European Research Network on Foundations, Software Infrastructures and Applications for large scale distributed, GRID and Peer-to-Peer Technologies
A Network of Excellence funded by the European Commission

A Grid Environment for Real-Time Multiplayer Online Games

Sergei Gorlatch, Frank Glinka, Alexander Ploß and Jens Müller-Iden
{gorlatch,glinkaf,a.ploss,jmueller}@uni-muenster.de
Institute of Computer Science
University of Münster, Germany

Radu Prodan, Vlad Nae and, Thomas Fahringer
{radu,vlad.nae,thomas.fahringer}@dps.uibk.ac.at
University of Innsbruck
Austria

CoreGRID Technical Report
Number TR-0133
May 15, 2008

Institute on Institute on Programming Model
Institute on Grid Information, Resource
and Workflow Monitoring Services
Institute on Resource Management and Scheduling

CoreGRID - Network of Excellence
URL: http://www.coregrid.net

CoreGRID is a Network of Excellence funded by the European Commission under the Sixth Framework Programme

Project no. FP6-004265

A Grid Environment for Real-Time Multiplayer Online Games

Sergei Gorlatch, Frank Glinka, Alexander Ploß and Jens Müller-Iden
{gorlatch,glinkaf,a.ploss,jmueller}@uni-muenster.de
Institute of Computer Science
University of Münster, Germany

Radu Prodan, Vlad Nae and, Thomas Fahringer
{radu,vlad.nae,thomas.fahringer}@dps.uibk.ac.at
University of Innsbruck
Austria

CoreGRID TR-0133
May 15, 2008

Abstract

We present joint work between the University of Innsbruck and University of Münster on targeting online games as a novel class of Grid applications, whose user community (general public) is much broader than of contemporary scientific Grids. Online games are a new, large, generic class of applications not yet studied by the Grid community, with the following distinctive features in comparison to traditional parameter studies or scientific workflows: large number of concurrent users connecting to a single application instance, frequent real-time user interactions, negotiation and enforcement of precise Quality of Service (QoS) parameters, adaptivity to changing loads and levels of user interaction, and competition-oriented interaction between users, other actors, and services.

We develop a novel multi-layer, service-oriented architecture for executing online games in a distributed Grid infrastructure. Firstly, scheduling and runtime steering services assist the users in transparently connecting to game sessions while maintaining certain levels of QoS. Secondly, resource allocation, monitoring, and capacity planning services allow efficient resource management that removes the cost and scalability barriers in game hosting. Finally, a Real-Time Framework (RTF) provides the fundamental technology for scaling game sessions to an increased number and density of users through real-time protocols and a variety of parallelization and distribution techniques.

This research work is carried out under the FP6 Network of Excellence CoreGRID funded by the European Commission (Contract IST-2002-004265).

1 Motivation

Within the last ten years, *online gaming* has become a dominant component in the video game market. Every game genre has been impacted and even redefined by the multiplayer network feature, completely transforming the players' experience. This trend is developing further as all the hardware devices released on the market today have built-in network support and thus allow the game developer to rely on network availability. In contrast to existing interactive applications like those provided by Amazon and eBay exclusively oriented towards business transactions, online games have severe realtime and scalability requirements (like the minimum state update rate per second to all clients) which makes them rather unique and yet to be studied by the Grid community.

The online game market is nowadays dominated by three classes of online games: (1) *Massively Multiplayer Online Role Playing Games (MMORPGs)* (e.g. Anarchy Online, Silkroad Online, World of Warcraft) are slow-paced games with few thousands of players sharing one game session in a huge game world map; (2) *First Person Shooter (FPS)* (e.g. Counter Strike Source, Battlefield 2142) are fast-paced action games that only scale to a few players (maximum 64) due to the extremely frequent action processing and communication (up to 60 Hz) required for updating the state of each player in order to experience a realistic environment; (3) *Real-Time Strategy (RTS)* games (e.g. Command & Conquer, Starcraft) are comparatively fast-paced games that typically contain a high number of ongoing interactions in the game world. Users usually have direct control over a huge amount of in-game objects, e.g. military units, and can trigger actions on them that can cause interactions with a lot of other in-game objects and consume a lot of bandwidth and processing time. In this paper we present a novel multi-layer service-oriented architecture as a platform that eliminates the following three main barriers which currently hinder the scalability of online games.

