Dynamic Resource Provisioning in Massively Multiplayer Online Games

Vlad Nae, Alexandru Iosup, *Member*, *IEEE*, and Radu Prodan, *Member*, *IEEE*

Abstract—Today's Massively Multiplayer Online Games (MMOGs) can include millions of concurrent players spread across the world and interacting with each other within a single session. Faced with high resource demand variability and with misfit resource renting policies, the current industry practice is to overprovision for each game tens of self-owned data centers, making the market entry affordable only for big companies. Focusing on the reduction of entry and operational costs, we investigate a new dynamic resource provisioning method for MMOG operation using external data centers as low-cost resource providers. First, we identify in the various types of player interaction a source of short-term load variability, which complements the long-term load variability due to the size of the player population. Then, we introduce a combined MMOG processor, network, and memory load model that takes into account both the player interaction type and the population size. Our model is best used for estimating the MMOG resource demand dynamically, and thus, for dynamic resource provisioning based on the game world entity distribution. We evaluate several classes of online predictors for MMOG entity distribution and propose and tune a neural network-based predictor to deliver good accuracy consistently under real-time performance constraints. We assess using trace-based simulation the impact of the data center policies on the quality of resource provisioning. We find that the dynamic resource provisioning can be much more efficient than its static alternative even when the external data centers are busy, and that data centers with policies unsuitable for MMOGs are penalized by our dynamic resource provisioning method. Finally, we present experimental results showing the real-time parallelization and load balancing of a real game prototype using data center resources provisioned using our method and show its advantage against a rudimentary client threshold approach.

Index Terms—Distributed applications, modeling techniques, performance attributes, real time, games.

1 Introduction

Massively *Multiplayer Online Games (MMOGs)* have emerged in the past decade as a new type of large-scale distributed application characterized by a huge real-time virtual world entertaining millions of players spread across the world, e.g., World of Warcraft counts over six million unique players daily. Invented and promoted by industry, MMOGs have started to attract the interest of scientific researchers as well, and challenges such as scalability, trust, and data consistency have been identified by the distributed systems [1] and database [2] communities. In this paper, we draw the attention to the resource provisioning problem for MMOGs as a new direction of research.

Today's MMOGs operate as client/server architectures in which the game server simulates a world via computing and database operations, receives and processes commands from the clients, and interoperates with a billing and accounting system [3], [4]. Based on the actions submitted by the players, the game servers compute the global state of the game world represented by the position and interactions of the entities, and send appropriate real-time responses to the players containing the new relevant state information. Depending on the game, typical response times to ensure fluent play must be between 100 milliseconds in online *First Person Shooter (FPS)* action games [5] and 1-2 seconds for *Role-Playing Games (RPGs)*. A good game experience is critical in keeping the players engaged, and has an immediate consequence on the income of the MMOG operators. Failing to deliver timely simulation updates leads to a degraded game experience and triggers player departure and account closing [3], [4].

Today, a single computer is limited at around 500 simultaneous and persistent network connections, and databases can manage the update of around 500 objects per second [2]. To support at the same time millions of *active concurrent players* and many more other game entities, MMOG operators need to install and operate a large dedicated multiserver infrastructure [4], with hundreds to thousands of computers hosting the distributed load of each game [3]. However, due to the dynamic character of MMOGs, both on the short and long term, the game providers have to overprovision their infrastructure, which leads to a low and inefficient resource utilization and new providers finding it difficult to join the market. For example, the operating infrastructure of the *Massively Multiplayer Online Role Playing Game (MMORPG)* World of Warcraft has over 10,000 computers [6].

In this paper, we propose a dynamic resource provisioning solution that addresses the overprovisioning and the low-cost market joining problems. We first present in Section 2 a

- *V. Nae and R. Prodan are with the Distributed and Parallel Systems Group, Institute of Computer Science, University of Innsbruck, Technikerstrasse 21a, A-6020 Innsbruck, Austria.*
E-mail: {vlad, radu}@dps.uibk.ac.at.
- *A. Iosup is with the Parallel and Distributed Systems Group, Delft University of Technology, Mekelweg 4, 2628CD, Delft, The Netherlands.*
E-mail: a.iosup@tudelft.nl.

Manuscript received 23 Aug. 2008; revised 9 Apr. 2009; accepted 4 Sept. 2009; published online 5 Apr. 2010.
Recommended for acceptance by M. Parashar.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2008-08-0321.
Digital Object Identifier no. 10.1109/TPDS.2010.82.

realistic model for an MMOG ecosystem that describes the resource providers, the application, and their integration. Extending previous work where resources were provided by a single data center [7], [8], in our model, the MMOG resources are provided by multiple geographically distributed data centers with different lease policies. In contrast to the previous models of client/server Internet applications [7], [8], [9], [10], which focus on noninteracting user requests, our model is designed around the notion of player interactions within the same application instance. In Section 3, we demonstrate through the analysis of traces from the popular MMOG RuneScape [11] that MMOGs are more dynamic than previously believed [12] and that the interaction between players is another major component of the MMOG workload, complementing the player population size.

Motivated by the dynamic MMOG workloads, we propose a dynamic provisioning method in which the amount of resources is first predicted, and then obtained dynamically from data centers external to the MMOG operator. We devise in Section 4 an analytical MMOG load model for three types of resources: processor, memory, and network. The model takes into account both the player interaction type and the population size. By feeding real-time measurements or online predictions into the model, the game operators can dynamically estimate the amount of resources needed for running their MMOG. However, measuring the values of our proposed MMOG load model's parameters in real time may not be cost-efficient or even feasible due to the system scale and geographical spread. Moreover, player interactivity types are not uniform across the whole game, making real-time prediction difficult to adapt to MMOGs. We address these two problems in Section 5 by proposing a novel strategy for applying predictions to MMOG player interaction. We partition the simulated world into small areas, where real-time prediction proves to be accurate, and aggregate the estimates to get an overall prediction. We evaluate several classes of online predictors from moving averages to simple neural networks, and find that the simple neural networks have better accuracy than the studied alternatives for a variety of MMOG workloads.

In Section 6, we evaluate the efficiency and the Quality-of-Service (QoS) of our dynamic resource provisioning. We show through simulations that dynamic resource provisioning, which favors data centers that lease fewer resources at a time and for shorter periods of time considerably, reduces the MMOG operation costs with a reasonable loss of QoS, even when the centers are busy. We also find that when game operators use our dynamic resource provisioning, the data centers are incentivized to offer MMOG-friendly hosting policies when competition exists on the data center market.

We designed, implemented, and deployed our methods in the edutain@grid environment [13] targeting a platform for scalability, QoS provisioning, and user-friendly business models for real-time online interactive applications, with string focus on MMOG. In Section 7, we present a prototype experiment of applying our integrated methods on a real online game in a distributed infrastructure.

Finally, in comparison with the related work surveyed in Section 8, our study is the first to investigate the resource provisioning for a multi-MMOG, multidata center ecosystem.

2 MMOG Ecosystem Model

In this section, we introduce a model and platform for an MMOG ecosystem in which a global network of data centers host services that execute many MMOGs at the same time. Our multi-MMOG and multidata center model extends previous work, which focuses on either a single application (usually a web service) and/or a single data center [9], [14], [15].

2.1 Application Model

MMOGs are large-scale simulations of persistent *game worlds* comprising various objects or *entities* that we classify into four categories:

1. *Avatars* are in-game representation of the players.
2. *Bots* or *nonplayer characters* (NPCs) are mobile entities that have the ability to act independently.
3. *Movable objects* (such as boxes or guns) are passive entities which can be manipulated but do not initiate interactions.
4. *Immutable entities* or decor.

The mostly employed architectural model for MMOGs is client/server [3], with *game operators* maintaining the servers that simulate a distributed game world. The simulation consists for every MMOG of an identical set of steps to be executed each discrete game time unit (*game tick*), described in Section 4.1. The clients dynamically connect to a joint *game session* and interact with each other by sending play actions such as movements, shootings, operations on game objects, or chat. To ensure scalability and real-time response, an MMOG session is distributed on multiple game servers, and each player is mapped to an avatar on one of the servers, usually to one in its closest proximity to minimize latencies. We call the entities hosted by game server in a distributed session as *active entities* (from the point of view of that server). The vast majority of game servers follow a similar computational model implementing an infinite loop, where in each loop iteration (also called tick), there are certain steps to be performed: 1) processing events coming from the connected clients and other servers; 2) processing the states of the active entities; and 3) broadcasting state update to the connected clients.

2.2 Game Session Parallelization

Today's MMOGs use three main session parallelization techniques to serve at the same time hundreds of thousands of players: zoning, replication, and instancing. These techniques use the combined capacity of a group of servers working in parallel; in this sense, we use in this work the terms server and *server group* interchangeably. We describe in the following each of these techniques, in turn. In Section 7, we present experiments with a real game that uses all three techniques.

Spatial scaling of a game session is achieved through a conventional parallelization technique called *zoning* [16], based on similar data locality concepts as in scientific

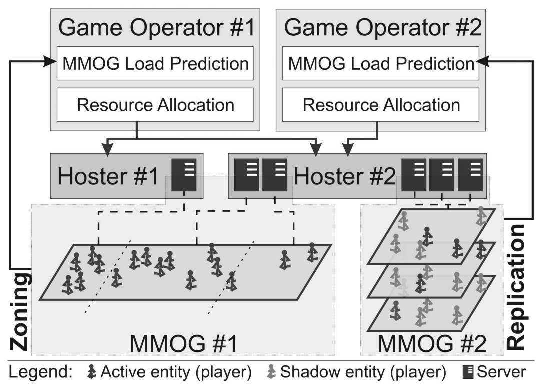

Fig. 1. The MMOG ecosystem architecture.

parallel processing. Zoning partitions the game world into geographical areas to be handled independently by separate machines (see Fig. 1). Zones are not necessarily of same shape and size, but should have an even load distribution that satisfies the QoS requirements. Today, zoning is successfully employed in slower paced (compared to fast-paced FPS action games) MMORPG, where the transition between zones can only happen through certain portals (e.g., special doors, teleportation, accompanied on the screen by a load clock or some standard animation video) and requires an important amount of time. Typically, zones are started manually by the game operators based on the current load, player demand, or new game world and scenario developments.

