mathbf{U}; R \rangle := \text{BSB}(\mathbf{C}'; \mathbf{D}; N)$ {Evaluate configuration $\mathbf{C}'$}
6: **if** $(R \geq (R_{\text{high}} + R_{\text{low}})/2)$ **then**
7: Let b the tier with highest utilization U_b
8: $\mathrm{C}'_b := \mathrm{C}'_b + 1$
9: **end if**
10: **until** $R\,(R_{\text{high}} + R_{\text{low}})\,/\,2$
11: **Return**$\mathbf{C}'$

S represents the set of tiers from which one host could be removed. We identify the tier $u \in S$ with the; lowest utilization, and try to remove one node from tier u (line 8). We estimate the new system response time R using BSB (line 9). If R becomes larger than $(R_{\text{high}} + R_{\text{low}})/2$, we do not deallocate any host from tier u: we thus remove u from S (line 11), and iterate again. The process stops when S becomes empty, which means that either (i) all tiers have exactly one host; or (ii) it is not possible to remove any host from any tier without causing the estimated system response time to become larger than $(R_{\text{high}} + R_{\text{low}})/2$.

It is worth noticing that the architecture of the monitor and planner components is very simple and does not require any major modification to the existing MMOG service.

Furthermore, the monitor and planner can be implemented using Cloud-based services and therefore in an elastic way.

4.3. Computational Cost

Both Algorithms 4 and 5 execute a number of iterations in which a single host is added to or removed from a single tier. The cost of each iteration is O(1), since performance evaluation of the three-tier queueing system using BSB can be done in constant time. It should be observed that a more accurate estimation of performance parameters using the MVA algorithm would require O(N) operations, where N is the number of active users at the time the network is analyzed. Since N can become quite large (thousands of active players are common in most MMOGs), MVA is not appropriate for our application.

ALGORITHM 5: Release($\mathbf{C}$; $\mathbf{U}$; $\mathbf{X}$; $\mathbf{R}$) $\rightarrow$ $\mathbf{C}'$

Require: $\mathbf{C}$ Current system configuration
Require: $\mathbf{U}$ Measured utilizations
Require: $\mathbf{X}$; $\mathbf{R}$ Measured system throughput and response time
Ensure : $\mathbf{C}'$ New system configuration

1: $\mathbf{C}' := \mathbf{C}$
2: Compute N using Eq. (2)
3: Compute $D = (D_1; D_2; D_3)$ using Eq. (3)
4: $S := \{i | C'_i > 1\}$
5: $\langle \mathbf{U}; R \rangle := \text{BSB}(\mathbf{C}'; \mathbf{D}; N)$ {Evaluate configuration $\mathbf{C}'$}
6: **while** $(S \neq \emptyset)$ **do**
7: Let u the tier in S with lowest utilization
8: $C'_u := C'_u + 1$ {Try to reduce tier u}
9: $\langle \mathbf{U}; R \rangle := \text{BSB}(\mathbf{C}'; \mathbf{D}; N)$ {Evaluate configuration $\mathbf{C}'$}
10: **if** $(R \geq (R_{\text{high}} + R_{\text{low}})/2)$ **then**
11: $C'_u := C'_u + 1$ {Rollback to old configuration}
12: $S := S \backslash \{u\}$ {Remove u from the set S}
13: **else**
14: $S := \{i | C'_i > 1\}$
15: **end if**
16: **end while**
17: **Return** $\mathbf{C}'$

Given the current system configuration $\boldsymbol{C} = (C_1, C_2, C_3)$, the new configuration $\boldsymbol{C}' = (C'_1, C'_2, C'_3)$ identified by the procedure Acquire() has the property that $C'_i \geq C_i$ for all $i = 1, 2, 3$. Thus, the computational cost of Acquire() is $O(\sum_{i=1}^{3} (C'_i - C_i))$. Similarly, given the current configuration $\mathbf{C}$, the new configuration $\mathbf{C}'$ returned by procedure Release() is such that $C'_i \leq C_i$ for all i. The worst case happens when the initial configuration (C_1, C_2, C_3) is reduced to (1, 1, 1); that is, all hosts (except one host for each tier) are released. Thus, the worst-case computational cost of Release() is $O(\sum_{i=1}^{3} C_i)$.

