Dynamic Scalability for Next Generation Gaming Infrastructures

Moreno Marzolla Stefano Ferretti Gabriele D'Angelo

Technical Report UBLCS-2010-12

December 2010

Department of Computer Science
University of Bologna
Mura Anteo Zamboni 7
40127 Bologna (Italy)

Recent Titles from the UBLCS Technical Report Series

2009-07 *Self-Organizing Mechanisms for Task Allocation in a Knowledge-Based Economy (Ph.D. Thesis)*, Marcozzi, A., February 2009.

2009-08 *3-Dimensional Protein Reconstruction from Contact Maps: Complexity and Experimental Results (Ph.D. Thesis)*, Medri, F., February 2009.

2009-09 *A core calculus for the analysis and implementation of biologically inspired languages (Ph.D. Thesis)*, Versari, C., February 2009.

2009-10 *Probabilistic Data Integration*, Magnani, M., Montesi, D., March 2009.

2009-11 *Equilibrium Selection via Strategy Restriction in Multi-Stage Congestion Games for Real-time Streaming*, Rossi, G., Ferretti, S., D'Angelo, G., April 2009.

2009-12 *Natural deduction environment for Matita*, C. Sacerdoti Coen, E. Tassi, June 2009.

2009-13 *Hints in Unification*, Asperti, A., Ricciotti, W., Sacerdoti Coen, C., Tassi, E., June 2009.

2009-14 *A New Type for Tactics*, Asperti, A., Ricciotti, W., Sacerdoti Coen, C., Tassi, E., June 2009.

2009-15 *The k-Lattice: Decidability Boundaries for Qualitative Analysis in Biological Languages*, Delzanno, G., Di Giusto, C., Gabbrielli, M., Laneve, C., Zavattaro, G., June 2009.

2009-16 *Landau's "Grundlagen der Analysis" from Automath to lambda-delta*, Guidi, F., September 2009.

2010-01 *Fast overlapping of protein contact maps by alignment of eigenvectors*, Di Lena, P., Fariselli, P., Margara, L., Vassura, M., Casadio, R., January 2010.

2010-02 *Optimized Training of Support Vector Machines on the Cell Processor*, Marzolla, M., February 2010.

2010-03 *Modeling Self-Organizing, Faulty Peer-to-Peer Systems as Complex Networks* Ferretti, S., February 2010.

2010-04 *The qnetworks Toolbox: A Software Package for Queueing Networks Analysis*, Marzolla, M., February 2010.

2010-05 *QoS Analysis for Web Service Applications: a Survey of Performance-oriented Approaches from an Architectural Viewpoint*, Marzolla, M., Mirandola, R., February 2010.

2010-06 *The dark side of the board: advances in Kriegspiel Chess (Ph.D. Thesis)*, Favini, G.P., March 2010.

2010-07 *Higher-Order Concurrency: Expressiveness and Decidability Results (Ph.D. Thesis)*, Perez Parra, J.A., March 2010.

2010-08 *Machine learning methods for prediction of disulphide bonding states of cysteine residues in proteins (Ph.D. Thesis)*, Shukla, P., March 2010.

2010-09 *Pseudo-Boolean clustering*, Rossi, G., May 2010.

2010-10 *Expressiveness in biologically inspired languages (Ph.D. Thesis)*, Vitale, A., March 2010.

2010-11 *Performance-Aware Reconfiguration of Software Systems*, Marzolla, M., Mirandola, R., May 2010.

Dynamic Scalability for Next Generation Gaming Infrastructures

Moreno Marzolla[1] **Stefano Ferretti**[2] **Gabriele D'Angelo**[3]

Technical Report UBLCS-2010-12

December 2010

Abstract

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 variabilities in the workload, and nevertheless must always operate under very strict 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 the service providers to allocate as many resources as they need using a pay as you go model. We harness this paradigm by describing a dynamic provisioning algorithm which can resize the resource pool to adapt to workload variabilities, still maintaining a response time below a user-defined threshold. Our algorithm uses a Queueing Network performance model to quickly evaluate different configurations. Numerical experiments are used to validate the effectiveness of the proposed approach.

[1] Dipartimento di Scienze dell'Informazione, Università di Bologna, Mura Anteo Zamboni 7, I-40127 Bologna, Italy, Email: marzolla@cs.unibo.it

[2] Dipartimento di Scienze dell'Informazione, Università di Bologna, Mura Anteo Zamboni 7, I-40127 Bologna, Italy, Email: sferrett@cs.unibo.it

[3] Dipartimento di Scienze dell'Informazione, Università di Bologna, Mura Anteo Zamboni 7, I-40127 Bologna, Italy, Email: g.dangelo@unibo.it

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 [2, 3]: 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, 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. Multiple datacenters also help to address scalability problems, because the virtual world can be partitioned or replicated across the available servers [1, 14].