Firstly, to improve the scalability of MMORPG game sessions, game developers and hosting companies currently divide the game world in zones (often representing different realms) which are hosted on different machines. The zones are typically predefined and managed manually by the Hosters. The transition of a client between two zones is in general not a seamless action and requires an important amount of waiting time. This solution has therefore the same limitations as a monotonic environment, since the number of players within one zone is still limited and cannot be extended. To address this limitation, we propose a *Real-Time Framework (RTF)* providing the fundamental technology for dynamic on-the-fly scaling of game sessions to the user demands through real-time protocols and a variety of novel parallelization and distribution techniques. RTF is not limited to MMORPG games, but addresses the more dynamic and harder to satisfy FPS and RTF games too.

Secondly, the only significant attempts in terms of hosting to apply Grid technologies to online games are the Butterfly.net [3] and BigWorld [1] environments. Both are commercial Grid systems for MMORPGs which optimise resource utilisation within a single data centre typically residing within the same local area network. However, because of this restriction, this solution is subject to the same limitations as with any data centre as there is no way to add substantially more resources if those assigned become insufficient. This limitation has also negative economic impacts by preventing any but the largest hosting centres from joining the market which will dramatically increase prices because those centres must be capable of handling peaks in demand, even if the resources are not needed for much of the time. We propose in this paper an architecture that uses the potential of the global Grid to provide on-demand access to a potentially unbounded amount of cheap compute resources connected to the Internet through three advanced middleware services: *resource allocation*, *monitoring*, and *capacity planning*.

Thirdly, current online game infrastructures are usually based on the best-effort Internet protocol with no QoS support, which is needed for a smooth and realistic experience. We provide *scheduling* and *runtime steering* services to assist the users in transparently connecting to game sessions while enforcing certain levels of QoS for the entire duration of the interaction through proactive load balancing and session steering decisions.

We present our architecture design in Section 2, followed by implementation and experimental results in Section 3. We conclude our paper with an outlook into the future collaboration work in Section 4.

2 Architecture

We designed a distributed service-oriented architecture depicted in Figure 1 to support transparent and scalable access of an increased number of users (compared to current state of the art) to existing online gaming applications. The architecture is based on the interaction of four main actors. (1) *End-user* is the game player that accesses the online game sessions through graphical clients, typically purchased as a DVD; (2) *Scheduler* is an intermediary that negotiates on behalf of the end-user appropriate game sessions based on the user-centric QoS requirements (e.g. connection