5. NUMERICAL RESULTS

In this section we evaluate the dynamic reconfiguration strategy described in Section 4. Algorithms 4 and 5 have been implemented in GNU Octave [Eaton 2002], an interpreted language for numerical computations. We performed two sets of experiments:

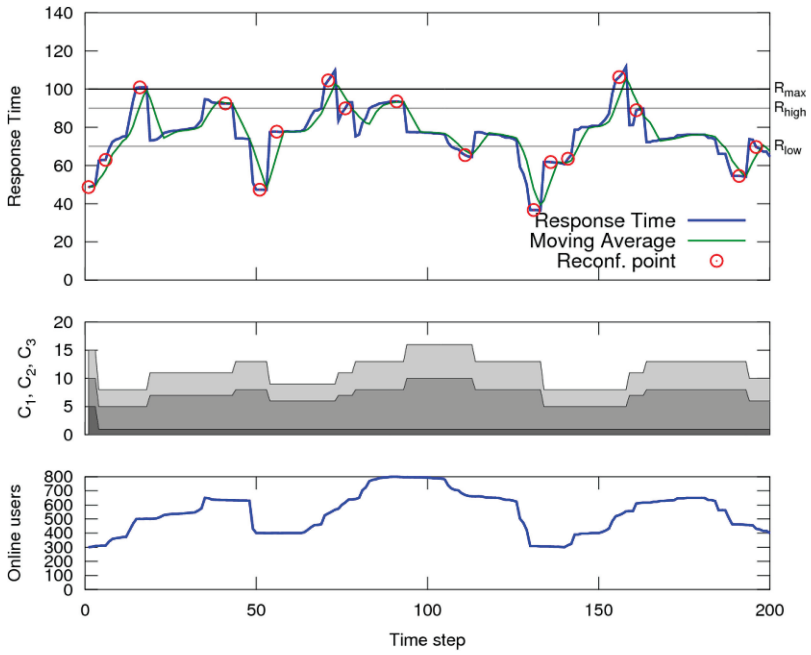

Fig. 5. Simulation result, irregular workload with low churn.

one with a completely synthetic workload, and one using a real workload obtained by monitoring the number of online users as reported by the RuneScape Web site.

Synthetic workloads. We performed a time-stepped simulation of duration $T = 200$ steps. In order to reduce the number of input parameters, we generated a sequence of values for the number of online users N_t at time t, for each $t = 1, \ldots, T$. We defined fixed values for the average service times S_i at tier i as $S_1 = 0.08$, $S_2 = 0.8$, $S_3 = 0.58$. The measured system response time R, throughput X, and tier utilizations U_i are computed using the MVA algorithm on the QN model of Figure 4. In our experiments we set $R_{\max} = 100$, $R_{\text{high}} = 90$ and $R_{\text{low}} = 70$. The planner uses a window of size $W = 5$ time steps. We assume that acquiring/releasing hosts and reconfiguring the system takes some time. Specifically, a reconfiguration completes $\Delta t = 3$ time steps after it has been requested. We evaluate our approach in three different scenarios, using different workloads.

Figures 5 to 8 show the results. Each plot contains three parts: on top we show the observed system response time (thick line), achieved using the dynamic adaptation algorithm described in this article, while the thin line shows the time-averaged system response time over the last W steps. Horizontal lines show the values of $R_{\max}$, R_{high}, and R_{low}, respectively. The times at which a new configuration is applied (reconfiguration points) are also shown. On the middle part we show the number of allocated nodes on each tier: the height of colored bands are the values of C_1, C_2, and C_3, from bottom to top. Finally, the bottom part shows the number N_t of concurrent online users at each time step t.