The second technique called *replication* [17] targets parallelization of game sessions with a large density of players located and interacting within each other's area of interest (see Fig. 1). Such situations are typical to fast-paced FPS action games in which players typically gather in certain hot-spot action areas that overload the game servers that are no longer capable of delivering state updates at the required rate. To address this problem, replication defines a novel method of distributing the load by replicating the same game zone on several CPUs. Each replicated server computes the state for a subset of entities called *active entities*, while the remaining ones, called *shadow entities* (which are active in the other participating servers), are synchronized across servers.

The third technique called *instancing* is a simplification of replication, which distributes the session load by starting multiple parallel instances of the highly populated zones. The instances are completely independent of each other, meaning that two avatars from different instances will not see each other, even if they are located at nearby coordinates.

2.3 Load Complexity Model

The load of an MMOG depends not only on the number of active concurrent players, but also on the number and type of their interactions (see Section 3.4 for empirical evidence). The interaction type and count span a wide range depending on the game design. The number of interactions between the entities may be very low (e.g., for puzzle games where a player interacts with the system after long periods of thinking), to low (e.g., for MMORPGs where small groups of people interact with a sparse environment), and to very high (e.g., for FPS action games where many players test their reaction time in a confined area).

Assuming that the number of entities is n, the interaction complexity may range from $O(n)$ for games in which players are mostly solitary or the game does not need to make many state changes or compute complex interactions, to $O(n^2)$ for games in which many players acting individually are interacting, and to $O(n^3)$ for games in which groups of many players are interacting. To reduce the computational load, most MMOGs simulate and send updates only for world regions representing the *area of interest* of each avatar [18]. When using such techniques, the interaction complexity may become $O(n \cdot \log n)$ from $O(n^2)$, and $O(n^2 \cdot \log n)$ from $O(n^3)$.

The lack of responsiveness of an MMOG may also be caused by high latency, independently from the game operator's server and bandwidth capacity. However, depending on the game design, MMOGs have different *latency tolerance*. The latency tolerance has been investigated by previous research [5], [19]; for example, for FPS, action games latencies above 100 ms severely disrupt the gameplay, while for MMORPGs, any latency below 1.5 s is tolerable.

2.4 Hosting Model

We consider the hosting platform as consisting of data centers scattered around the world, where each center pools together resources that may serve several game operators simultaneously (see Fig. 1). For simplicity, we assume that each data center consists of a single computing resource, which can be a shared or distributed memory parallel machine owned by a *hoster*. The game operators submit requests to the data center by specifying the type, number, and duration for which the resources are desired. We consider four resource types: *CPU*, *memory*, input from the external network (*ExtNet[in]*), and output to the external network of a data center (*ExtNet[out]*); here, the external network connects the data center with the Internet.

Depending on the data center's service model (either best effort or advance reservation-based), resource requests are queued or immediately fitted in the schedule, respectively. Once the available resources are matched against the requests, these resources are *allocated* to the game operators. From the game operator's point of view, we say that the resources have been *provisioned*. We use from here on the terms resource allocation and resource provisioning interchangeably. The allocated resources are reserved for executing the MMOG servers for the entire duration of the game operator's request (task preemption or migration is not supported).

Our hosting model considers the size and the duration of the minimal resource allocation, which may be not only for a resource as a whole (e.g., a server in web data centers [14] or a multiprocessor node in a Grid system), but also for a fraction of that resource (e.g., a virtual machine running on a physical node [10], or a channel of an optical network). Similarly, the minimal duration for which a resource may be allocated may be between a few seconds (servicing one user request by a web service) to several months (a typical value for web server hosting). We define the *resource bulk* as the minimum number of resources that can be allocated for one request, expressed as the multiple of a minimal

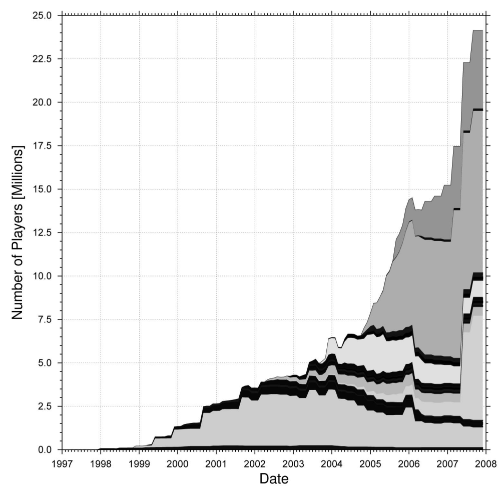

Fig. 2. Number of MMORPG players over time.

resource size. Similarly, we define the *time bulk* as the minimum duration for which a resource allocation can be performed expressed as multiple of a minimal time period. A data center may choose to allocate resources for MMOGs only in bulks under a space-time *hosting policy* imposed by the hosters.

2.5 Complete Ecosystem Model

The MMOG ecosystem comprises multiple hosters and multiple game operators, where each hoster may set a different space-time policy for its resources. The game operators handle simultaneously MMOGs of different genres and designs with different interactivity types and counts, and with different latency tolerance. The relative success of each game is characterized by the number of registered players. Fig. 2 displays the number of MMORPG players over time for the US and European markets based on the survey of Woodcock [20] for dates until June 2006, and on our own research afterward. The chart shows that there are currently six games with more than 500 thousand players each. The total number of MMORPG players is well approximated by the exponential trend $\alpha \cdot e^{\beta x}$, where $\alpha = 7 \cdot 10^{-9}$ and $\beta = 0.028$ give a Pearson's coefficient of determination $R^2 = 0.974$. Assuming the same rate of growth, we can estimate over 60 million players by 2011 in the US and EU markets alone. The large number of MMOG players, for the whole ecosystem and for each game in part, is an important motivation for our work.

The game operators make requests based on the load of their game servers and the data centers respond with offers based on their local time-space renting policy. The resource allocation is realized by a request-offer matchmaking mechanism according to three criteria that favor the game operator. First, the number and the type of resources requested must match with the offer, and when they do not match, an offer that includes at least the requested amounts is selected. Second, depending on the game latency tolerance, the resources closest to the request are preferred. Third, to deal with data center hosting policies, the finer grained resources with the shorter period of reservation time are preferred.

3 MMOG Workload Analysis

Previous work on MMOG workload characterization focused on highly interactive games with few users playing together [21], traced small- to mid-sized MMOGs [12], [22], or collected data from only one server from a large distributed MMOG [23]. In contrast, our analysis focuses on all server groups of one of the largest commercial MMOGs for which we analyze the workload at both server and network level based on server location and user interactions.

3.1 RuneScape Traces

RuneScape [11], ranked second by number of players in the US and European markets (see Fig. 2), is not a traditional MMORPG, but combines elements of RPG and FPS (and other genres) in specific parts of the game world called *minigames*, where player interaction follows different rules. Thus, various levels of player interactivity coexist and the game load cannot be trivially computed, for example, using the linear models employed in [15]. We started monitoring and collecting traces from the official RuneScape webpage [11] in August 2007. The traces are sampled every two minutes and contain the number of players over time for each server group used by the game operators. In this work, we analyze the traces for a period of over six months until March 2008.

3.2 Global Number of Players

The number of RuneScape players has surged over the past two years, starting with the introduction of the minigames. From 180,000 active players (i.e., players that played at least once in the last month) at the beginning of 2005, the game is estimated to have now over 5,000,000 active players from over 8,000,000 open accounts in 2007. Our study of the official list of top RuneScape players counted over 3,000,000 active players in September 2007. Since a player needs to be efficiently active for about a month to become a top player, we conclude that RuneScape converts into *dedicated* players between 30 and 60 percent of the starting players.

Our study shows that the maximum global number of active concurrent players for RuneScape is around 250,000. However, this number is strongly driven by the mood of the player base. Fig. 3 depicts the number of active concurrent users for RuneScape over a period of two months. A highly unpopular decision issued on 10 December 2007 resulted in massive account cancellations, which dropped the number of active concurrent players by over 30,000 units (a quarter of its value) in less than one day. Under intense pressure, the game operators agreed to amend the changes and the number of active concurrent players raised again to 95 percent of the previous value. On 18 December 2007 and 15 January 2008, the game operators released new content, which caused an over 50-percent surge of the number of active concurrent players after about one week in each case.

This high MMOG variability in the number of active and concurrent players determines a very dynamic resource requirement, which means that static resource provisioning would lead to significant overprovisioning that is a strong motivation for our work.

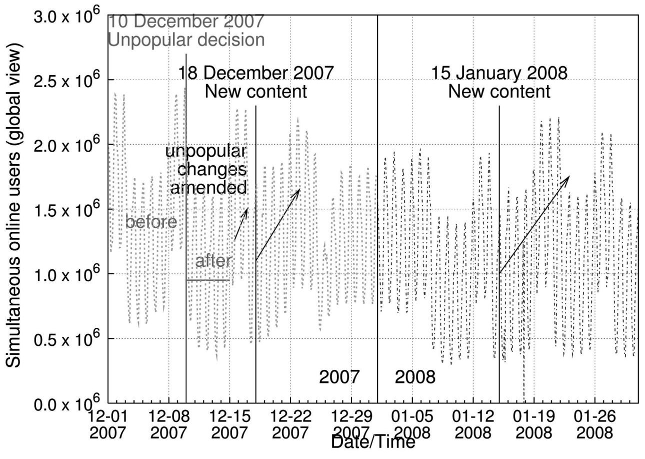

Fig. 3. The global RuneScape active concurrent players.

3.3 Patterns in the Regional Number of Players

Our RuneScape traces characterize a global population spread across five geographical regions including Europe and the US East and West Coasts. We present in this section the analysis of the number of players that use the server groups of region zero representing Europe. The input for this analysis is a subset of two weeks (middle to end of August 2007) from the original RuneScape traces aggregating over 11,000 data samples, where each sample contains one load value for each server in the region. The analysis of the other four regions, as well as of the rest of the traces, yielded very similar results. The top subplot of Fig. 4 shows the minimum, the median, and the maximum load measured in number of online users in any server group in the region at each time step. The median load shows a diurnal pattern and a strong load variation during the peak hours when the median is about 50 percent higher than the minimum. There are also some heavy fluctuations caused by few sporadic and short-lived server group outages, which fall outside the scope of this analysis.