Nevertheless, scalability problems still exist because the system is subject to high variability of the workload. To cope with this, 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, being based on a worst-case scenario, a number of allocated resources may remain unused at run time.

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 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 the top of Cloud infrastructures. We consider a large scale gaming service implemented over multiple, geographically distributed datacenters, each one providing resources on demand using Cloud computing infrastructures. Each datacenter hosts a three-tier system, which handles one partition of the virtual world; thanks to the underlying Cloud, it is possible to add (remove) computing nodes to (from) any tier. We assume that the MMOG service is capable of transparently reconfigure itself when the configuration 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. 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 is not valid when multiple servers working on the same virtual region of the same game session are geographically distributed. In this case, in fact, other factors might influence the level of provided responsiveness, e.g. the synchronization algorithm, the distribution of clients connected to distributed servers [12, 14].

We assume that the system response time R must be kept below a pre-defined threshold $R_{\max}$ (which in general depends on the kind of game considered). 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 $R < R_{\max}$.

To do so, we enhance the gaming service hosted in each Cloud with 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

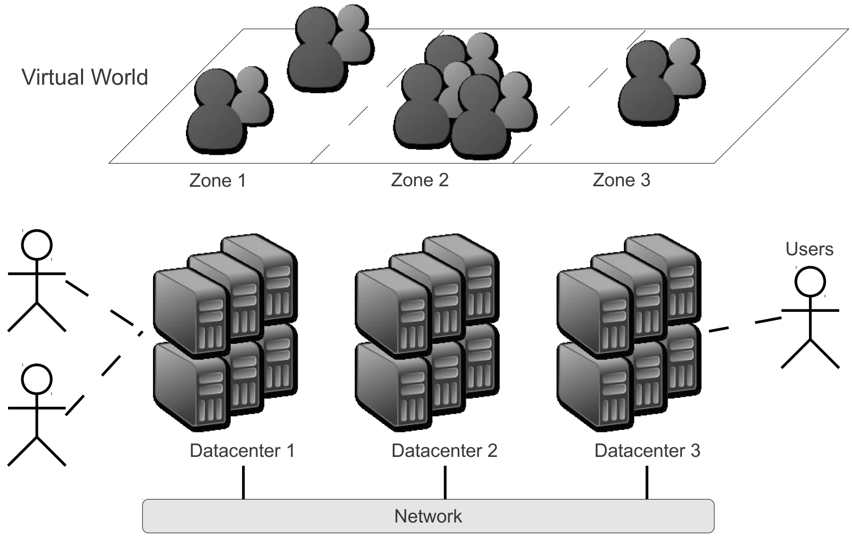

Figure 1. High level architecture of a distributed, Cloud-based gaming infrastructure

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 at a time is added (or removed) from a suitably chosen tier. The planner uses a Queueing Network (QN) performance model 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 performance model, the planner is capable of computing 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 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 reconfiguration algorithm. The effectiveness of the proposed 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

We consider a large-scale distributed infrastructure to support MMOGs; we use the term *MMOG service* to denote the whole infrastructure. The MMOG service is responsible for maintaining state information about a single virtual universe, or *metaverse*. In order to ensure scalability of the MMOG service, many existing implementations partition the metaverse across multiple servers [8]. As the size and complexity of metaverses increase, it is reasonable to consider splitting the metaverse across multiple datacenters, each one handling a partition (see Fig. 1).

Given that communication between datacenters may incur significant delays, it is necessary to partition the metaverse in such a way that interaction across neighboring partitions 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 gaming infrastructure makes crossing a partition a seamless operation.

Thus, depending on the (virtual) mobility pattern of each player, some areas of the metaverse

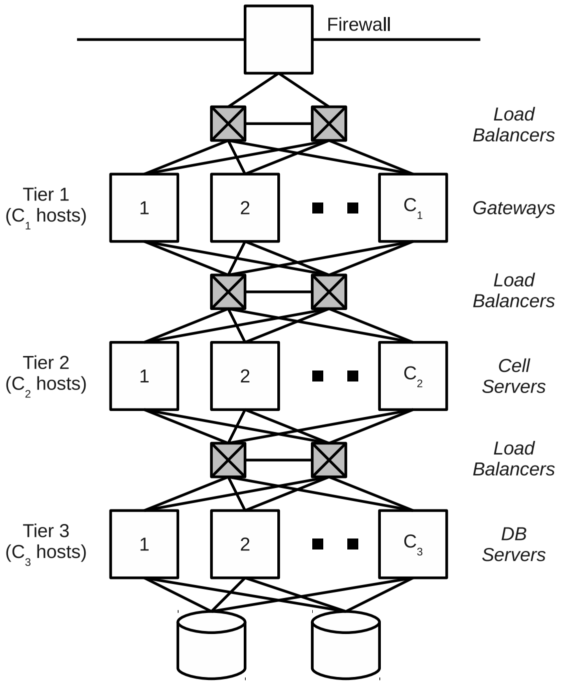

Figure 2. Multi-tier architecture of the MMOG server hosted by a single datacenter

may become crowded, while others may become less populated. Crowds may migrate from one partition of the virtual world to a different one, so each datacenter can be subject to fluctuating workloads. In order to cope with this variability, as assume that each datacenter provides a Cloud infrastructure, so that it is possible to dynamically allocate and release resources.

