

# Dynamic Resource Provisioning for Cloud-based Gaming Infrastructures

MORENO MARZOLLA, STEFANO FERRETTI and GABRIELE D'ANGELO, Università di 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, and 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 paper we address the problem of building a large-scale, multi-tier 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 Queueing 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

## 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 high performance requirements, especially response time [Chen et al. 2006; Dick et al. 2005]: depending on the type of game, the response times to ensure a responsive play may range from tens of ms for First Person Shooter action games, to a few seconds for Role-Playing games. MMOGs are usually implemented as client-server architectures, where the server is responsible for maintaining the global state of the virtual play field in response of users (clients) requests. Since the client-server network connection can easily become a significant source of latency, which the game service providers can not control, MMOG services are often hosted on multiple, geographically distributed datacenters, so that each user can be redirected to the “fastest” (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

---

Author's addresses: 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 (M. Marzolla); sferrett@cs.unibo.it (S. Ferretti); g.dangelo@unibo.it (G. D'Angelo).

© ACM, 2011. This is the authors version of the work. It is posted here by permission of ACM for your personal use. Not for redistribution. The definitive version will be published in ACM Computers in Entertainment (CIE).

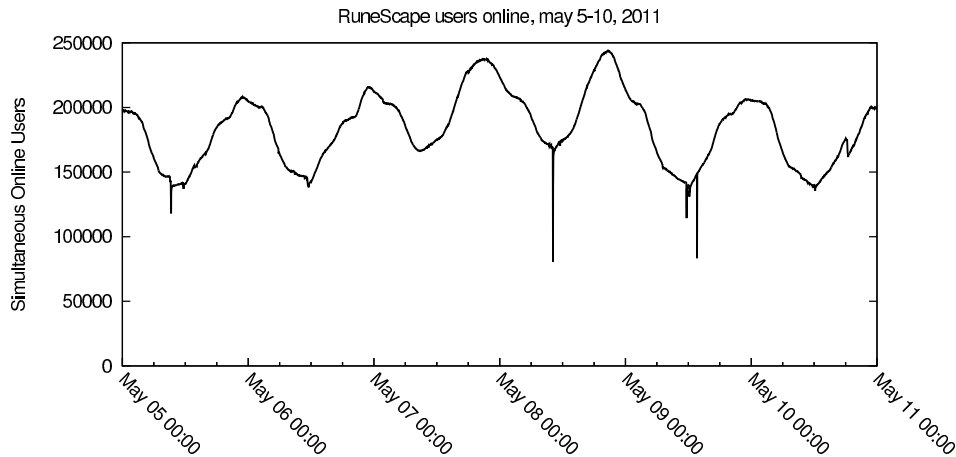


Fig. 1. Number of simultaneous RuneScape players during the period may 5–may 10, 2011

datacenters become congested, users can be migrated to different servers. Multiple datacenters 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, that 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 number of 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 storyling; players can decide to combat against monsters, practice skills or interact competitively or cooperatively with other players. From the technical point of view, RuneScape client code is written in Java and runs inside a browser. Players connect to servers which are located in different world regions<sup>1</sup> to reduce the communication latency.

We collected the total number of online players as displayed in <http://www.runescape.com/> using a sampling period of two minutes during may 2011. Fig. 1 shows a subset of the collected data, from may 5 to may 10, 2011. A daily period 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); this number becomes as low as about 110.000 players 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 Fig. 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 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

<sup>1</sup>The regional servers list is at <http://www.runescape.com/slu.ws?order=WMLPA>

“rent” computing resources, and only pay for resources 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 engineer for peak load limits, and (ii) the MMOG service can be augmented to request more resources during peak periods, and release them when no longer needed.

In this paper we propose a dynamic resource allocation strategy for large-scale MMOG services implemented on top of Cloud infrastructures. We consider a large scale gaming service built over multiple, geographically distributed datacenters, each one providing resources on demand using Cloud infrastructures. Each datacenter 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 pre-defined threshold  $R_{\max}$ ; the specific value of  $R_{\max}$  may depend on the kind of game considered.