To characterize the load variability between different server groups, the middle subplot depicts the load interquartile range (IQR) over time, defined as the difference between the 75th and 25th percentiles of a data set. Similar to the median load, the load variability has a diurnal cycle. Unlike the load of e-business and web servers [24], the median load shows no weekend effects, e.g., the load does not differ significantly between weekend and normal work days.

To establish the duration of the cycles observed in the top and middle subplots, the bottom subplot displays the autocorrelation function for each of the European server groups. We see a significant peak at around 720 (720 samples * 2 minutes per sample = 24 hours) and a strong negative peak at around 360 (12 hours), which shows again the diurnal pattern of the load of most servers. However, the same subplot shows that the load of 2-5 percent of the servers is always 95 percent except for outages, which does not follow a diurnal pattern.

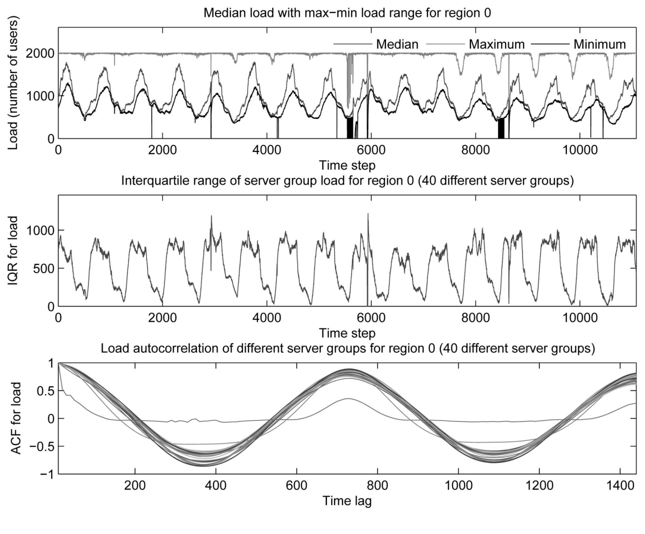

Fig. 4. RuneScape workload for region zero (Europe).

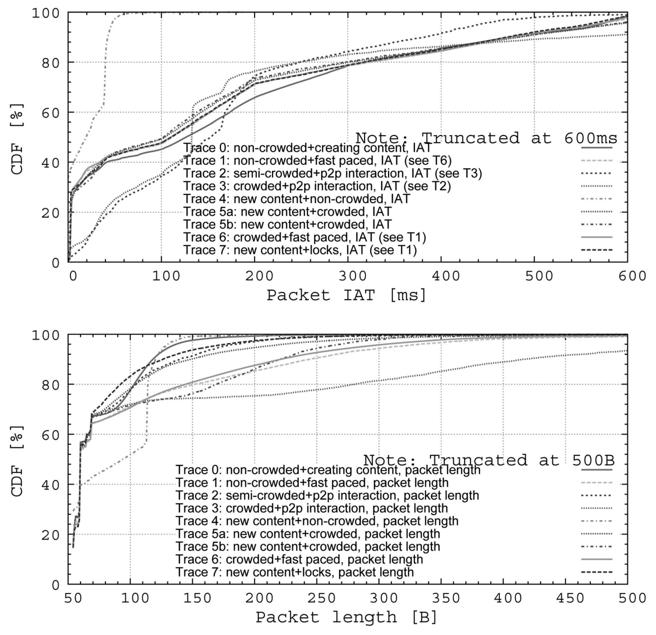

Fig. 5. The CDF of packet length (*top*) and the CDF of packet interarrival time (*bottom*) for different levels of player interaction.

3.4 Player Interaction Influence on Server Load

A fundamental premise of our work is that MMOG workloads depend on the player interaction. We demonstrate in this section that this is indeed the case. Using the `tcpdump` tool, we collected eight RuneScape traces labeled $T1, \ldots, T8$, and analyzed the packet size distribution and the interarrival time (IAT) between consecutive packets. Each trace is collected from a session of at least five minutes and at most one hour. To ensure the independence of the measurements, the traces were collected at different dates over a six month period, while the traces $T5a$ and $T5b$ were collected from the same environment at consecutive periods of time.

Fig. 5 illustrates that the network load depends on the number and type of player interaction. For traces $T1$ and $T6$ that involve a fast-paced game, the level of interaction (i.e., crowded or noncrowded) does not increase the server load as the players are very sensitive to delays. Thus, for fast-paced games, the server needs to send packets as often as possible and including as much information as possible. For traces $T2$ (market) and $T7$ (new content) that involve direct player-to-player interaction, the packet sizes are similar but their IAT is very different. The statistical moments of the IAT of $T7$ are lower than those of $T2$, as for $T2$ the need for updates is conditioned by players starting and agreeing to trades, with more thinking time than for the player actions in $T7$. For the trace $T4$ that involves group interaction, the packets need to arrive more often (lower IAT than for other traces) and to include information about more objects (higher packet size).

4 MMOG Load Modeling

We propose in this section an analytical model for the load of MMOGs. Our model covers three main types of resources used by MMOGs: CPU, memory, and network.

Let us consider N clients connected to a distributed game session aggregating a total of H (parallel, cluster) machines from different hosters. Let us further consider that the game world is populated by BE moving bots or NPCs (entity category 2 in Section 2.1). On each machine, there are only AE active entities and C clients connected.

4.1 CPU Load Model

For modeling the load of one machine in a distributed session, we distinguish three basic time-consuming activities within one game tick: 1) the computation of the interaction between pairs of entities (consumed time for each computation: t_i); 2) the reception of event messages from each client (t_m); and 3) the update of entity states received from/sent to another machine (t_u). In order to keep the complexity of the model acceptable, we assume that NPC entities do not interact, which is true for many MMOGs. We model the CPU time t_M spent for sending and receiving messages from a server to each client (active avatars) as

$$t_M = C \cdot t_m.$$

The CPU time t_U spent by the server for processing state updates from the other machines is

$$t_U = (N - C) \cdot t_u + (BE - AE) \cdot t_u + AE \cdot t_u,$$

and the time t_I spent by the server for computing the interactions between the active entities is

$$t_I = I \cdot t_i,$$

where I is the total number of interactions involving the active entities. Obviously, the computation of interactions that do not involve active entities is allotted to other machines.

For quantifying the interactions between entities, we use a generic function $f(e_1, e_2)$, which has to be instantiated for each interaction type introduced in Section 2.3:

$$f(e_1, e_2) = \begin{cases} e_1 + e_2, & for\ O\ (n)\ interaction; \\ e_1 \cdot \log(e_2), & for\ O\ (n \cdot \log(n))\ interaction; \\ e_1 \cdot e_2, & for\ O\ (n^2)\ interaction; \\ e_1^2 \cdot \log(e_2), & for\ O\ (n^2 \cdot \log(n))\ interaction; \\ e_1^2 \cdot e_2, & for\ O\ (n^3)\ interaction; \end{cases}$$

where e_1 and e_2 are two classes of interacting entities.

Let IC denote the number of avatars interacting with any other entities (either avatars or NPCs). Furthermore, we define p_{ci} as the average number of interactions involving active avatars entities expressed as a percentage of IC. Analogously, we define p_{ei} as the average number of interactions involving active NPCs expressed as a percentage of BE. The total number of interactions is composed of the number of interactions between active avatars and the number of interactions between active avatars and NPC entities:

$$I = p_{ci} \cdot f(IC, IC) + p_{ei} \cdot f(IC, BE).$$

Consequently, the CPU time t_I for processing the interactions involving all active entities can be calculated as follows:

$$t_I = (p_{ci} \cdot f(IC, IC) + p_{ei} \cdot f(IC, BE)) \cdot t_i.$$

Approximating the time for sending/receiving an event as equal to the time for updating the state of one entity ($t_m = t_u$), the total CPU time consumed in one tick becomes

$$t_C = (N + BE) \cdot t_u + (p_{ci} \cdot f(IC, IC) + p_{ei} \cdot f(IC, BE)) \cdot t_i.$$

Furthermore, quantifying t_i with regard to t_u as $t_i = p_{ui} \cdot t_u$, where $p_u i$ is the ratio between the time necessary for one entity update and the time for computing one interaction (in percentage), the CPU time consumed in one tick becomes

$$t_C = (N + BE + p_{ui} \cdot p_{ci} \cdot f(IC, IC) \\ + p_{ui} \cdot p_{ei} \cdot f(IC, BE)) \cdot t_i.$$

Finally, considering t_{SAT} as the tick saturation threshold, we can define the CPU load function:

$$L_{CPU} = \frac{t_C}{t_{SAT}} \\ = \frac{N + BE + p_{ui} \cdot p_{ci} \cdot f(IC, IC) + p_{ui} \cdot p_{ei} \cdot f(IC, BE)}{v},$$

where v is the CPU speed expressed as an integer representing the number of t_u-long tasks the CPU is able to perform in a t_{SAT}-long time interval.

4.2 Memory Load Model

The memory model is less complex than the processor load model, since all machines keep the entity state records for all entities participating in the game session. First, we take into account the game-dependent constants such as the amount of memory m_{game} needed to run the actual game engine with no game world loaded and no clients connected. Next, we define m_{world} as the amount of memory used for the game world being played. As for entity-related memory constants, let m_{cs} denote the amount of memory needed to store the state of one avatar, and m_{es} the amount of memory needed to store the state of an NPC entity. The interaction between entities does not have a significant impact on the memory load and we ignore it. Aggregating all these data, the memory consumption M on a machine taking part in a distributed session is

$$M = N \cdot m_{cs} + BE \cdot m_{es} + m_{game} + m_{world}.$$

As a consequence, the final memory load function is

$$L_{MEM} = \frac{M}{M_{machine}} = \frac{N \cdot m_{cs} + BE \cdot m_{es} + m_{game} + m_{world}}{M_{machine}},$$

where $M_{machine}$ represents the amount of memory available on a machine.

4.3 Network Load Model

In terms of network consumption, we define the outgoing network bandwidth usage for a machine running a server of a distributed game session as follows:

$$D_{out} = C \cdot d_{cout} + (H - 1) \cdot (C + AE) \cdot d_{updt},$$

where d_{cout} represents the amount of data sent to a client and d_{updt} the amount of data exchanged between machines for updating a single entity state.

The incoming network bandwidth usage for a machine in a distributed session is defined as

$$D_{in} = C \cdot d_{cin} + (N - C + BE - AE) \cdot d_{updt},$$

where d_{cin} is the amount of data received from a client.