Real Workload. We performed another experiment by using a real workload collected from RuneScape. We monitored the number of online users as reported in the Web page [Jagex Ltd. 2011], collecting one sample every two minutes over a period of several weeks from May to October 2011. For our test, we consider a subset of the data which consists of tree days, from May 5 to May 7, 2011 (about 2160 data points). We

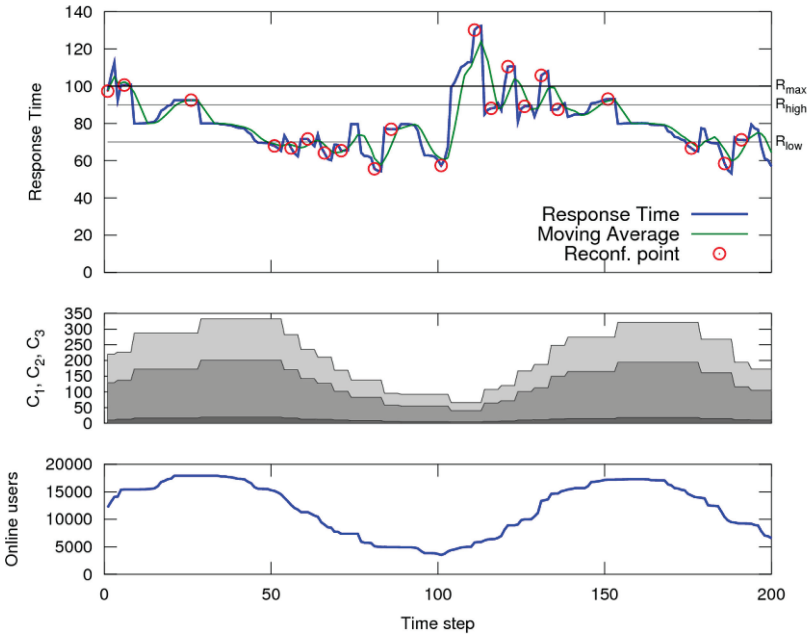

Fig. 6. Simulation result, periodic workload with low frequency.

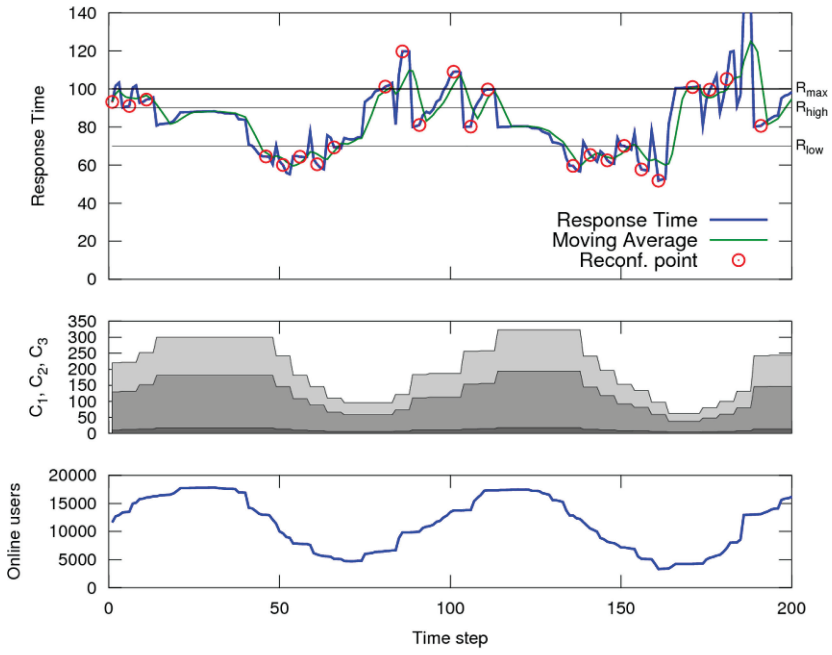

Fig. 7. Simulation result, periodic workload with medium frequency.