Internally, each datacenter hosts a multi-tier architecture as described in [6], shown in Fig. 2. The *firewall* represents the entry point to each datacenter, 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 *databases* 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 datacenter to another as they move across the metaverse. In this paper we propose a model-based algorithm for dynamic reconfiguration of the MMOG service; specifically, the algorithm tries to maintain the response time at each datacenter below a predefined maximum value $R_{\max}$. To do so, our algorithm uses a simple QN model to estimate the kind and number of servers to add/remove at each datacenter.

In this section we describe the reconfiguration algorithm which is used by each datacenter to dynamically add/remove hosts in order to adapt to workload fluctuations.

At any time, 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 the tier i. 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 system configuration with minimum total number of servers which are necessary to achieve an (estimated) system response time below a pre-defined threshold $R_{\max}$. Formally, we aim at solving the following optimization problem:

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

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

Since the total number of hosts which are allocated at each datacenter are 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. The idea is to monitor the datacenter by collecting run-time parameters such as the average utilization of each server and the overall system response time $\hat{R}$[4]. When $\hat{R}$ deviates from $R_{\max}$ we compute a new system configuration by finding an approximate solution to the optimization problem (1). Specifically, if the system is overloaded we add new servers to the bottleneck tier(s). Symmetrically, when the measured system response time becomes low, we try to remove server instances from the tier with lowest utilization.

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.

We describe a heuristic for solving the optimization problem (1) with the help of a QN performance model of the MMOG service hosted in each datacenter. The QN model is used to quickly estimate the system response time for different configurations. In this way, the impact of different configuration changes can be examined quickly without the need of actually reconfiguring the system.

To implement the idea above, we enhance the MMOG system shown in Fig. 2 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 above independently from the others. The *monitor* collects run-time statistics on the running system. In particular, the monitor measures the system response time $\hat{R}$ and other parameters which are used to analyze the QN model (details will be given below). When the system response time $\hat{R}$ deviates from $R_{\max}$, the monitor invokes the planner which finds a new system configuration; the system then can be actually reconfigured by allocating new server instances (or removing unused servers) from the Cloud. The *planner* uses the run-time informations collected by the monitor to find an approximate solution to the optimization problem (1) using a single-class, closed QN model to quickly explore different arbitrary system configurations.

It is worth noting that the *monitor* and *planner* architecture is quite simple and does not require any major modification to be applied in existing gaming infrastructures. Furthermore, it can be implemented itself relying on Cloud based services and therefore in a very elastic way.

3.1 The Monitor

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

[4] All variables with a hat denote monitored parameters; variables without a hat denote computed values. Variables whose names are in bold are vectors.

Algorithm 1 QoS-Aware Reconfiguration Algorithm

```
Require: $R_{\text{low}} < R_{\text{high}} < R_{\max}$: Thresholds
 1: Let $\mathbf{C}$ be the initial configuration
 2: loop
 3:   Monitor the system and determine $\hat{R}, \hat{X}, \hat{\mathbf{U}}$
 4:   $\mathbf{C}' \leftarrow \mathbf{C}$
 5:   if $(\hat{R} > R_{\text{high}})$ then
 6:     $\mathbf{C}' \leftarrow \mathsf{Acquire}(\mathbf{C}, \hat{\mathbf{U}}, \hat{X}, \hat{R})$
 7:   else if $(\hat{R} < R_{\text{low}})$ then
 8:     $\mathbf{C}' \leftarrow \mathsf{Release}(\mathbf{C}, \hat{\mathbf{U}}, \hat{X}, \hat{R})$
 9:   if a new configuration $\mathbf{C}' \neq \mathbf{C}$ has been found then
10:     Apply the new configuration $\mathbf{C}'$ to the system
11:     $\mathbf{C} \leftarrow \mathbf{C}'$
```

- The average system response time $\hat{R}$;
- The average system throughput (rate at which requests, i.e., game events generated by clients connected to that node, are processed) $\hat{X}$;
- The utilizations $\hat{\mathbf{U}} = (\hat{U}_1, \hat{U}_2, \hat{U}_3)$, where $\hat{U}_i$ is the utilization of tier i.