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 datacenter depends on the number of nodes allocated there. We thus seek to minimize the total cost of a datacenter 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 virtual region of the virtual world in a specific game session. This is actually what happens in current 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, e.g. 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 which 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 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 paper is structured as follows. In Section 2 we describe the high-level architecture of the MMOG services we consider, and precisely formulate the dynamic provisioning problem as an optimization problem. Section 3 describes our proposed dynamic

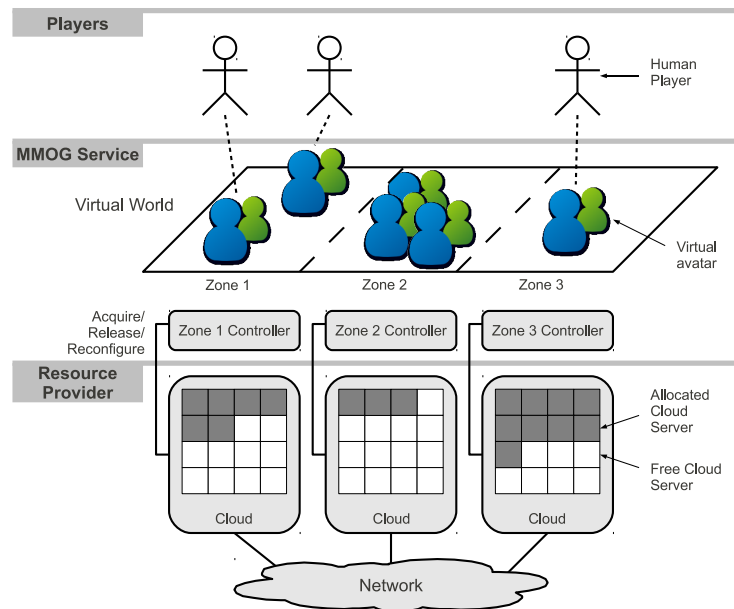


Fig. 2. High level architecture of a distributed, Cloud-based gaming infrastructure

reconfiguration algorithm. The effectiveness of the solution is evaluated in Section 4 by means of numerical experiments. In Section 5 we compare our approach with the relevant literature. Finally, conclusions and future research directions are illustrated in Section 6.

## 2. PROBLEM FORMULATION AND SYSTEM MODEL

We consider a large-scale distributed infrastructure to support MMOGs as shown in Fig. 2. The system has three types of actors: (i) users (game players), which 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*, which 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 implementations partition the metaverse across multiple servers [Kumar et al. 2008]. As the size and complexity of metaverses increase, it is reasonable to consider splitting the metaverse across multiple datacenters, each one handling a partition. We assume that the virtual playfield is split into non-overlapping (or only partially overlapping) zones. 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 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 distribute the workload, each zone is handled by a separate hardware infrastructure hosted on different datacenters. Given that communication between datacenters may incur significant delays, it is important that interaction across neighboring zones is minimized. For example, each partition may hold a collection of “islands” such that all interac-

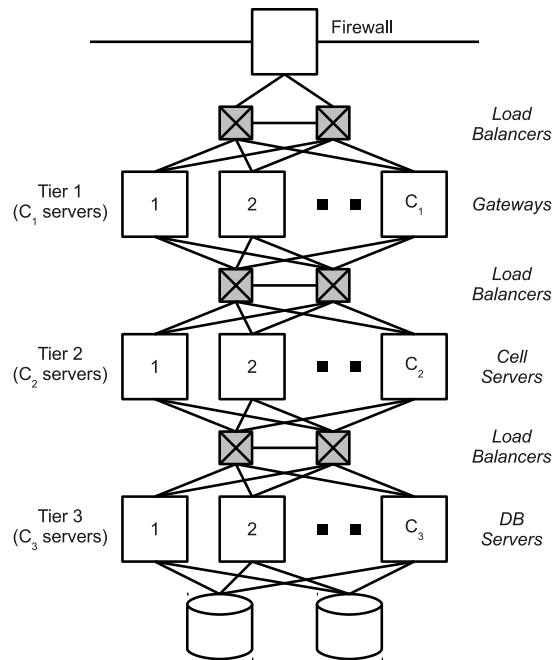


Fig. 3. Multi-tier architecture of the MMOG server hosted by a single datacenter

tions 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 meta-universe may become crowded, while others may become less populated. In order to cope with this variability, each zone controller is physically hosted on resources provided and operated by a Cloud infrastructure. The Cloud provider is in general a separate entity providing computational and storage resources to the game operator on a pay-as-you go model [Zhang et al. 2010]. This means that the game operator can request additional servers and/or additional 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 threshold. 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 Fig. 3. The *firewall* represents the entry point and allows basic 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 datacenter. Finally, the *database servers* are used to store persistent game state information. *Load balancers* are used to evenly distribute requests among servers across layers.