Finally, we define an overall network load function as the maximum between the incoming and outgoing loads, since the network is congested once one of the two maxima is reached:

$$L_{NET} = \max\left(\frac{C \cdot d_{cout} + (H-1) \cdot (C + AE) \cdot d_{updt}}{B_{out}}, \frac{C \cdot d_{cin} + (N - C + BE - AE) \cdot d_{updt}}{B_{in}}\right),$$

where B_{in} and B_{out} denote, respectively, the input and output network bandwidths.

4.4 Complete Load Model

We integrate the presented models into a complete resource load model for MMOGs, where the load of the entire system is imposed by the maximum load of the individual resources:

$$L = \max(L_{CPU}, L_{MEM}, L_{NET}).$$

To stress the generality of our approach, MMOG classes can be defined using the set of constants involved in all the models previously described:

$$MMOG_{class} = \{(BE), (m_{cs}, m_{es}, m_{game}, m_{world}), (d_{cout}, d_{cin}, d_{updt})\}.$$

Note that BE is not MMOG-dependent, but rather game world and play style (e.g., single versus team play)-dependent. Nevertheless, we included it among the constants because we can consider games running different game worlds and play styles as belonging to different MMOG classes.

5 MMOG Load Prediction

In this section, we introduce an MMOG load prediction solution comprising a load prediction strategy and a tuned neural network-based predictor that offers good real-time prediction accuracy for a variety of MMOG workloads. We also present the evaluation of several classes of predictors leading to the selection of the neural network-based predictor.

5.1 MMOG Load Prediction Strategy

We have shown in Section 3 that the load of MMOGs is more dynamic than previously believed, mostly because of the player interactions. Therefore, fast and accurate load prediction algorithms are required to dynamically allocate resources for MMOGs, which, besides the entity count, also consider the entity interaction. However, due to the different play styles of the players logged in at different moments of time, the number of interactions can vary greatly. Thus, predicting the number of interactions for the whole game in real time leads to low accuracy. To address this problem, our prediction strategy for MMOGs is to first partition the game world into subareas, where the size of a subarea is small enough to be characterized by the entity count and interaction type (see Section 2.3). Then, the entity count for each subarea is used as input for the model described in Section 4. We define the overall entity distribution as the set of entity counts for each subarea. The overall MMOG load prediction is obtained by using the entity distribution as input to many instances of the load model.

The main problem to be solved for this strategy is finding a predictor with high accuracy for a variety of MMOG workloads and good performance. As the overall entity distribution is required for one overall MMOG load prediction, the predictor needs to be able to deliver tens of thousands of predictions per second. In the remainder of this section, we address the problem of finding such a predictor.

5.2 Predictor Families

Two options are available for quantitative predictions in MMOGs: explanatory models and time series prediction. While explanatory models can deliver good accuracy with little computation, they are difficult to obtain for complex applications such as MMOGs, and are tightly coupled to the application instance and sometimes to platform for which they have been constructed. With MMOGs relying on frequent and large updates to maintain interest among the players (in Fig. 3, we see a rate of one update per month), the explanatory models quickly become unmaintainable. Thus, our work is based on prediction algorithms that use historical values to discover patterns in the historical data series, and extrapolate these patterns into the future.

Many such prediction algorithms have already been proposed [25]. Simple prediction algorithms like exponential smoothing and variants thereof are computationally inexpensive and can be applied in parallel on several data sets, but their predictive power is limited. More elaborated prediction algorithms like autoregressive (AR), integrated (I), moving average (MA) models, and combinations thereof like ARMA or ARIMA try to find the best prediction model for the given data set. Although their predictive power is higher, such methods are also more time-consuming and resource intensive, thus, being ill suited for highly dynamic MMOGs.

5.3 Neural Network-Based Prediction

As an alternative to other prediction algorithms, we investigated the use of neural networks [26] that provide a robust approach to approximating real or discrete-valued

TABLE 1
Configuration Parameters of the Trace Data Simulation Sets

Data set	Player behaviour [%]				Peak hours modelling	Peak load	Overall dynamics (17 h.)	Instantaneous dynamics (2 min.)
	Aggressive	Scout	Team player	Camper				
Set 1	80%	10%	0%	10%	No	+++	+++++	+++
Set 2	60%	10%	0%	20%	No	+	+++	++++
Set 3	70%	20%	0%	10%	No	++++	++++	++++
Set 4	70%	30%	0%	0%	No	++	++++	+++++
Set 5	30%	40%	30%	0%	Yes	++++	++	+++
Set 6	10%	80%	10%	0%	Yes	+++++	+	+
Set 7	20%	40%	40%	0%	Yes	++++	+++++	++
Set 8	20%	80%	0%	0%	Yes	++++	+	+

target functions. Low-complexity neural networks are capable of approximating complex and noisy functions provided that good input preprocessing is performed. Our predictor uses one separate neural network for each game subarea, which receives as input the entity count at equidistant past time intervals and delivers as output the entity count at the next time step.

A downside of neural networks is that they require a long offline training phase consisting of several subphases. First, the *data set collection* phase is a long process in which the game is observed by gathering entity count samples for all subareas at equidistant time steps. The second *training* phase uses most of the collected samples as training sets, and the remaining ones as test sets. The training phase runs for a number of eras, until a convergence criterion is fulfilled. A training era consists of three steps: 1) presenting all the training sets in sequence to the network, 2) adjusting the network's weights to better fit the expected output (the real entity count for the next time step), and 3) testing the network's prediction capability with the different test sets. Separating the training from the test sets is crucial for avoiding memorization and ensures that the network has enough generalization potential for delivering good results on new data sets.

5.4 Traces for Testing MMOG Predictors

To experiment and validate the neural network prediction, we developed a distributed game simulator, which realistically emulates the behavior of game players. The motivation for using an emulator is twofold: 1) we do not have available the exact coordinates of entities in the RunScape game world (and we do not have access to the code or the documentation of the RuneScape server either) and 2) through this emulator, we are able to give further evidence that the player interaction determines the server load (see also Section 3.4).

5.4.1 Emulator

We have designed an MMOG emulator, which simulated a virtual environment populated by avatars. The environment consists of real maps from the popular FPS game Counter Strike [27]. The avatars are driven by several Artificial Intelligence profiles that match the four behavioral patterns most encountered in MMOGs [3]: the *achiever* determines the avatar to seek and interact with opponents; the *socializer* causes the avatar to act in a group together with its teammates; the *explorer* leads the avatar for discovering uncharted zones of the game world (not guaranteeing any interaction); and the *killer* simulates a well-known tactic in FPS games to hide and wait for the opponent (thus, gaining a tactical advantage through the element of surprise). To also account for the mixed behavior encountered in deployed MMOGs [3], each entity has its own preferred profile, but can change the profiles dynamically during the emulation.

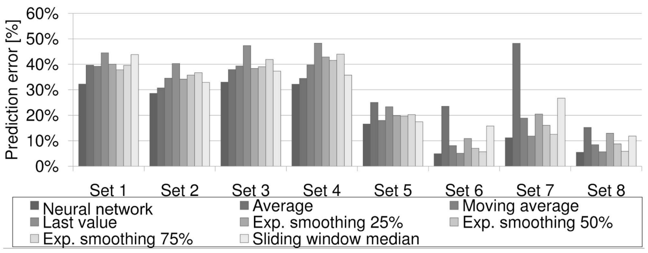

Fig. 6. Accuracy of seven prediction algorithms applied to MMOG data.

5.4.2 Generated Traces

We used this emulator to generate eight different data traces for a duration of one day each with a sampling rate of two minutes, modeling four parameters (see Table 1): peak hours, peak load, overall dynamics, and instantaneous dynamics. The peak hours correspond to the periods with high player count in online gaming such as late afternoons (see Section 3). The peak load represents the highest load observed in an MMOG, which is a good measure for its relative popularity. The overall dynamic represents the variability of the entity interaction over a period of one day, while the instantaneous dynamic indicates the same variability over a period of two minutes. The eight data sets exhibit three major types of signals: *type I* with high instantaneous dynamics and medium overall dynamics (sets 2-4); *type II* with low instantaneous dynamics (sets 6-8); and *type III* with medium instantaneous dynamics (sets 1 and 5).

5.5 Prediction Results

In this section, we compare the neural network prediction against other well-known fast prediction methods such as last value, average, moving average, sliding window median, and three levels of exponential smoothing. We used a well-tuned Multilayer Perceptron with a $[6, 3, 1]$ structure representing the number of neurons on each layer, which delivered the most accurate results in a series of offline network tuning experiments [28]. The goal of our experiment is to minimize the *prediction error*:

$$PE = \frac{\sum_{i=1}^{N} \left| n_i^{(real)} - n_i^{(pred)} \right|}{\sum_{i=1}^{N} n_i^{(real)}},$$

where N is the total number of samples in the trace data set, and $n_i^{(real)}$ and $n_i^{(pred)}$ are, respectively, the real and predicted entity counts at time step i.

Each prediction method receives as input every trace data set described in Table 1, and outputs for each input set a sample prediction. The results in Fig. 6 show that, apart from having lower prediction errors, the quality of our neural network-based predictor is its ability to adapt to various types of input signals. In contrast, other algorithms

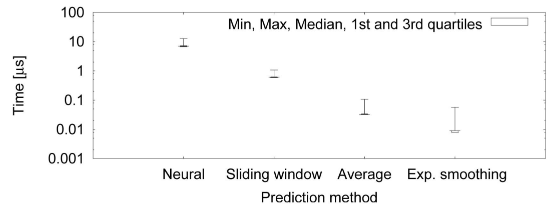

Fig. 7. Statistical properties of the duration of one prediction for four prediction algorithms applied to MMOG data.

exhibit poor performance for some types of signals (e.g., the average method is the second most accurate for Type I signals, but performs poorly for all Type II and for some Type III signals). Notably, our neural network predictor was significantly better than the others for the sets with high instantaneous dynamics (Types I and III signals).