Given that most of the observed 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, which will be used in this paper, is to collect multiple samples of the parameter of interest, and compute a moving average over a time window of length W.

The system administrators must define two additional thresholds: R_{low} and R_{high}, such that $R_{\text{low}} < R_{\text{high}} < R_{\max}$. The monitor checks whether the average response time $\hat{R}$ computed over the last time window is less than R_{low} or greater than R_{high}. In either case, the planner is invoked. If $\hat{R} < R_{\text{low}}$ 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 $R_{\max}$ (details will be given in the next section).

The values for R_{low}, R_{high} and W must be chosen carefully. If R_{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_{high} is too large, violations of the "hard" response time constraint $\hat{R} < R_{\max}$ 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 $\mathbf{C}$ (line 1), after which an infinite loop is used to collect monitoring data and add/remove resources if needed.

The functions Acquire and Release will be described in the next section.

3.2 The Planner

The planner is responsible for identifying a new configuration $\mathbf{C} = (C_1, C_2, C_3)$ 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 reconfiguration 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. 3. For any configuration $\mathbf{C} = (C_1, C_2, C_3)$, 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, as shown. Each host is represented by a $M/M/1$

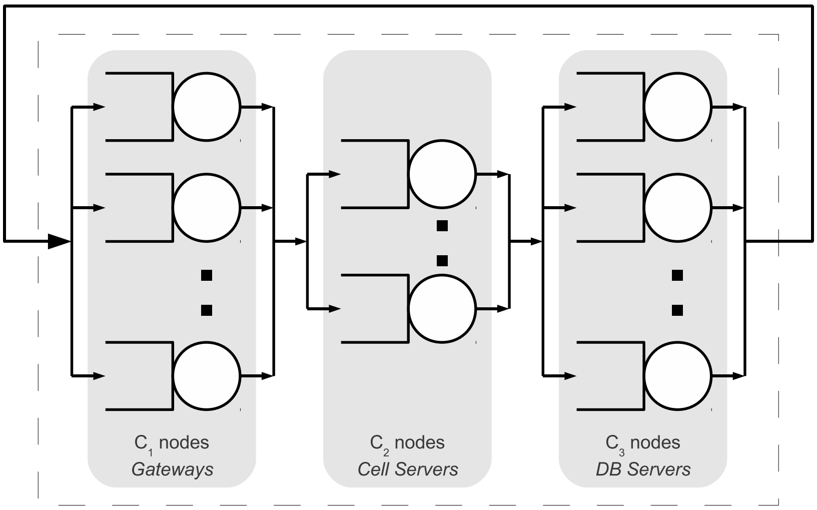

Figure 3. Queueing Model of a datacenter

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 needed to make 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. In particular, the QN shown in Fig. 3 can be analyzed using the Mean Value Analysis (MVA) algorithm. To apply the MVA algorithm, the following additional parameters are needed to compute the performance results: (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 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 [11] as:

$$N = \hat{X}\hat{R} \tag{2}$$

The service demand D_i of tier i is defined as the total time spent by requests in any node of that tier. According to the Utilization Law [9], it is possible to express the service demands D_i as:

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

Table 1 summarizes all symbols used in this paper.

Given the above parameters, the system response time $R(\mathbf{C})$ for any arbitrary system configuration $\mathbf{C}$ can be estimated using Algorithm 2, which calculate Balanced System Bounds (BSB) [18] for a closed network with N requests. Bound analysis only provides 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). Thus, we estimate R as the average of its upper and lower bounds $(R^+ + R^-)/2$ (line 9). The parameter $\mathbf{D} = (D_1, D_2, D_3)$ in Algorithm 2 represents the *total* service demands at tiers $1, 2, 3$ as computed with Eq. (3), according to measurements performed by the monitor. The service demand of a single node on tier i, assuming that there are C_i nodes on tier i, is D_i/C_i.

Acquiring new resources When the measured response time $\hat{R}$ is greater than R_{high}, we need to allocate new resources to improve the system responsiveness. Let $\mathbf{C}$ be the current configuration at the time the planner is invoked. The new configuration $\mathbf{C}'$ is computed by the procedure

$\hat{R}$	Measured system response time
$\hat{X}$	Measured system throughput
$\hat{U}_i$	Measured utilization of tier i
$R_{\max}$	Maximum allowed response time
R_{high}	Upper limit for reconfiguration
R_{low}	'Lower limit for reconfiguration
N	Number of concurrent requests
C_i	Number of nodes at tier i
D_i	Total service demand of tier i
U_i	Utilization of nodes at tier i