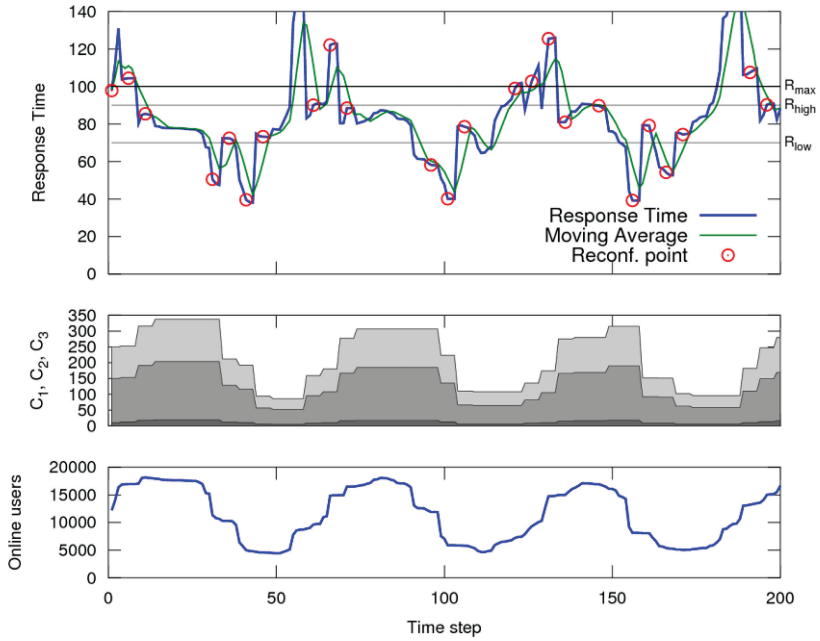

Fig. 8. Simulation result, periodic workload with high frequency (implying high churn).

resampled the data to one data point every 10 minutes, reducing the dataset to about 432 data points. Thus, each simulation step refers to 10 minutes of wall- clock time. We set the same fixed values for the average service times S_i at tier i as in the previous set of experiments ($S_1 = 0.08$, $S_2 = 0.8$, $S_3 = 0.58$). We considered a moving average over $W = 5$ samples, and we assume that a system reconfiguration requires $\Delta t = 2$ time steps (20 minutes of wall-clock time). Threshold have been set as $R_{\max} = 100$, $R_{\text{high}} = 90$, and $R_{\text{low}} = 80$. The simulation result is shown in Figure 9; remarkably, no violation of the service- level agreement (SLA) occurred, as our algorithm was capable of following the fluctuations of the workload.

Discussion. Simulation results are summarized in Table II. The following parameters are reported:

—The figure the result refers to;
—the minimum and maximum number of online users;
—the number of times a new configuration has been applied;
—the number of time steps in which the SLA constraint $R(C) < R_{\max}$ has been violated;
—The minimum, maximum, and total number of servers that have been allocated by the dynamic provisioning algorithm during the simulation run. If $C_i(t)$ is the number of servers allocated at time t at tier $i = 1, 2, 3$, then the minimum number of servers is $\min_t \{C_1(t)+C_2(t)+C_3(t)\}$, the maximum number of servers if $\max_t \{C_1(t)+C_2(t)+C_3(t)\}$, and the total number of servers is $\sum_t (C_1(t) + C_2(t) + C_3(t))$;
—the total number of servers which would have been statically allocated in case of provisioning for the worst-case scenario. This number is simply the maximum number of servers multiplied by the length of the simulation run;
—the ratio between the total number of servers allocated by the dynamic provisioning algorithm and the total number of servers for the worst-case static allocation; lower is better.