Fig. 7 depicts the duration of one prediction on an Intel Core Duo E6700 (2.66 GHz) processor. Although the neural network predictor is the slowest with an average prediction duration of approximately 7 microseconds, it is nevertheless fast enough and suitable to MMOGs. To justify this statement, let us consider a RuneScape setup consisting of a hoster managing 200 game servers, each one with an upper client limit of 2,000. Moreover, let us consider the worse case scenario in which our predictor would use a number of subareas equal to the number of players, and that all game worlds are handling the maximum number of players (i.e., 250,000 subareas in total, see Section 3.2). The approximate total prediction time using our proposed predictor would be 14 milliseconds per server, and 2.8 seconds per hoster considering a single threaded execution. From a realistic prediction interval of 2 minutes, the prediction time would amount approximately 2 percent, leaving the remaining 98 percent for resource allocation and load balancing actions.

6 MMOG Resource Provisioning

In this section, we present an evaluation of our MMOG resource provisioning model described in Section 2.5. As outlined in Table 2, we cover an experimental space with six axes: the provisioning mechanisms, the prediction algorithms, the player interaction, the hosting policies, the latency tolerance, and the multi-MMOG workloads.

6.1 Experimental Setup

6.1.1 Metrics

We evaluate each experiment by using three metrics: resource overallocation, resource underallocation, and number of significant underallocation events.

TABLE 2
The Experimental Space Coverage

Section	Resource provisioning	Prediction algorithm	Update models	Hosting policies	Latency tolerance	Number of MMOGs
Section VI-B	**static, dynamic**	**all**	$O(n^2)$	optimal	none	one
Section VI-C	dynamic	Neural	**all**	optimal	none	one
Section VI-D	dynamic	Neural	$O(n^2)$	**all**	none	one
Section VI-E	dynamic	Neural	$O(n^2)$	optimal	**all**	one
Section VI-F	dynamic	Neural	$O(n^2)$	optimal	none	**several**

TABLE 3
Data Center Physical Characteristics

Location: Continent	Location: Country	Data Centres	Machines (total)
Europe	Finland	2	8
	Sweden	2	8
	U.K.	2	20
	Netherlands	2	15
North America	U.S. (West)	2	35
	Canada (West)	1	15
	U.S. (Central)	1	15
	U.S. (East)	2	32
	Canada (East)	1	10
Australia	Australia	2	8

The *resource overallocation* characterizes the percentage of a resource (i.e., CPU, memory, network) that has been allocated from the amount used for the seamless execution of an MMOG session. We define the resource overallocation at time instance t as the cumulated overallocation for all M resources participating in the game session, where $\alpha_m(t)$ is the total allocated resource and $\lambda_m(t)$ is the resource usage (the generated load):

$$\Omega(t) = \frac{\sum_{m=1}^{M} \alpha_m(t)}{\sum_{m=1}^{M} \lambda_m(t)} \cdot 100[\%].$$

The *resource underallocation* characterizes the percentage of resources that have not been allocated from the amount necessary for the seamless execution of the MMOG, but considering that missing resources on one machine can be hidden by overallocating the resource on other machines. We define resource underallocation at time instance t as the average difference between the allocated and used resources. The $\min$ function limits the maximum underallocation value to at most zero, and thus, an overallocation at a certain time does not reduce impact of an underallocation at a different time instance (that is, $\Omega(t)$ and $\Upsilon(t)$ are not correlated):

$$\Upsilon(t) = \frac{\sum_{m=1}^{M} \min(\alpha_m(t) - \lambda_m(t), 0)}{M} \cdot 100[\%].$$

The *number of significant underallocation events* indicates the number of times the underallocation causes game play disruption over a long period of time. In this work, we consider an underallocation as being disruptive (and frustrate players that may quit the game) if its absolute value is over 1 percent for a period of time of at least two minutes.

6.1.2 Environment

We performed experiments in a simulated RuneScape-like environment. The input workload consisted of the first two weeks from the RuneScape trace data analyzed in Section 3, which, with the metrics being evaluated every two minutes, gives over 10,000 metric samples for each simulation, ensuring statistical soundness. The data centers are located on four continents and seven countries, as depicted in Table 3. Each machine in the specified setup is capable of handling at least one game server at full load (e.g., 2,000 simultaneous clients for RuneScape). The data centers were configured to use different hosting policies, where each policy describes one time and one resource bulk for each type of resource (see

TABLE 4
Hosting Policies

Hosting policy	CPU	Memory	External net. in	External net. out	Time [minutes]
HP-1	0.25	n/a	6	0.33	360
HP-2	0.25	n/a	4	0.5	360
HP-3	0.22	2	n/a	n/a	180
HP-4	0.28	2	n/a	n/a	180
HP-5	0.37	2	n/a	n/a	180
HP-6	0.56	2	n/a	n/a	180
HP-7	1.11	2	n/a	n/a	180
HP-8	0.37	2	n/a	n/a	360
HP-9	0.37	2	n/a	n/a	720
HP-10	0.37	2	n/a	n/a	1440
HP-11	0.37	2	n/a	n/a	2880

Section 2.4). The measurement unit for the policy resources is a generic "unit," which represents the requirement for the respective resource in a fully loaded RuneScape game server (e.g., one external outward network unit is equivalent to a real bandwidth value of 3 MB/s—see also Fig. 5).

In our simulation, the game operators perform a prediction of the game load (that is, of the number of players and of the interactions per zone) every two minutes and request an appropriate amount of resources to the data centers. The protocol how resources are matched based on their number, type, location, and time in this ecosystem was presented in Section 2.5. We assume zero overhead in resource allocation, provisioning, and setup from data centers to game operators.

6.2 Prediction Impact

In this experiment, we evaluate the impact of the prediction method on the provisioning process. We assign the *HP-1* and *HP-2* hosting policies described in Table 4 to the data centers presented in Table 3 in a round-robin fashion. When two data centers have the same location (column "Country" in Table 3), their hosting policies are set differently to *HP-1*, respectively, *HP-2*, and their machine size to half the number of resources at that location (column "Machines" in Table 3).

We first compare the performance of our dynamic provisioning strategy using one of the six prediction algorithms (see Section 5), as presented in Table 5. For overallocation, we observe two performance classes: the poor performance class with one member (the average predictor) and the normal performance class with the other five predictors as members. The reason for the big overallocations for the external network bandwidth is that the two utilized policies were not well fitted to the input workload (i.e., the policies in Table 4 included too much external network bandwidth relative to the CPU). We will show in Section 6.5 that the resources from data centers with such policies are not used when other suitable alternatives exist.

We further rank the five predictors from the normal performance class using the underallocation metric. First, only the neural network and the last value predictors lead to no underallocation of the external network bandwidth, in addition to leading to the lowest CPU underallocation. Fig. 8 depicts the cumulative number of significant underallocation events over time for the five predictors with a normal overallocation value, which shows that the neural network predictor exhibits the lowest number and the most stable evolution. We conclude that our novel neural network predictor enables the best resource provisioning, followed by the last value predictor which confirms our results from Section 5.5.

Fig. 9 comparatively displays the resource overallocation resulted from the use of static, respectively, dynamic provisioning strategy (using the neural network predictor) for the same workload. As expected, the dynamic provisioning of resources achieves better results, its average overallocation being around 25 percent, compared to 250 percent for the static allocation. The overallocation for dynamic provisioning is not the outcome of unreliable predictions and can be lowered if the data centers policies are more favorable. In this experiment, the deallocation of resources was only allowed at least six hours after the start of the allocation (column "Time" in Table 4).

6.3 Player Interaction Impact

In this section, we study the impact of player interaction on the dynamic provisioning using the neural network predictor.

Fig. 10 shows the resource over- and underallocation over time for the $O(n)$, $O(n \cdot \log n)$, $O(n^2)$, $O(n^2 \cdot \log n)$, and $O(n^3)$ MMOG update models described in Section 2.1, for dynamic resource provisioning. We observe that the higher the complexity of the update model is, the greater the fluctuations in resource overallocation are. At the same time, the significant underallocation events become more frequent as the complexity of the update model increases.

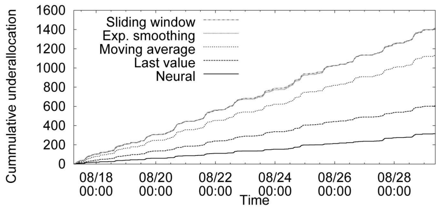

Fig. 8. Cumulative underallocation events for five predictors.

TABLE 5
Dynamic Resource Allocation Results

Predictor type	Avg. Over-allocation [%] CPU	ExtNet [in]	ExtNet [out]	Avg. Under-allocation [%] CPU	ExtNet [in]	ExtNet [out]	$\vert\Upsilon\vert > 1\%$ events
Neural network	**25.90**	**995.27**	**66.04**	**-0.09**	**0**	**0**	**317**
Average	32.41	1023.43	69.29	-12.84	0	-2.46	8123
Last value	**25.11**	**989.10**	**65.36**	**-0.16**	**0**	**0**	**608**
Moving average	24.92	992.06	65.69	-0.33	0	-0.03	1142
Sliding window	24.97	992.73	65.76	-0.41	0	-0.03	1423
Exponential smoothing	24.76	977.85	64.11	-0.42	0	-0.03	1429

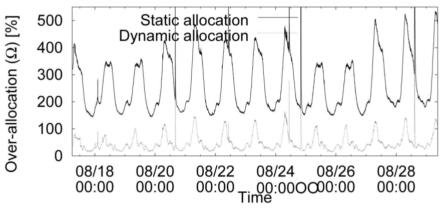

Fig. 9. Static versus dynamic resource overallocation.

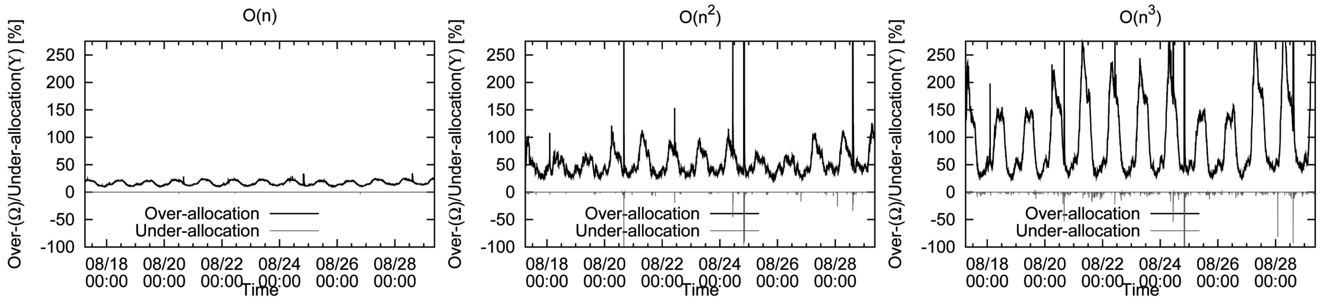

Fig. 10. Over and underallocation for three update models.