Table 1. Symbols used in this paper

Algorithm 2 BSB$(\mathbf{C}, \mathbf{D}, N) \rightarrow \langle \mathbf{U}, R \rangle$

```
Require: $\mathbf{C}$ configuration to analyze
Require: $\mathbf{D} = (D_1, D_2, D_3)$ service demands
Require: $N$ number of active users
Ensure: $\mathbf{U} = (U_1, U_2, U_3)$ is the approximate utilization of tiers
Ensure: $R$ is the approximate system response time
$D_{\max} \leftarrow \max\{D_1, D_2, D_3\}$
$D_{\text{tot}} \leftarrow (D_1 + D_2 + D_3)$
$D_{\text{ave}} \leftarrow D_{\text{tot}}/3$
$X^- \leftarrow N/(D_{\text{tot}} + (N-1) \times D_{\max})$
$X^+ \leftarrow \min\{1/D_{\max}, N/(D_{\text{tot}} + (N-1) \times D_{\text{ave}})\}$
$R^- \leftarrow \max\{N \times D_{\max}, D_{\text{tot}} + (N-1) \times D_{\text{ave}}\}$
$R^+ \leftarrow D_{\text{tot}} + (N-1) \times D_{\max}$
$X \leftarrow (X^+ + X^-)/2$                                   {System Throughput}
$R \leftarrow (R^+ + R^-)/2$                                   {Response Time}
for $i \leftarrow 1$ to $3$ do
    $U_i \leftarrow X \times D_i$
Return $\langle \mathbf{U}, R \rangle$
```

Acquire() shown in Algorithm 3. The procedure uses a simple greedy strategy to iteratively define $\mathbf{C}'$ starting from $\mathbf{C}$. 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 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 $\hat{R}$ is less than R_{low}, we try to release hosts. Let $\mathbf{C}$ be the current configuration at the time the planner is invoked. The new configuration $\mathbf{C}'$ is computed by the procedure Release() shown in Algorithm 4. Procedure Release() uses again an iterative greedy strategy to compute $\mathbf{C}'$ from $\mathbf{C}$. At each iteration we consider the set S 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 then 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 the set S (line 11), and iterate again. The process stops when S becomes empty, which means that either (i) all tier 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 $(R_{\text{high}} + R_{\text{low}})/2$ (that is, the candidate set S becomes empty).

Algorithm 3 Acquire$(\mathbf{C}, \hat{\mathbf{U}}, \hat{X}, \hat{R}) \to \mathbf{C}'$

```
Require: $\mathbf{C}$ Current system configuration
Require: $\hat{\mathbf{U}}$ Measured utilizations
Require: $\hat{X}$, $\hat{R}$ Measured system throughput and response time
Ensure: $\mathbf{C}'$ New system configuration
$\mathbf{C}' \leftarrow \mathbf{C}$
Compute $N$ using Eq. (2)
Compute $\mathbf{D} = (D_1, D_2, D_3)$ using Eq. (3)
repeat
    $\langle \mathbf{U}, R \rangle \leftarrow$ BSB$(\mathbf{C}', \mathbf{D}, \mathbf{N})$        {Evaluate configuration $\mathbf{C}'$}
    if $(R \geq (R_{\text{high}} + R_{\text{low}})/2)$ then
        Let $b$ the tier with highest utilization $U_b$
        $C'_b \leftarrow C'_b + 1$
until $R < (R_{\text{high}} + R_{\text{low}})/2$
Return $\mathbf{C}'$
```

Algorithm 4 Release$(\mathbf{C}, \hat{\mathbf{U}}, \hat{X}, \hat{R}) \to \mathbf{C}'$

```
Require: $\mathbf{C}$ Current system configuration
Require: $\hat{\mathbf{U}}$ Measured utilizations
Require: $\hat{X}$, $\hat{R}$ Measured system throughput and response time
Ensure: $\mathbf{C}'$ New system configuration
$\mathbf{C}' \leftarrow \mathbf{C}$
Compute $N$ using Eq. (2)
Compute $\mathbf{D} = (D_1, D_2, D_3)$ using Eq. (3)
$S \leftarrow \{i \mid C'_i > 1\}$
$\langle \mathbf{U}, R \rangle \leftarrow$ BSB$(\mathbf{C}', \mathbf{D}, \mathbf{N})$        {Evaluate configuration $\mathbf{C}'$}
while $(S \neq \emptyset)$ do
    Let $u$ the tier in $S$ with lowest utilization
    $C'_u \leftarrow C'_u - 1$        {Try to reduce tier $u$}
    $\langle \mathbf{U}, R \rangle \leftarrow$ BSB$(\mathbf{C}', \mathbf{D}, \mathbf{N})$        {Evaluate configuration $\mathbf{C}'$}
    if $(R \geq (R_{\text{high}} + R_{\text{low}})/2)$ then
        $C'_u \leftarrow C'_u + 1$        {Rollback to old configuration}
        $S \leftarrow S \setminus \{u\}$        {Remove $u$ from the set $S$}
    else
        $S \leftarrow \{i \mid C'_i > 1\}$
Return $\mathbf{C}'$
```

3.3 Computational Cost

Both Algorithms 3 and 4 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.