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 paper we address the problem of ensuring that the average response time experienced by a player does not exceed some pre-defined 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*  $\mathbf{C}$  is defined as a vector with three elements  $\mathbf{C} = (C_1, C_2, C_3)$  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:

$$\begin{aligned} & \text{minimize} && (C_1 + C_2 + C_3) && (1) \\ & \text{subject to} && R(\mathbf{C}) < R_{\max} \\ & && \mathbf{C} = (C_1, C_2, C_3) \\ & && C_i \in \{1, 2, \dots\} \quad i = 1, 2, 3 \end{aligned}$$

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

Since the total number of hosts allocated at each Cloud datacenter 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.

### 3. RECONFIGURATION ALGORITHM

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

The monitor is a passive observer which collects run-time performance metrics; in particular, the monitor measures the system response time  $\hat{R}^2$ . When  $\hat{R}$  deviates from  $R_{\max}$ , the monitor triggers the planner, which 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. 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 errors, 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 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

<sup>2</sup>All variables with a hat denote monitored parameters; variables without a hat denote computed values. Variables whose names are in bold are vectors.

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.

### 3.1. The Monitor

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

- The average system response time  $\hat{R}$ ;
- The average system throughput  $\hat{X}$  (rate at which requests, i.e., game events generated by clients connected to that node, are processed);
- The tier utilizations  $\hat{U} = (\hat{U}_1, \hat{U}_2, \hat{U}_3)$ , where  $\hat{U}_i$  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 be actually used. A simple approach is to collect multiple samples of the parameter of interest, 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:  $R_{\text{low}}$  and  $R_{\text{high}}$ , such that  $R_{\text{low}} < R_{\text{high}} < R_{\text{max}}$ . The monitor checks whether the average response time  $\hat{R}$  computed over the last time window is less than  $R_{\text{low}}$  or greater than  $R_{\text{high}}$ . In either case, the planner is invoked. If  $\hat{R} < R_{\text{low}}$  the planner tries to deallocate resources from under-utilized tiers; if  $\hat{R} > R_{\text{high}}$  the planner adds resources to the bottleneck tiers in order to reduce the system response time before it hits the threshold  $R_{\text{max}}$  (details will be given in the next section).

The values for  $R_{\text{low}}$ ,  $R_{\text{high}}$  and  $W$  must be chosen carefully. If  $R_{\text{low}}$  is too small, unnecessary over provision may happen, because unused resources are relinquished only when  $\hat{R} < R_{\text{low}}$ , which may rarely happen. Similarly, if  $R_{\text{high}}$  is too large, violations of the “hard” response time limit  $R_{\text{max}}$  may happen before the system has the opportunity to react. Similarly, 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.

The above control loop is shown in Algorithm 1. The system administrator must choose an initial configuration  $C$  (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 will be described in the next section.

### 3.2. The Planner

The planner is responsible for identifying a new configuration  $C = (C_1, C_2, C_3)$  as a solution of the optimization problem 1. The planner uses a QN performance model to estimate the system response time of different configurations. In this way complex reconfigurations involving addition or removal of multiple hosts at different tiers can be evaluated in a single step; as already observed, this is particularly desirable as the system can adapt quickly.

A datacenter is modeled using the single-class, product-form closed QN shown in Fig. 4. For any configuration  $C = (C_1, C_2, C_3)$ , 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 modeled as a  $M/M/1$

**ALGORITHM 1: QoS-Aware Reconfiguration Algorithm**


---

**Require:**  $R_{low} < R_{high} < R_{max}$ : Thresholds

- 1: Let  $C$  be the initial configuration
- 2: **loop**
- 3:   Monitor the system and compute  $\hat{R}, \hat{X}, \hat{U}$
- 4:    $C' := C$
- 5:   **if** ( $\hat{R} > R_{high}$ ) **then**
- 6:      $C' := \text{Acquire}(C, \hat{U}, \hat{X}, \hat{R})$
- 7:   **else if** ( $\hat{R} < R_{low}$ ) **then**
- 8:      $C' := \text{Release}(C, \hat{U}, \hat{X}, \hat{R})$
- 9:   **end if**
- 10:   **if** a new configuration  $C' \neq C$  has been found **then**
- 11:     Apply the new configuration  $C'$  to the system
- 12:      $C := C'$
- 13:   **end if**
- 14: **end loop**