This is further confirmed in Fig. 11, which depicts the cumulative number of significant underallocation events over time, at the end of the two simulation weeks, this number being significantly higher for $O(n^3)$ than for $O(n)$.

Table 6 compares the average static and dynamic resource provisioning mechanisms for various interaction types. The static provisioning has five to seven times higher resource overallocation than the dynamic one, but no underallocation events. However, the number of significant underallocation events over the whole simulated period remains below 3 percent for dynamic provisioning (at most 30 samples from over 10,000 samples used in the simulation). When even this low occurrence cannot be tolerated, a mechanism that increases the overallocation shall be used.

6.4 Data Center Hosting Policy Impact

We further evaluate the influence of the hosting policies on the dynamic resource provisioning, where each hosting policy expresses the sizes of the resource and of the time bulks (see also Section 2). When more resource types are involved in the matchmaking process, there is a separate bulk size for each resource type. Because the resource and the time bulks have a combined influence on the provisioning of resources, we conduct three separate experiments that use a different resource hosting policy setup each. We first study in Section 6.4.1 the individual impact of the resource and time bulk parameters on resource provisioning by varying one of them and keeping the other constant. Then, we study the combined effect of the resource and time bulks by using a heterogeneous hosting policy setup in Section 6.4.2.

6.4.1 The Independent Impact of the Resource and Time Bulks

We first estimate the impact of the CPU resource bulk variation on the resource allocation performance by using five hosting policies from *HP-3* to *HP-7* in Table 4. The resource bulks for other resource types and the time bulk are kept constant. The values selected for the CPU resource bulk, i.e., from 0.22 to 1.11, are not evenly distributed in the selected interval, which reflects the real-life policies of data centers that try to maximize their own resource usage and do not willingly adapt to a specific MMOG's resource requirements. We will show in Section 6.5 that in our MMOG ecosystem, the game operators are not always forced to accept such conditions and can penalize the data centers with unsuitable hosting policies by not using their resources.

Experiment batch A in Table 7 illustrates the influence of the CPU resource bulk on the dynamic resource provisioning. There is a visible tendency of higher overallocation values for bigger CPU resource bulks. Conversely, we can observe an increase in significant underallocation events as the CPUs are offered with finer grained quantities. In conclusion, the finer grained policies have the potential to increase the resource provisioning efficiency, but also the risk of increasing the number of significant underallocation events. An optimal value for the resource bulk granularity can be determined with respect to the type of game serviced and its tolerance to resource shortages.

To observe the impact of the time bulk, we vary it from 0.1 to 2.0 days while keeping the resource bulks constant. To this end, we use the policies *HP-5*, and *HP-8* to *HP-11* described in Table 4. The results of experiment batch B presented in Table 7 show that the efficiency of resource provisioning can be significantly improved by using resources from the data centers whose policies specify the shortest time bulks. The increase of the average underallocation is low if the time bulks are set to realistic values, e.g., above one hour.

6.4.2 The Combined Impact of the Resource and Time Bulks

To study the combined impact of the resource and time bulks, we aggregate the policies used in the previous two

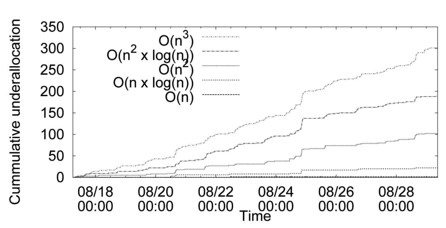

Fig. 11. Cumulative underallocation events.

TABLE 6
Static versus Dynamic Provisioning for Various Interactions

Interaction type	Static provisioning Over-allocation [%]	Dynamic provisioning Over-allocation [%]	Under-allocation [%]	$\|\Upsilon\| > 1\%$ events
$O(n)$	55.71	8.47	0	1
$O(n \cdot log(n))$	71.86	16.07	-0.024	22
$O(n^2)$	146.03	27.77	-0.066	103
$O(n^2 \cdot log(n))$	180.60	36.26	-0.096	191
$O(n^3)$	242.04	54.62	-0.130	304

TABLE 7
Experimental Results Showing the Impact of Different Resource and Time Bulks on Resource Provisioning

Experiment batch	Allocation time [hours]	CPU resource [units]	Over-allocation [%]	Under-allocation [%]	Under-allocation events
A: Time bulk variation	*3*	0.56	57.1	-0.028	78
	6	0.56	60.78	-0.022	62
	12	0.56	68.42	-0.020	54
	24	0.56	77.79	-0.014	46
	48	0.56	78.75	-0.008	23
B: Resource bulk variation	3	*1.11*	112.79	0	0
	3	*0.56*	51.03	-0.048	90
	3	*0.37*	57.1	-0.028	78
	3	*0.28*	46.62	-0.064	137
	3	*0.22*	38.95	-0.061	151
C: Time bulk variation [average over all resource bulk sizes]	*3*	*0.22 – 1.11*	56.83	-0.047	127
	6	*0.22 – 1.11*	59.86	-0.040	108
	12	*0.22 – 1.11*	66.36	-0.029	91
	24	*0.22 – 1.11*	79.93	-0.018	70
	48	*0.22 – 1.11*	81.16	-0.011	41

experiments by taking all possible CPU-time bulk combinations (the other resource bulks are identical for all involved policies). The results from the experiment batch *C* depicted in Table 7 follow the same pattern as those obtained when varying the time bulk in Section 6.4.1 with higher under-allocation values. We conclude that changing the time bulk has a higher impact on the dynamic resource provisioning than changing the resource bulk.

6.5 MMOG Latency Tolerance Impact

We now investigate the impact of the MMOG latency tolerance on the quality of the dynamic resource provisioning. We consider an ideal network behavior, where the latency between players and data centers is exclusively determined by the physical distance between them. The higher the latency tolerance of an MMOG, the further away the servers can be located from the users, and the longer the list of data centers from which resources can be dynamically provisioned is. We show in this experiment that higher latency tolerance leads to resources of the data centers with unsuitable hosting policies being unused when other suitable alternatives exist.

We define five classes of maximal physical distance between the players and the server locations, where, in practice, the distance values would depend on the design of each MMOG:

1. *Same location*, when users must be handled by resources at the same location.
2. *Very close*, when resources can be allocated within a radius of 1,000 km from their users.
3. *Close*, within a radius of 2,000 km.
4. *Far*, within a radius of 4,000 km.
5. *Very far*, when any server can serve any user.

We consider from the setup described in Table 3 only the data centers located in the North American region, and select from the workload only the requests that arrive at these data centers. The hosting policies are coarse grained (i.e., with large resource and time bulks) for the data centers located on the US East Coast and become gradually finer grained for the data centers located at the Central and West Coast locations.

We first consider a restricted workload in which only the data centers from the US East Coast location receive allocation requests from game operators. Fig. 12 shows the distribution of the allocated resources for various latency tolerance values. As expected, because of the hosting policies setup, the desirability for a data center increases with its distance from the US East Coast. For the maximal latency tolerance, all requests are served on resources located at the maximal distance from the US East Coast, i.e., on the data centers from the US West Coast.

We now investigate the system behavior in a realistic situation under the combined workload of all North American game operators. Fig. 13 shows for this setup the distribution of the allocated resources for various latency tolerance values. Due to the resource contention, the resource allocation follows different patterns than in the optimal case. Fig. 14 depicts the resource allocation for all North American data centers, which demonstrates that the US East Coast data centers with the most unsuitable hosting policies are penalized by having more unused resources,

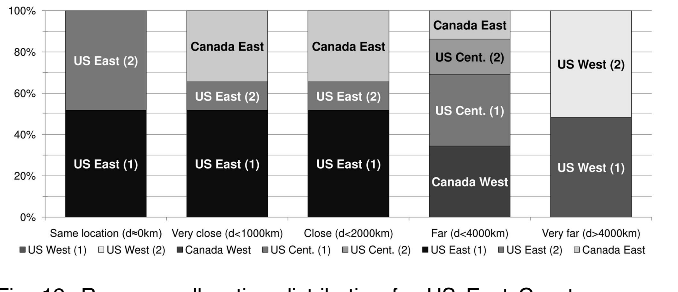

Fig. 12. Resource allocation distribution for US East Coast resource requests only and various latency tolerance values.

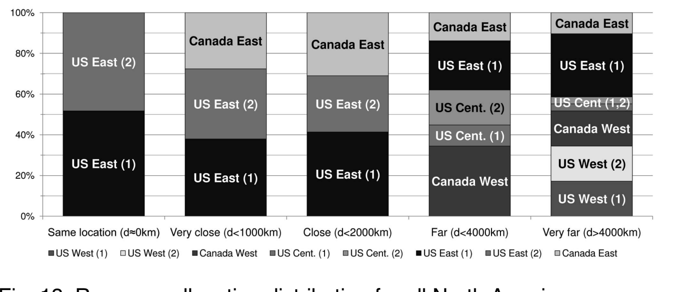

Fig. 13. Resource allocation distribution for all North American resource requests and various latency tolerance values.

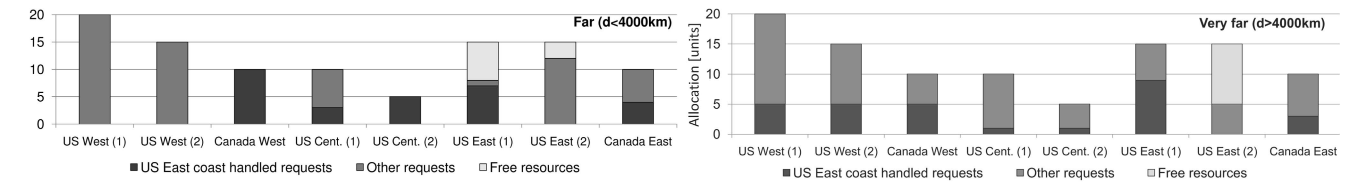

Fig. 14. Far maximal allocation distance (left) and very far maximal allocation distance (right) for all North American data centers and resource requests.

especially when the latency tolerance admits far and very far maximal service distances. In addition, the US East Coast requests are served under the best policies even in a busy system.