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

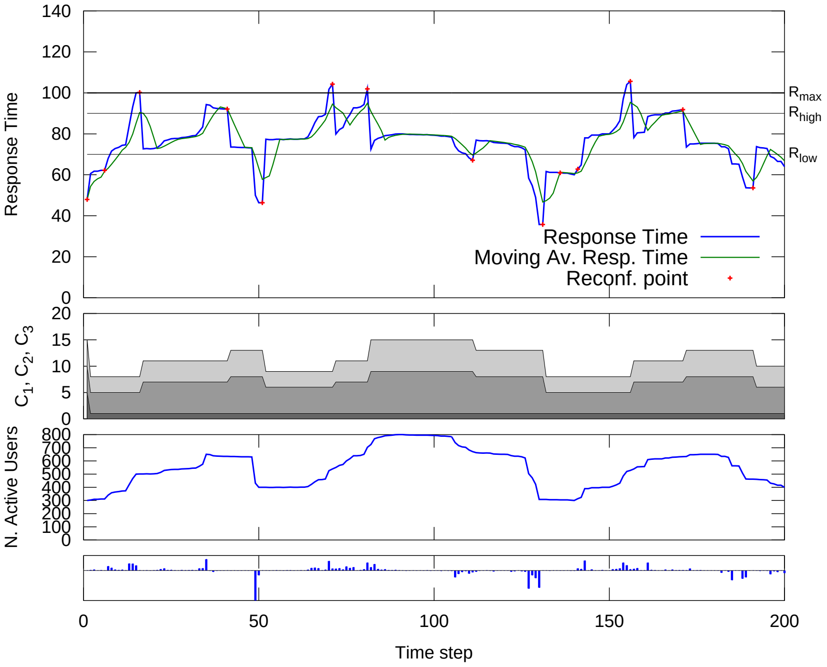

Figure 4. Simulation results: Test 1

4 Numerical Results

In this section we evaluate the dynamic reconfiguration strategy described in Section 3. Algorithms 3 and 4 have been implemented in GNU Octave [4], which allows to concisely describe numerical algorithms involving vector computations.

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 $S_1 = 0.08, S_2 = 0.8, S_3 = 0.58$; in this way, the QN model in Fig. 3 has been used to compute, at each step, the system response time, throughput and individual server utilizations using the MVA algorithm.

In all our experiment we set $R_{\text{max}} = 100$, $R_{\text{high}} = 90$ and $R_{\text{low}} = 70$. The planner uses a window size of $W = 5$ time steps. We evaluate our approach in three different scenarios, using different workload intensities.

Figures 4–7 show the results obtained under different workloads. Each plot contains four parts, as follows. The top part shows 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_{high} and R_{low} respectively. The times at which a new configuration is applied are also shown. Below this, we show the number of allocated nodes: the height of each colored band denotes the number of hosts on each tier (C_1, C_2 and C_3, from bottom to top). Finally, in the bottom we show the number of active users N_t for each time step t; below that we also show the number of users who joined/left during step t (instantaneous arrivals/departures, computed as $N_t - N_{t-1}$).

Results are summarized in Table 2. The first column indicates the plot the results refer to. The second column (N_t) indicates the range (min–max) of the number of concurrent requests which appear in the workload N_t, $t = 1, \dots, T$. The third column (*Nrec*) indicates the number