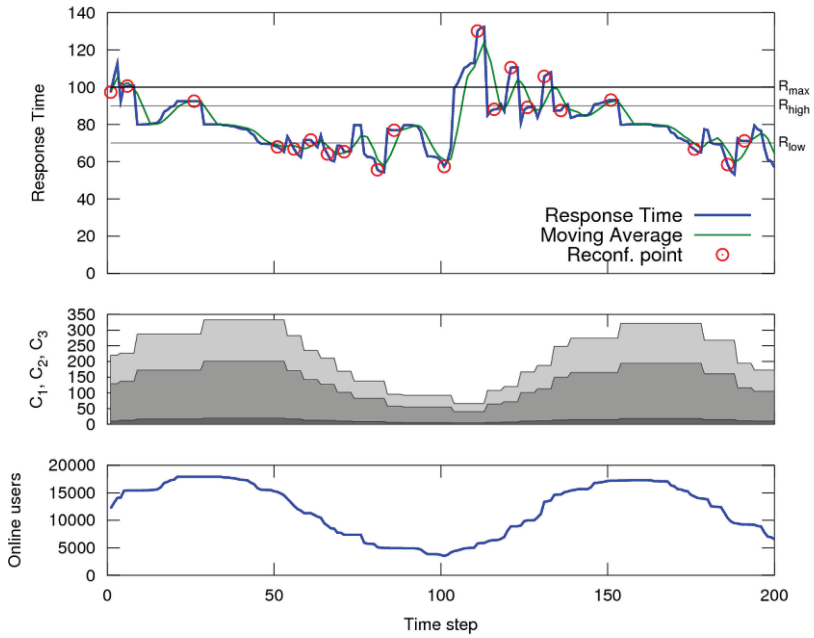

Fig. 9. Simulation result with the real workload; one time step corresponds to 10 minutes of wall clock time.

Table II. Simulation Results

	Online users			SLA	N. Servers (dynamic)			N. Servers	N. Serv. Dynamic/
	min	max	Reconf.	violations	Min	max	tot	(static)	N. Serv. Static
Fig. 5	300	800	17	12	8	16	2295	3200	0.72
Fig. 6	3517	17895	21	22	66	333	45386	66600	0.68
Fig. 7	3319	17809	25	32	63	323	41623	64600	0.64
Fig. 8	4441	18157	26	35	86	338	42785	67600	0.63
Fig. 9	134608	229939	23	0	2440	3977	1258126	1590800	0.79

We observe that our algorithm is effective in reducing the number of resources (hosts) which are necessary to satisfy the QoS constraint on the system response time. A clear correlation is seen in Figures 5 to 9 between the number of active sessions and the total number of allocated hosts: as the number of concurrent users increases, so does the system response time, which in turn triggers reconfigurations, resulting in more hosts being added to the appropriate tiers. When the number of concurrent users decreases, servers are deallocated from the tiers.

The number of violations of the SLA, as shown in Table II, is generally quite low; SLA violations happen when there is very high churn, that is, when many users join the system in few time steps. This can be seen by considering Figures 6 to 8, which have an increasingly high workload fluctuation frequency, causing higher churn in a very short time. If the workload fluctuates smoothly, as in Figure 5, the response time is almost always kept below the threshold $R_{\max}$. If larger fluctuations happen, as in Figures 6 to 8, our adaptation algorithm may require some time to react properly.

It is interesting to observe that real workloads for MMOG do not exhibit steep fluctuations, as can be seen on Figure 9. Remarkably, for the real workload, our dynamic provisioning algorithm produces no SLA violation, despite the fact that the controller is invoked every 10 minutes of wall-clock time, and a new configuration requires two

simulation steps (20 minutes) to be applied to the system. The latter parameter is set to an extremely conservative value, yet the results are extremely good.

Finally, the last column of Table II shows that the dynamic provisioning strategy allows a considerable reduction in the number of allocated servers. Recall that the computing resources used by the MMOG operator are provided by a third party Cloud provider; the number of allocated servers is proportional to the cost of the MMOG infrastructure, and is paid by the MMOG operator to the Cloud provider. Our algorithm allows this cost to be significantly reduced, while still providing an appropriate level of QoS to the game players.