---

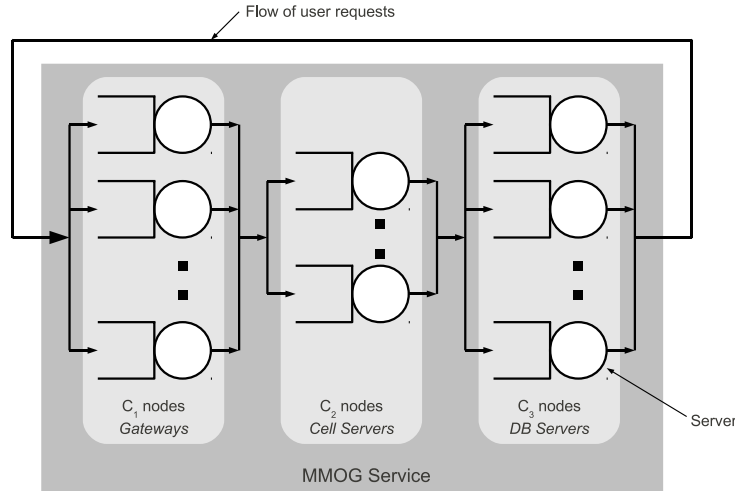


Fig. 4. Queuing Model of a datacenter

queuing center with exponentially distributed inter-arrival times, exponentially distributed service times and FIFO service discipline. We assume that each tier is perfectly balanced, that is, each node in tier  $i$  has exactly the same utilization  $\hat{U}_1$  as all other nodes in the same tier.

These assumptions are made for the sole purpose of making the system easy to analyze. This is important because new configurations must be computed 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. For these reasons we use a simpler model with few parameters which can be easily estimated, as we are about to describe.

Table I. Symbols used in this paper

$\hat{R}$	Measured system response time
$\hat{X}$	Measured system throughput
$\hat{U}_i$	Measured utilization of tier $i$
$R_{\max}$	Maximum allowed response time
$R_{\text{high}}$	Upper response time limit
$R_{\text{low}}$	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 $i$

To analyze the network of Fig. 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 basically 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  $\hat{R}$  and throughput  $\hat{X}$  using Little's Law [Little 1961] as:

$$N = \hat{X}\hat{R} \quad (2)$$

The service demand  $D_i$  of tier  $i$  can be seen as 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  $\hat{U}_i/\hat{X}$ . 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

$$D_i = C_i \times \frac{\hat{U}_i}{\hat{X}}, \quad i = 1, 2, 3 \quad (3)$$

Table I summarizes all symbols used in this paper.

Under the assumptions above, the QN has product-form solution [Balsamo 2000], which means that steady-state performance measures (such as response times, average number of requests in each center, and so on) can be computed efficiently.

The Mean Value Analysis (MVA) algorithm [Lazowska et al. 1984; 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  $1, 2, \dots, N$ . A closed network with  $N$  requests and  $K$  queueing centers can be analyzed using the general MVA algorithm in time  $O(NK)$ . However, since all servers in each tier are equivalent, the network of Fig. 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 Fig. 4 can be analyzed in time  $O(1)$  (independent from 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 computation-

**ALGORITHM 2:**  $MVA(C, D, N) \rightarrow \langle U, R \rangle$ 


---

**Require:** C configuration to analyze  
**Require:**  $D = (D_1, D_2, D_3)$  total service demands of tiers  
**Require:**  $N$  number of active users  
**Ensure:**  $U = (U_1, U_2, U_3)$  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  $k := 1, \dots, 3$  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  $\langle U, R \rangle$ 

```

---

**ALGORITHM 3:**  $BSB(C, D, N) \rightarrow \langle U, R \rangle$ 


---

**Require:** C configuration to analyze  
**Require:**  $D = (D_1, D_2, D_3)$  total service demands of tiers  
**Require:**  $N$  number of active users  
**Ensure:**  $U = (U_1, U_2, U_3)$  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  $\langle U, R \rangle$ 

```

---

ally efficient computation of BSB. Therefore, we will 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  $D = (D_1, D_2, D_3)$  represents the *total* service demand vector of tiers 1–3 as computed using Eq. (3), using the measurements performed by the monitor.

**ALGORITHM 4:**  $\text{Acquire}(C, \hat{U}, \hat{X}, \hat{R}) \rightarrow C'$ 


---

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

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

---

**Acquiring new resources.** When the measured response time  $\hat{R}$  is greater than  $R_{\text{high}}$ , 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  $\text{Acquire}()$  shown in Algorithm 4. The procedure uses a simple greedy strategy to iteratively define  $C'$  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  $C'$  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  $\hat{R}$  is less than  $R_{\text{low}}$ , we try to release hosts. Let  $C$  be the current configuration at the time the planner is invoked. The new configuration  $C'$  is computed by procedure  $\text{Release}()$  shown in Algorithm 5. Again, procedure  $\text{Release}()$  uses an iterative greedy strategy to compute  $C'$  from  $C$ . At each iteration we consider the set  $S \subseteq \{1, 2, 3\}$  of tiers with more than one node (line 4);  $S$  represents the set of tiers from which one host could be removed. We identify the tier  $u \in S$  with 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.