6.6 Impact of Servicing Multiple MMOGs

In this last experiment, we evaluate the dynamic provisioning when servicing multiple types of MMOGs. We select three MMOG types with different interaction complexities, as defined in Section 6.3: *MMOG A* uses an $O(n \cdot \log(n))$, *MMOG B* an $O(n^2)$, and *MMOG C* an $O(n^2 \cdot \log(n))$ interaction complexity. We run seven scenarios in which the system has to handle resource requests for the three selected MMOG types in different proportions. The workload structure for each scenario is depicted in the "Workload structure" group of columns from Table 8.

When the workload is dominated by the more computationally intensive B or C MMOGs, i.e., the first six rows in Table 8, the behavior of dynamic provisioning is stable. The overallocation under a workload of only C MMOGs is less than 3 percent higher than under a workload of only B MMOGs. In the seventh scenario, when the workload comprises only the (less computationally intensive) A MMOGs, the dynamic resource provisioning is significantly better than in the other scenarios. We conclude that the quality of the dynamic provisioning is determined by its biggest consumer. Under these circumstances, game operators of a different MMOG type (e.g., type A) may find it more convenient to install their own infrastructure. As an alternative, we plan to investigate in future work the impact of prioritizing resource requests according to the MMOG interaction complexity.

7 Real-World Experiment

We designed and implemented our MMOG ecosystem within the edutain@grid project [13], aiming to provide a platform for scalability, QoS provisioning, and user-friendly business models for real-time online interactive applications, with strong focus on MMOGs. edutain@grid is using as pilot a commercial FPS game application called Hunter developed by the Darkworks game development company with the headquarters in Paris, France. Parallelization of a game session according to the zoning, mirroring, and instancing techniques outlined in Section 2.2 is achieved by means of a generic portable library called Real-Time Framework (RTF) [29].

We used in our experiments an FPS demonstrator application that uses the open-source three-dimensional graphics engine (OGRE—http://www.ogre3d.org/) for graphics and sound, and the RTF library for game session parallelization. The resource testbed consists of six servers provided by the Amis telecommunication company located in Maribor, Slovenia and managed by one resource allocation service. The game world consists of two adjacent zones, handled in the beginning by two idle game servers. Clients connect to a game operator located at the University of Innsbruck, which processes their requests and arranges for the real-time connection the game servers. To stress the servers, we generated waves of incoming clients, implemented as nonplayer characters (or bots) and modeled using the profiles described in Section 5.4.

We ran three distributed sessions using different resource provisioning methods. One session used our dynamic provisioning method and the other two were managed using a fixed threshold on the number of clients to create replication servers and migrate clients. We set the replication thresholds to 40 and 50 clients, respectively. The goal of each session was to accommodate a total of 190 clients, which connect in four waves (one minute apart from each other) of 80, 30, 30, and 50 clients with as little resource utilization as possible, and at the same time providing good QoS for the clients (i.e., the least underallocation).

Fig. 15 shows three histograms with the total number of clients, their zone distribution, the resource provisioning, and load balancing measures taken for the distributed session using our provisioning method. Fig. 16 presents the underallocation events for the three distributed sessions. The average underallocation for the session managed by our dynamic resource provisioning method is 0.66 percent, and for the two sessions utilizing the client threshold is 0.86 and 8.69 percent, respectively. Although the underallocation for the client threshold method with a threshold of 40 is relatively close to the one exhibited by our method, there is a difference in the resource utilization. In this case, the client threshold method utilizes a total of six machines, whereas our provisioning method only needed five machines to accommodate the same amount of clients. It is possible to reduce the resource utilization using a higher client threshold, but the resource underallocation events increase drastically, as seen in the session with the client

TABLE 8
Average over and underallocation for Concurrent MMOGs

MMOG workload structure			Over-allocation	Under-allocation	$\|\Upsilon\| \geq 1\%$
A [%]	B [%]	C [%]	[%]	[%]	events
0	0	100	35.98	-0.12	240
5	5	90	36.89	-0.12	231
10	10	80	36.50	-0.12	220
25	25	50	35.80	-0.10	213
33	33	33	33.84	-0.09	200
0	100	0	33.24	-0.09	216
100	0	0	19.79	-0.03	75

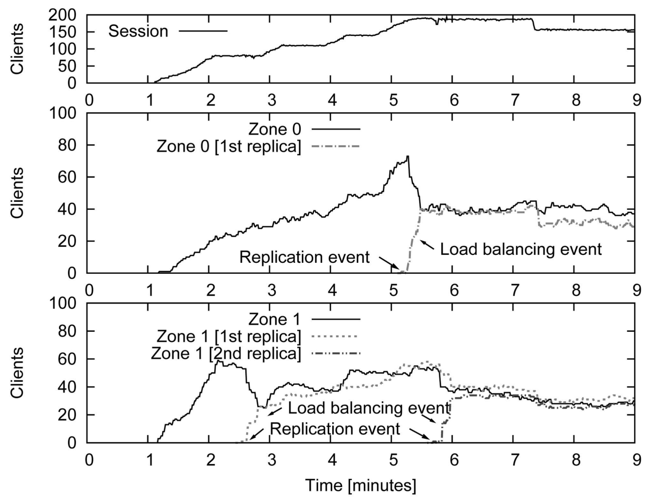

Fig. 15. Number of clients connected over time in a distributed session.

threshold of 50, which utilized only five machines and had an average underallocation of 8.69 percent. We conclude that our dynamic resource provisioning method performs better than the rudimentary client threshold-based method by providing a better service to the clients while offering a good resource utilization.

8 Related Work

We have already reviewed, in this paper, existing works related to our MMOG ecosystem, workload, and prediction models. We now turn our attention to the related work in the area of resource provisioning and identify three main directions from the resource provider's perspective.

8.1 Data Centers

The case when resources from one data center are shared between multiple applications with statistical performance guarantees has received much attention [9], [10], [14], [30], [31]. In all these approaches, the variables characterizing requests (such as "service time") can be expressed independently of the system state, for example, with a random variable whose behavior is well characterized by a well-known statistical distribution. However, an important component of the resource demands in the MMOG ecosystem is the interaction between players, which makes the resource demands of MMOGs very different from traditional web applications. To express the interaction between concurrent system users, in our model, a request is dependent on transient system state parameters, such as the number of active players and the location of players (see Section 4.1).

From this body of related work, much attention has been given to modeling single-tier web applications; with the notable difference in expressing interaction between concurrent users, these models are similar to ours. The MUSE system [14] allocates periodically a percentage of the resource capacity for each service class such that a target utilization is achieved at the data center level. An approach based on virtual machines, as opposed to physical resources, has also been explored [30]. Resource demand profiles for business applications are constructed for various durations (e.g., hour, day, week) and resources are allocated in advance to provide statistical performance guarantees [9]. A similar approach based on application profiles was proposed in [10]. Closest to our work, the benefit of provisioning resources from single data centers has been evaluated for databases and web services [8]. Our work differs from this approach in two significant aspects. First, MMOGs have a different load model and their load in particular depends also on the interaction between users. Second, we consider multiple data centers to handle the different load patterns in different geographical locations specific to MMOGs.

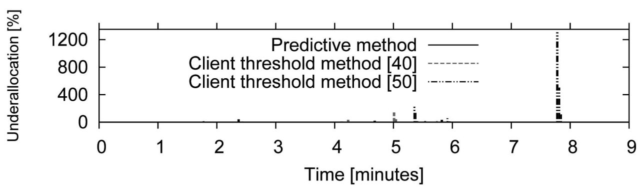

Fig. 16. Underallocation event comparison.

Recently, research has focused on multitier models of web applications [8], [31]. They use probability distributions to characterize resource demands, queuing theory to analyze the system, and proactive and reactive resource provisioning; the target environment is a single data center. In contrast, our approach seems less sophisticated, but it already fosters emerging behavior in a multidata center MMOG ecosystem, that is, the emergence of a complex system from the large number of simple choices and interactions; for example, we have shown in Section 6.5 evidence that hosters have incentives to offer MMOG-friendly hosting policies when the market is competitive.

8.2 Grid Computing

The problem of dynamically allocating geographically distributed resources to applications has been a popular topic in Grid computing research. Recent work investigates mechanisms for resource allocation across single and multicluster Grids [32]. They assess the performance of various resource allocation mechanisms for typical Grid workloads comprising batches of scientific and engineering jobs [33]. Unlike MMOGs, scientific Grid applications do not change their resource requirements at runtime. Moreover, the Grid resource allocation policies only allow for whole resources be allocated at a time, while our work also considers the subunitary allocation sizes specific to business data centers.

Closest to our work, the industrial game hosting platform Butterfly.net Grid (now renamed the Emergent Platform) [34] uses Grid technology to provide on-demand access to cluster resources. Their hosting policy only considers multiunitary resource bulks and long time bulks; as such, this platform fits well into our MMOG ecosystem as a typical large hoster.

8.3 Peer-to-Peer Computing

Peer-to-peer computing has emerged as a scalable and low-cost technology, and as a potential alternative to traditional on-demand resource provisioning. When employing peer-to-peer technology, the game operators make use of the resources of their clients instead of renting them from hosters. The NPSNET project [35] uses a peer-to-peer approach in which all the game computation is performed on client resources. The SimMud [36] project uses a similar approach to NPSNET, but also balances and optimizes the use of resources. However, three problems have prevented so far the adoption of peer-to-peer technology for MMOGs: the lack of appropriate business models, the widespread attempts of cheating, and the low availability of peers observed for other peer-to-peer systems (such as the Gnutella and the BitTorrent file sharing networks [37], [38]).

9 Conclusion and Future Work

We focused in this work on MMOGs as a new type of large-scale distributed simulation with a growing user base of tens to millions of players. To ensure that the user demand is satisfied at all times, game operators resort to static resource provisioning by building and maintaining computing platforms of up to 10,000 machines located on several continents for a single MMOG. In this paper, we proposed a more efficient alternative based on the dynamic resource provisioning and management of data center resources. Ours is the first thorough investigation of an MMOG ecosystem, that is, of a multi-MMOG, multidata center environment.