6. CONCLUSIONS AND FUTURE WORKS

In this article we described a framework for runtime performance-aware reconfiguration of a distributed Cloud-based MMOG system. We consider a large-scale MMOG service implemented across a geographically distributed data center, each data center providing resources on demand, according to the Cloud computing paradigm. Each Cloud hosts a three-tier system, which handles one partition of the virtual game space. Each data center is passively monitored to detect when the average response time deviates from the threshold $R_{\max}$. When that happens, we reconfigure the data center by adding or removing computing nodes. We use a greedy heuristic to allocate the minimum number of nodes such that the expected response time does not exceed the threshold. Different configurations are evaluated using a product-form QN performance model.

The methodology proposed in this article can be improved along several directions. In this article we assumed that the cost of all Cloud resources is the same; this may not be the case, for example, if a DB server machine needs a different configuration (and thus has a different cost) than a Gateway machine. Thus, we are working towards a more sophisticated optimization problem which takes into account the price of the resources. We are also exploring the use of forecasting techniques as a means to trigger reconfigurations in a proactive way. Another extension of the proposed approach, currently under investigation, is the instrumentation of software clients used by the players. In this way, it would be possible to collect run-time statistics about the gaming experience of each player and to consider the reallocation of gamers among the Tier 1 hosts (i.e., Gateways). This could bring a reduction in the latency experienced by each client, including in the adaptive evaluation process, the whole gaming infrastructure, and therefore, to some extent, improving the gaming experience.

Finally, we remark that the approach described in this article is not limited to MMOG services only, but can be easily extended to any large-scale multitier service for which resources at the different tiers can be dynamically provisioned. Algorithms 1 to 5 require trivial modifications to cope with the general case of t-tiered systems, for any t. As such, our proposal is quite general as it could be applied to, for example, e-commerce sites, online auction services, and similar online data intensive (OLDI) applications [Meisner et al. 2011], which are characterized by response time constraints and are subject to workload fluctuations induced by user-generated queries.

REFERENCES

Balsamo, S. 2000. Product form queueing networks. In *Performance Evaluation: Origins and Directions,* G. Haring et al., Eds., LNCS 1769, Springer, Berlin, 377–401.

Baskett, F., Chandy, K. M., Muntz, R. R., and Palacios, F. G. 1975. Open, closed, and mixed networks of queues with different classes of customers. *J. ACM 22*, 2 (April), 248–260.

Cai, W., Xavier, P., Turner, S. J., and Lee, B.-S. 2002. A scalable architecture for supporting interactive games on the internet. In *Proceedings of the 16th Workshop on Parallel and Distributed Simulation (PADS'02)*, IEEE, Washington, D.C., 60–67.

CHEN, K.-T., HUANG, P., AND LEI, C.-L. 2006. How sensitive are online gamers to network quality? *Commun. ACM 49*, 34–38.

DICK, M., WELLNITZ, O., AND WOLF, L. C. 2005. Analysis of factors affecting players' performance and perception in multiplayer games. In *Proceedings of the 4th Workshop on Network and System Support for Games (NETGAMES 2005)*, ACM, New York, 1–7.

EATON, J. W. 2002. *GNU Octave Manual*. Network Theory Limited.

FERRETTI, S., GHINI, V., PANZIERI, F., PELLEGRINI, M., AND TURRINI, E. 2010. QoS-aware Clouds. In *Proceedings of the IEEE 3rd International Conference on Cloud Computing (CLOUD 2010)*, 321–328.

GUSTAFSSON, F. 2000. *Adaptive Filtering and Change Detection*. John Wiley, Ltd.

HSIAO, T.-Y. AND YUAN, S.-M. 2005. Practical middleware for massively multiplayer online games. *IEEE Internet Comput. 9*, 47–54.

JAGEX LTD. 2011. RuneScape. http://www.runescape.com/.