### 3.3. 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

**ALGORITHM 5:** Release( $C, \hat{U}, \hat{X}, \hat{R}$ )  $\rightarrow C'$ 


---

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

- 1:  $C' := C$
- 2: Compute  $N$  using Eq. (2)
- 3: Compute  $D = (D_1, D_2, D_3)$  using Eq. (3)
- 4:  $S := \{i \mid C'_i > 1\}$
- 5:  $\langle U, R \rangle := \text{BSB}(C', D, N)$  {Evaluate configuration  $C'$ }
- 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:    $\langle U, R \rangle := \text{BSB}(C', D, N)$  {Evaluate configuration  $C'$ }
- 10:   **if** ( $R \geq (R_{\text{high}} + R_{\text{low}})/2$ ) **then**
- 11:      $C'_u := C'_u + 1$  {Rollback to old configuration}
- 12:      $S := S \setminus \{u\}$  {Remove  $u$  from the set  $S$ }
- 13:   **else**
- 14:      $S := \{i \mid C'_i > 1\}$
- 15:   **end if**
- 16: **end while**
- 17: **Return**  $C'$

---

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.

Given the current system configuration  $C = (C_1, C_2, C_3)$ , the new configuration  $C' = (C'_1, C'_2, C'_3)$  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  $O\left(\sum_{i=1}^3 (C'_i - C_i)\right)$ .

Similarly, given the current configuration  $C$ , the new configuration  $C'$  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  $O\left(\sum_{i=1}^3 C_i\right)$ .

#### 4. NUMERICAL RESULTS

In this section we evaluate the dynamic reconfiguration strategy described in Section 3. Algorithms 4 and 5 have been implemented in GNU Octave [Eaton 2002], an interpreted language which is particularly suitable for implementing numerical algorithms involving vector computations. We performed two sets of experiments: 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  $t = 1, \dots, T$ . 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  $\hat{R}$ , throughput  $\hat{X}$  and tier utilizations  $\hat{U}_i$  are computed using the MVA algorithm on the QN model of Fig. 4.

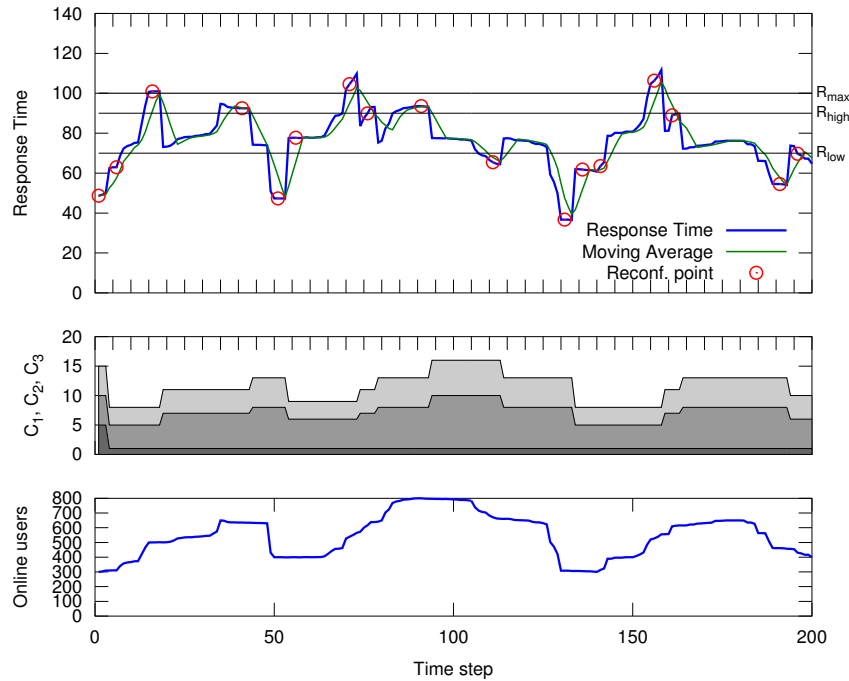


Fig. 5. Simulation result, irregular workload with low churn.