We showed in this work that the number and the type of interactions between players, and between players and the environment, are an important contributor to the game load. To address it, we have introduced a new MMOG model that focuses on the interaction count and type between game entities, shown that interaction leads to much more dynamic resource demands than previously believed, and proposed a novel prediction algorithm based on neural networks that is fast yet accurate. Our algorithm performed significantly better than the six-time predictors also investigated in this work. We have further investigated the performance of the resource provisioning and management of data center resources with a large variety of scenarios that focus both on MMOG-specific properties and data center hosting policies. Most importantly, we have shown that the static resource provisioning can be, on average, from five up to 10 times more inefficient than dynamic allocation under the same conditions, and that the game operators can penalize the data centers with unsuitable hosting policies, by not using their resources. Finally, we have designed and implemented our methods on top of the platform offered by the EU project edutain@grid, and presented an experiment showing the real-time resource provisioning for a real game prototype.

References

[1] C. Neumann, N. Prigent, M. Varvello, and K. Suh, "Challenges in Peer-to-Peer Gaming," *Computer Comm. Rev.*, vol. 37, no. 1, pp. 79-82, 2007.

[2] W.M. White, C. Koch, N. Gupta, J. Gehrke, and A.J. Demers, "Database Research Opportunities in Computer Games," *SIGMOD Record,* vol. 36, no. 3, pp. 7-13, 2007.

[3] R. Bartle, *Designing Virtual Worlds.* New Riders Games, 2003.

[4] A. Shaikh, S. Sahu, M.-C. Rosu, M. Shea, and D. Saha, "On Demand Platform for Online Games," *IBM Systems J.,* vol. 45, no. 1, pp. 7-20, 2006.

[5] T. Beigbeder, R. Coughlan, C. Lusher, J. Plunkett, E. Agu, and M. Claypool, "The Effects of Loss and Latency on User Performance in Unreal Tournament 2003," *Proc. Workshop Network and System Support for Games (NetGames),* pp. 144-151, 2004.

[6] B. Hack, M. Morhaime, J.-F. Grollemund, and N. Bradford, "Introduction to Vivendi Games," Presentation, http://www.vivendi.com/, June 2006.

[7] R.P. Doyle, J.S. Chase, O.M. Asad, W. Jin, and A. Vahdat, "Model-Based Resource Provisioning in a Web Service Utility," *Proc. USENIX Symp. Internet Technologies and Systems,* 2003.

[8] B. Urgaonkar, P.J. Shenoy, A. Chandra, and P. Goyal, "Dynamic Provisioning of Multi-Tier Internet Applications," *Proc. Int'l Conf. Autonomic Computing,* pp. 217-228, 2005.

[9] J. Rolia, X. Zhu, M.F. Arlitt, and A. Andrzejak, "Statistical Service Assurances for Applications in Utility Grid Environments," *Performance Evaluation,* vol. 58, nos. 2/3, pp. 319-339, 2004.

[10] A. Karve, T. Kimbrel, G. Pacifici, M. Spreitzer, M. Steinder, M. Sviridenko, and A.N. Tantawi, "Dynamic Placement for Clustered Web Applications," *Proc. Conf. World Wide Web (WWW),* pp. 595-604, 2006.

[11] Jagex, Ltd., "Runescape," MMOG, http://www.runescape.com/, Nov. 2007.

[12] K.-T. Chen, P. Huang, and C.-L. Lei, "Game Traffic Analysis: An MMORPG Perspective," *Computer Networks,* vol. 50, no. 16, pp. 3002-3023, 2006.

[13] T. Fahringer, C. Anthes, A. Arragon, A. Lipaj, J. Müller-Iden, C. Rawlings, R. Prodan, and M. Surridge, "The Edutain@grid Project," *Proc. Int'l Workshop Grid Economics and Business Models (GECON),* pp. 182-187, Aug. 2007.

[14] M. Aron, P. Druschel, and W. Zwaenepoel, "Cluster Reserves: A Mechanism for Resource Management in Cluster-Based Network Servers," *Proc. SIGMETRICS,* pp. 90-101, 2000.

[15] M. Ye and L. Cheng, "System-Performance Modeling for Massively Multiplayer Online Role-Playing Games," *IBM Systems J.,* vol. 45, no. 1, pp. 45-58, 2006.

[16] W. Cai, P. Xavier, S.J. Turner, and B.-S. Lee, "A Scalable Architecture for Supporting Interactive Games on the Internet," *Proc. 16th Workshop Parallel and Distributed Simulation,* pp. 60-67, 2002.

[17] J. Müller-Iden and S. Gorlatch, "Rokkatan: Scaling an RTS Game Design to the Massively Multiplayer Realm," *Computers in Entertainment,* vol. 4, no. 3, p. 11, 2006.

[18] A.R. Bharambe, J.R. Douceur, J.R. Lorch, T. Moscibroda, J. Pang, S. Seshan, and X. Zhuang, "Donnybrook: Enabling Large-Scale, High-Speed, Peer-to-Peer Games," *Proc. ACM SIGCOMM,* pp. 389-400, 2008.

[19] M. Claypool, "The Effect of Latency on User Performance in Real-Time Strategy Games," *Computer Networks,* vol. 49, no. 1, pp. 52-70, 2005.

[20] B.S. Woodcock, "An Analysis of MMOG Subscription Growth," Report, 21 ed., http://www.mmogchart.com, June 2006.

[21] W.-C. Feng, F. Chang, W.-C. Feng, and J. Walpole, "A Traffic Characterization of Popular On-Line Games," *IEEE/ACM Trans. Networking,* vol. 13, no. 3, pp. 488-500, June 2005.

[22] W.-C. Feng, D. Brandt, and D. Saha, "A Long-Term Study of a Popular MMORPG," *Proc. Workshop Network and System Support for Games (NetGames),* pp. 6-11, 2007.

[23] D. Pittman and C. Gauthier, "A Measurement Study of Virtual Populations in Massively Multiplayer Online Games," *Proc. Workshop Network and System Support for Games (NetGames),* pp. 25-30, 2007.

[24] D.A. Menascé, V. Almeida, R.H. Riedi, F. Ribeiro, R.C. Fonseca, and W. Meira, Jr., "A Hierarchical and Multiscale Approach to Analyze E-Business Workloads," *Performance Evaluation,* vol. 54, no. 1, pp. 33-57, 2003.

[25] G.E.P. Box, G.M. Jenkins, and G.C. Reinsel, *Time Series Analysis, Forecasting and Control.* Prentice Hall, 1994.

[26] C.M. Bishop, *Neural Networks for Pattern Recognition.* Oxford Univ. Press, 1996.

[27] I. GameData, "Counter Strike," http://www.counter-strike.com, 2010.

[28] V. Nae, R. Prodan, and T. Fahringer, "Neural Network-Based Load Prediction for Highly Dynamic Distributed Online Games," *Proc. Int'l Euro-Par Conf. Parallel Processing (Euro-Par),* 2008.

[29] F. Glinka, A. Ploss, J. Müller-Iden, and S. Gorlatch, "RTF: A Real-Time Framework for Developing Scalable Multiplayer Online Games," *Proc. Workshop Network and System Support for Games (NetGames),* 2007.

[30] P. Padala, K.G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, and K. Salem, "Adaptive Control of Virtualized Resources in Utility Computing Environments," *Proc. European Conf. Computer Systems (EuroSys),* pp. 289-302, 2007.

[31] B. Urgaonkar, P.J. Shenoy, A. Chandra, P. Goyal, and T. Wood, "Agile Dynamic Provisioning of Multi-Tier Internet Applications," *ACM Trans. Autonomous and Adaptive Systems (TAAS),* vol. 3, no. 1, 2008.

[32] M. Siddiqui, A. Villazón, and T. Fahringer, "Grid Allocation and Reservation—Grid Capacity Planning with Negotiation-Based Advance Reservation for Optimized Qos," *Proc. ACM/IEEE Conf. High Performance Networking and Computing (SuperComputing)*, p. 103, 2006.

[33] A. Iosup, C. Dumitrescu, D.H. Epema, H. Li, and L. Wolters, "How Are Real Grids Used? The Analysis of Four Grid Traces and Its Implications," *Proc. Int'l Conf. Grid Computing (GRID)*, pp. 262-270, 2006.

[34] Gamebryo, "Butterfly Grid/Emergent Platform," http://www.emergent.net/, Aug. 2008.

[35] M.R. Macedonia, D.P. Brutzman, M.J. Zyda, D.R. Pratt, P.T. Barham, J. Falby, and J. Locke, "NSPNET: A Multiplayer 3D Virtual Environment over the Internet," *Proc. Symp. Interactive 3D Graphics (SI3D '95)*, pp. 93-94, 1995.

[36] B. Knutsson, H. Lu, W. Xu, and B. Hopkins, "Peer-to-Peer Support for Massively Multiplayer Games," *Proc. INFOCOM*, pp. 96-107, 2004.

[37] M. Ripeanu, A. Iamnitchi, and I.T. Foster, "Mapping the Gnutella Network," *IEEE Internet Computing*, vol. 6, no. 1, pp. 50-57, Jan./Feb. 2002.

[38] J. Pouwelse, P. garbacki, D. Epema, and H. Sips, "The Bittorrent P2P File-Sharing System: Measurements and Analysis," *Proc. Int'l Workshop Peer-to-Peer Systems (IPTPS)*, 2005.

Vlad Nae received the diploma engineer degree from the Politechnica University of Bucharest, Romania, in 2006. He is currently a research assistant at the Institute of Computer Science, University of Innsbruck. His research interests include resource management and monitoring in distributed systems, particularly in the area of highly interactive applications such as massively multiplayer online games. He is the author of 12 publications including two journal articles.

Alexandru Iosup received the PhD degree in computer science from TU Delft in 2009. He is currently an assistant professor in the Parallel and Distributed Systems Group, Delft University of Technology (TU, Delft), The Netherlands. His research interests include parallel distributed computing, from applications to system design, development, and evaluation. His research was rewarded with several awards and distinctions. He received the IEEE P2P Best Paper Award for improvements to BitTorrent. He is a member of the IEEE.

Radu Prodan received the PhD degree from the Vienna University of Technology in 2004 and the Habiliatation degree from the University of Innsbruck. He is an assistant professor at the Institute of Computer Science, University of Innsbruck. His research interests include programming models, compiler technology, performance analysis and scheduling for parallel and distributed systems. He has authored more than 80 journal and conference publications, one book, and received one IEEE Best Paper Award. He is a member of the IEEE.