KORN, A., PELTZ, C., AND MOWBRAY, M. 2009. A service level management authority in the cloud. Tech. rep. HPL-2009-79, HP Laboratories.

KUMAR, S., CHHUGANI, J., KIM, C., KIM, D., NGUYEN, A., DUBEY, P., BIENIA, C., AND KIM, Y. 2008. Second life and the new generation of virtual worlds. *Computer 41*, 9, 46–53.

LAZOWSKA, E. D., ZAHORJAN, J., GRAHAM, G. S., AND SEVCIK, K. C. 1984. *Quantitative System Performance: Computer System Analysis Using Queueing Network Models*. Prentice Hall, Englewood Cliffs, NJ.

LI, J., CHINNECK, J., WOODSIDE, M., LITOIU, M., AND ISZLAI, G. 2009. Performance model driven guarantees and optimization in Clouds. In *Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing (CLOUD'09)*, IEEE, Washington, D.C., 15–22.

LITTLE, J. D. C. 1961. A proof for the queuing formula: $L = \lambda W$. *Oper. Res. 9*, 3, 383–387.

MARZOLLA, M. AND MIRANDOLA, R. 2010. Performanc-aware reconfiguration of software systems. In *Proceedings of the Computer Performance Engineering, 7th European Performance Engineering Workshop (EPEW 2010)*, 23–24.

MAUVE, M., VOGEL, J., HILT, V., AND EFFELSBERG, W. 2004. Local-lag and timewarp: providing consistency for replicated continuous applications. *IEEE Trans. Multimedia 6*, 1, 47–57.

MEISNER, L., SADLER, C. M., BARROSO, L. A., WEBER, W.-D.. AND WENISCH, T. F. 2011. Power management of online data-intensive services. In *Proceedings of the 38th Annual International Symposium on Computer Architecture (ISCA'11)*, ACM, New York, 319–330.

NAE, V., IOSUP, A., AND PRODAN, R. 2010. Dynamic resource provisioning in massively multiplayer online games. *IEEE Trans. Parallel Distrib. Syst. 99*, 1–15.

PALAZZI, C. E., FERRETTI, S., CACCIAGUERRA, S., AND ROCCETTI, M. 2006. Interactivity-loss avoidance in event delivery synchronization for mirrored game architectures. *IEEE Trans. Multimedia 8*, 4, 874–879.

RANJAN, S., ROLIA, J., FU, H., AND KNIGHTLY, E. 2002. QoS-driven server migration for Internet data centers. In *Proceedings of the Tenth IEEE International Workshop on Quality of Service*, IEEE, Washington, D.C., 3–12.

REISER, M. AND LAVENBERG, S. S. 1980. Mean-value analysis of closed multichain queuing networks. *J. ACM 27*, 2, 313–322.

SPILLNER, J. AND SCHILL, A. 2009. Dynamic SLA template adjustments based on service property monitoring. In *Proceedings of the 2009 IEEE International Conference on Cloud Computing (CLOUD'09)*, IEEE, Washington, D.C., 183–189.

URGAONKAR, B., PACIFICI, G., SHENOY, P., SPREITZER, M., AND TANTAWI, A. 2005. An analytical model for multi-tier internet services and its applications. *SIGMETRICS Perform. Eval. Rev. 33*, 1 (June), 291–302.

URGAONKAR, B., SHENOY, P., CHANDRA, A., GOYAL, P., AND WOOD, T. 2008. Agile dynamic provisioning of multi-tier internet applications. *ACM Trans. Auton. Adapt. Syst. 3*, 1, 1–39.

ZAHORJAN, J., SEVCIK, K. C., EAGER, D. L., AND GALLER, B. 1982. Balanced job bound analysis of queueing networks. *Commun. ACM 25*, 134–141.

ZHANG, Q., CHENG, L., AND BOUTABA, R. 2010. Cloud computing: State-of-the-art and research challenges. *J. Internet Services Appl. 1*, 7–18.

Received May 2011; revised July 2012; accepted July 2012