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 5–8 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 paper, while the thin line shows the time-averaged system response time over the last  $W$  steps. Horizontal lines show the values of  $R_{\max}$ ,  $R_{\text{high}}$  and  $R_{\text{low}}$  respectively. The times at which a new configuration is applied are also shown as crosses. 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 5 days from may 5 to may 10 2011; the data is shown in Fig. 1. Here we consider a subset of the collected data which consists of tree days, from may 5 to may 7 (about 2160 data points, one every two minutes). We re-sampled 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 (approx 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 re-

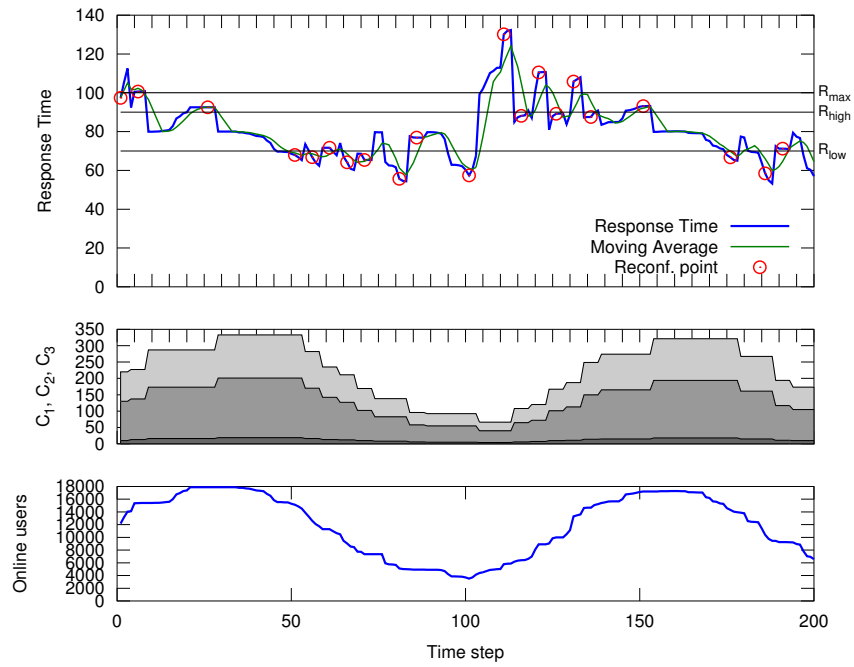


Fig. 6. Simulation result, periodic workload with low frequency.

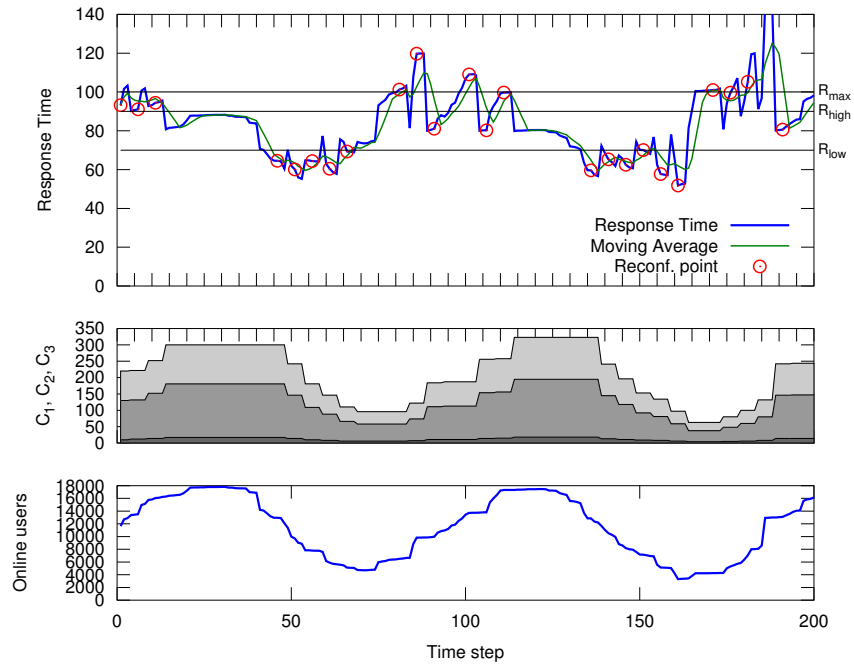


Fig. 7. Simulation result, periodic workload with medium frequency.