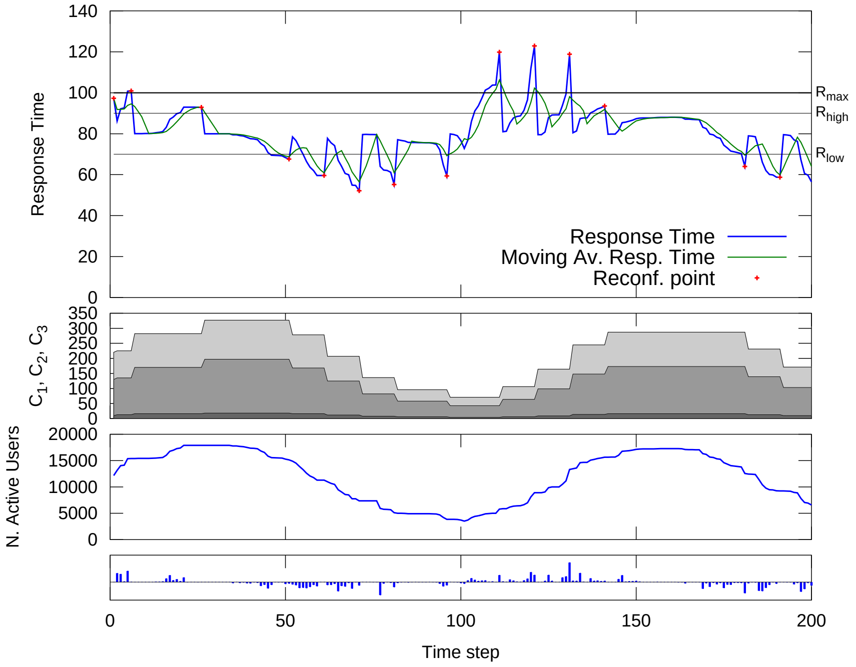

Figure 5. Simulation results: Test 2

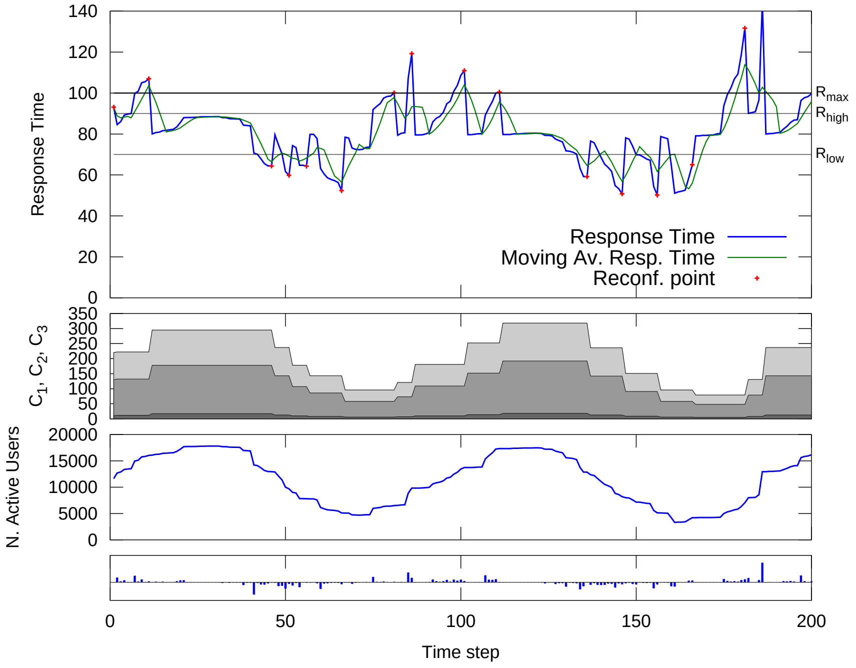

Figure 6. Simulation results: Test 3

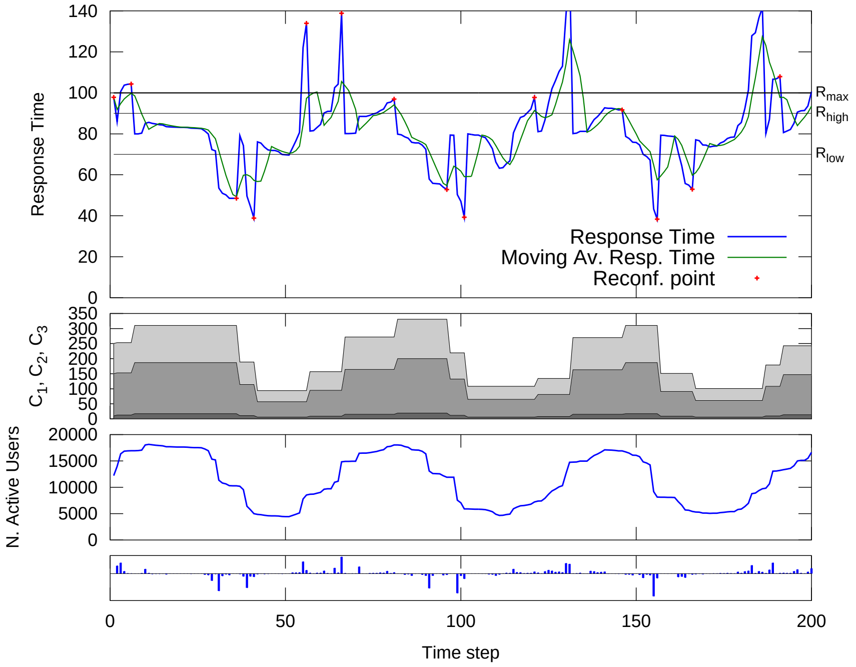

Figure 7. Simulation results: Test 4

	N_t	*Nrec*	*Nviol*	*Lviol*
	min–max			avg–max
Test 1	300–800	14	7	1.75/2.00
Test 2	3517–17895	14	10	2.50/5.00
Test 3	3319–17809	16	18	2.57/5.00
Test 4	4441–18157	16	24	3.43/6.00

Table 2. Simulation results

of reconfigurations which took place. The fourth column (*Nviol*) contains the number of steps in which the constraint $R < R_{\max}$ has been violated. Finally, in the last column (*Lviol*) we show the average and maximum number of contiguous steps in which the constraint $R < R_{\max}$ has been violated.