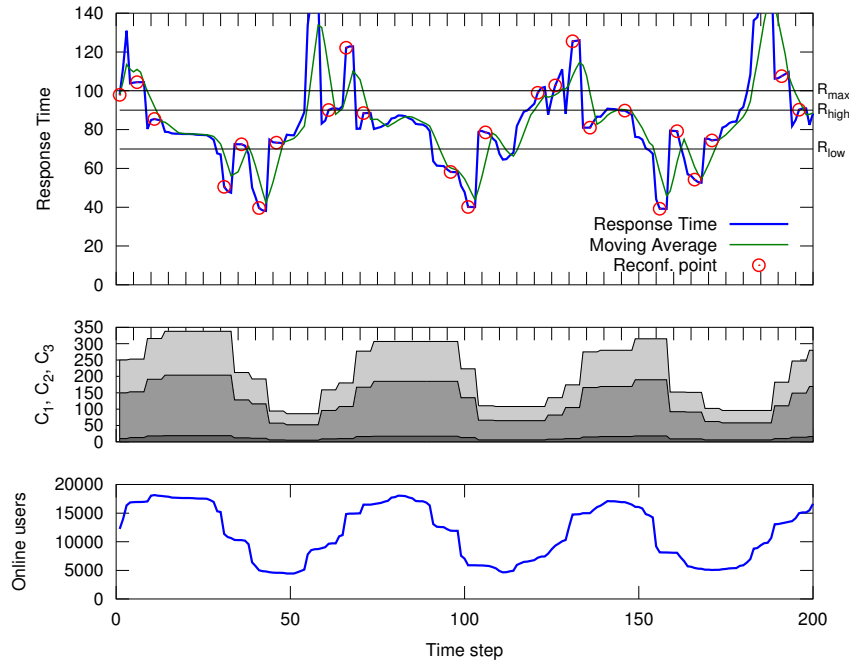


Fig. 8. Simulation result, periodic workload with high frequency (implying high churn).

Table II. Simulation results

	Online users		Reconf.	SLA violations	N. Servers (dynamic)			N. Servers (static)	N. Serv. Dynamic / N. Serv. Static
	<i>min</i>	<i>max</i>			<i>min</i>	<i>max</i>	<i>tot</i>		
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

sult is shown in Fig. 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 which 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  $\min\{C_1(t) + C_2(t) + C_3(t)\}$ , the maximum number of servers is  $\max\{C_1(t) + C_2(t) + C_3(t)\}$ , and the total number of servers is  $\sum_t (C_1(t) + C_2(t) + C_3(t))$ .

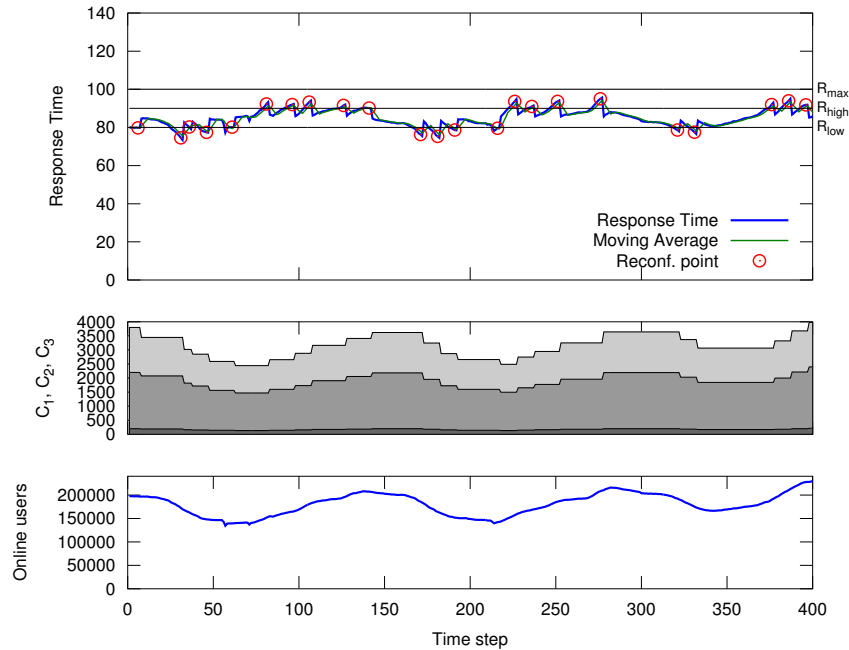


Fig. 9. Simulation result with the real workload (a time step corresponds to 10s of wall clock time)

- 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.

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 on Figures 5–9 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 Figs. 6–8, which have an increasingly high workload fluctuation frequency causing higher churn in a very short time. If the workload fluctuates smoothly, as in Fig. 5, the response time is almost always kept below the threshold  $R_{\max}$ . If larger fluctuations happen, as in Fig. 6–8, our adaptation algorithm may require some time to react properly.

It is interesting to observe that real workloads for MMOG does *not* exhibit steep fluctuations, as can be seen on Fig. 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 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.

## 5. RELATED WORKS

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 Queueing 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 multi-tier 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 which 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 in combination 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 be trivially applied also to any other kind of large-scale service which can be reasonably modeled with a suitable QN. For example, our approach can be applied to system architectures more complex than the three-tier model considered in Fig. 3.

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 re-configuration 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 surpasses a predetermined threshold.

Other works focus mostly on issues related to the definition and monitoring of the SLAs in a Cloud computing environment and do 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 non-functional 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 noticing that due to the modularity of our architecture, one could investigate the possibility of integrating a SLMA in our solution).

## 6. CONCLUSIONS AND FUTURE WORKS

In this paper 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 geographically distributed datacenter, each datacenter 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 datacenter is passively monitored to detect when the average response time deviates from the threshold  $R_{\max}$ . When that happens, we reconfigure the datacenter 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 paper can be improved along several directions. In this paper we assumed that the cost of all Cloud resources is the same; this may not be the case, e.g., if a DB server machine needs a different configuration (and thus, has 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 mean to trigger reconfigurations in a proactive way. Another extension of the proposed approach, that is currently under investigation, is the instrumentation of software clients used by the players. In this way, it would be possible to collect runtime statistics about the gaming experience of each player and to consider the re-allocation of gamers among the Tier 1 hosts (i.e. Gateways). This could bring to a reduction of the latency experienced by each client, including in the adaptive evaluation process the whole gaming infrastructure and therefore, in some extent, improving the gaming experience. Finally, we are working on the implementation of our methodology on a real testbed, to assess its effectiveness through a more comprehensive set of real experiments.

## REFERENCES

- BALSAMO, S. 2000. Product form queueing networks. In *Performance Evaluation: Origins and Directions*, G. Haring, C. Lindemann, and M. Reiser, Eds. Lecture Notes in Computer Science Series, vol. 1769. Springer, 377–401.
- 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 sixteenth workshop on Parallel and distributed simulation. PADS '02*. IEEE Computer Society, Washington, DC, USA, 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, 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 *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*. 321–328.
- GUSTAFSSON, F. 2000. *Adaptive Filtering and Change Detection*. John Wiley & Sons, Ltd.
- HSIAO, T.-Y. AND YUAN, S.-M. 2005. Practical middleware for massively multiplayer online games. *IEEE Internet Computing* 9, 47–54.

- JAGEX LTD. 2011. RuneScape. <http://www.runescape.com/>, may 2011.
- 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.
- LI, J., CHINNECK, J., WOODSIDE, M., LITOIU, M., AND ISZLAI, G. 2009. Performance model driven qos guarantees and optimization in clouds. In *Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*. CLOUD '09. IEEE Computer Society, Washington, DC, USA, 15–22.
- LITTLE, J. D. C. 1961. A proof for the queueing formula:  $L = \lambda W$ . *Operations Research* 9, 3, 383–387.
- MARZOLLA, M. AND MIRANDOLA, R. 2010. Performance aware reconfiguration of software systems. In *Computer Performance Engineering—7th European Performance Engineering Workshop, EPEW 2010, Bertinoro, Italy, September 23-24, 2010. Proceedings*, A. Aldini, M. Bernardo, L. Bononi, and V. Cortellessa, Eds. Lecture Notes in Computer Science Series, vol. 6342. Springer, 51–66.
- MAUVE, M., VOGEL, J., HILT, V., AND EFFELBERG, W. 2004. Local-lag and timewarp: providing consistency for replicated continuous applications. *IEEE Transactions on Multimedia* 6, 1, 47–57.
- NAE, V., IOSUP, A., AND PRODAN, R. 2010. Dynamic resource provisioning in massively multiplayer online games. *IEEE Transactions on Parallel and Distributed Systems* 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 Transactions on Multimedia* 8, 4, 874–879.
- RANJAN, S., ROLIA, J., FU, H., AND KNIGHTLY, E. 2002. Qos-driven server migration for internet data centers. In *Quality of Service, 2002. Tenth IEEE International Workshop on*. 3–12.
- REISER, M. AND LAVENBERG, S. S. 1980. Mean-value analysis of closed multichain queueing networks. *Journal of the 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 Computer Society, Washington, DC, USA, 183–189.
- 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. *Journal of Internet Services and Applications* 1, 7–18.