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 can be seen 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.

The number of violations of the Service Level Agreement (SLA), as reported in Table 2, is low; SLA violations mostly happen when there are surges in the number of concurrent users, as can be deduced by comparing the response times (top) and arrivals/departures (bottom) in each graph in Figures 4–7. If the workload fluctuates smoothly, as in Fig. 4, the response time is almost always kept below the threshold $R_{\max}$. If larger fluctuations happen, as in Fig. 5–7, our adaptation algorithm may require some time to react. The last column of Table 2 shows that, in the worst scenario (Test 7) the response time constraint is violated for at most 6 contiguous steps each time, with an average duration of 3.42 steps. This is a good result, especially considering that the number of concurrent users N_t fluctuates sharply from step to step. In this situation we can improve the responsiveness of the reconfiguration procedure by using lower values for the thresholds R_{low} and R_{high}.

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 [10] 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 [15] 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 [17] 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 [13] 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. 2.

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 [5] 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 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 [16] 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 is defined offline and cannot be modified at run time. Issues related to the SLA monitoring are presented in [7]. 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

[1] W. Cai, P. Xavier, S. J. Turner, and B.-S. Lee. A scalable architecture for supporting interactive games on the internet. In *Proceedings of the sixteenth workshop on Parallel and distributed simulation*, PADS '02, pages 60–67, Washington, DC, USA, 2002. IEEE Computer Society.

[2] K.-T. Chen, P. Huang, and C.-L. Lei. How sensitive are online gamers to network quality? *Commun. ACM*, 49:34–38, November 2006.

[3] M. Dick, O. Wellnitz, and L. C. Wolf. 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*, pages 1–7. ACM, 2005.

[4] J. W. Eaton. *GNU Octave Manual*. Network Theory Limited, 2002.

[5] S. Ferretti, V. Ghini, F. Panzieri, M. Pellegrini, and E. Turrini. QoS-Aware Clouds. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pages 321–328, 5–10 July 2010.

[6] T.-Y. Hsiao and S.-M. Yuan. Practical middleware for massively multiplayer online games. *IEEE Internet Computing*, 9:47–54, 2005.

[7] A. Korn, C. Peltz, and M. Mowbray. A service level management authority in the cloud. Technical Report HPL-2009-79, HP Laboratories, 2009.

[8] S. Kumar, J. Chhugani, C. Kim, D. Kim, A. Nguyen, P. Dubey, C. Bienia, and Y. Kim. Second life and the new generation of virtual worlds. *Computer*, 41(9):46–53, Sept. 2008.

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

[10] J. Li, J. Chinneck, M. Woodside, M. Litoiu, and G. Iszlai. 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, pages 15–22, Washington, DC, USA, 2009. IEEE Computer Society.

[11] J. D. C. Little. A proof for the queuing formula: $L = \lambda W$. *Operations Research*, 9(3):383–387, 1961.

[12] M. Mauve, J. Vogel, V. Hilt, and W. Effelsberg. Local-lag and timewarp: providing consistency for replicated continuous applications. *IEEE Transactions on Multimedia*, 6(1):47–57, 2004.

[13] V. Nae, A. Iosup, and R. Prodan. Dynamic resource provisioning in massively multiplayer online games. *IEEE Transactions on Parallel and Distributed Systems*, 99:1–15, Apr 2010.

[14] C. E. Palazzi, S. Ferretti, S. Cacciaguerra, and M. Roccetti. Interactivity-loss avoidance in event delivery synchronization for mirrored game architectures. *IEEE Transactions on Multimedia*, 8(4):874–879, 2006.

[15] S. Ranjan, J. Rolia, H. Fu, and E. Knightly. Qos-driven server migration for internet data centers. In *Quality of Service, 2002. Tenth IEEE International Workshop on*, pages 3–12, May 15–17 2002.

[16] J. Spillner and A. Schill. Dynamic sla template adjustments based on service property monitoring. In *Proceedings of the 2009 IEEE International Conference on Cloud Computing*, CLOUD '09, pages 183–189, Washington, DC, USA, 2009. IEEE Computer Society.

[17] B. Urgaonkar, P. Shenoy, A. Chandra, P. Goyal, and T. Wood. Agile dynamic provisioning of multi-tier internet applications. *ACM Trans. Auton. Adapt. Syst.*, 3(1):1–39, 2008.

[18] J. Zahorjan, K. C. Sevcik, D. L. Eager, and B. Galler. Balanced job bound analysis of queueing networks. *Commun. ACM*, 25:134–141, February 1982.