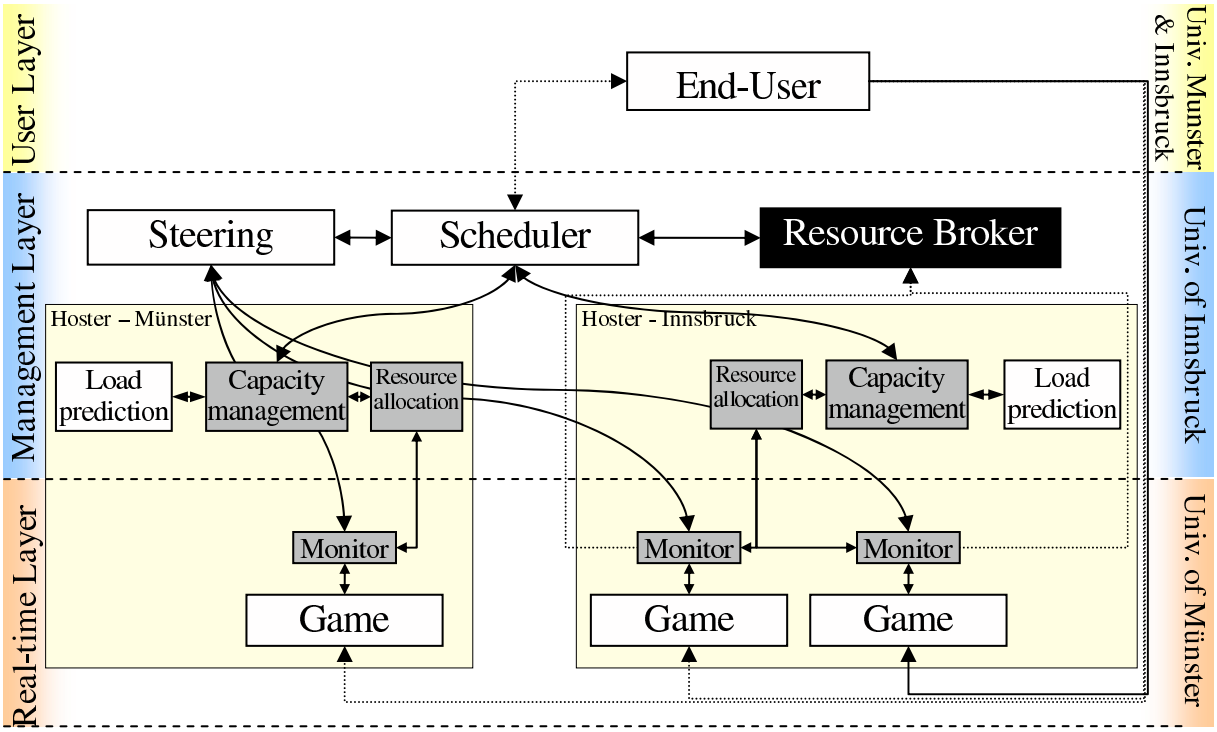

Figure 1: The overall Grid architecture.

latency, game genre, friends, expertise); (3) *Hoster* is an organisation (e.g. Universities of Innsbruck and Münster in the current project) that provides some computational and network infrastructure for running game servers; (4) *Resource broker* provides a mechanism for application Schedulers and Hosters (and possibly other actors) to find each other in a large-scale Grid environment and negotiate QoS relationships.

2.1 User layer

The user layer consists of the end-user game application which should be simple and easy to install and manage. One important requirement is that the Grid middleware must not add additional complexity to existing game installation procedures based on a DVD purchase and several mouse clicks. Rather, we aim at simplifying today's process of initiating and connecting to online game sessions, such that the user no longer needs to memorise and edit IP server addresses or manually edit custom configuration files.

2.2 Scheduling layer

The scheduling layer developed by the University of Innsbruck is where the different interactions between the actors are supported. Apart from an off-the-shelf Resource Broker [4] that we integrated into the environment for resource discovery, we designed two novel services as part of this layer.

2.2.1 Scheduling service

The Scheduler receives from the user specific QoS requirements which can be performance-related (e.g. maximum latency, minimum bandwidth, minimum throughput) or game-specific (e.g. game genre, minimum number of players, difficulty, tournament planning). Our Scheduler goes beyond the current state-of-the-art practice in scientific computing (i.e. centralised best-effort optimisation of single application-specific metrics), by negotiating with several distributed Hosters multiple and often contradictory (from each actor's perspective) QoS requirements (e.g. execution time versus computation cost) to be maintained during execution.

The mapping of players to game servers, as well as the allocation of Hoster resources to game servers, takes place as a distributed negotiation between the Scheduler and Hosters, each of them trying to optimise its own specific metrics expressing individual interests. While the Scheduler purely negotiates in terms of users-centric QoS parameters, the Hosters try to selfishly optimise metrics related to their own and often contradicting interests such as maximising resource utilisation or income. A centralised Scheduler, as typically approached in the scientific Grid community, is therefore not a feasible solution in such a distributed non-collaborative environment. Rather, we approach the problem using game theory [8], where a number of parties having different objectives and utility functions negotiate using cooperative or non-cooperative techniques, possible enhanced with auctioning or bargaining models. It is proven that such theories converge to a Nash equilibrium that represents a balance between risks and rewards for all participating parties. The result of the negotiation process is a performance contract that the Scheduler offers to the end-user and which does not necessarily match the original QoS request. The user has the option to accept the contract and connect to the proposed session, or reject it.

2.2.2 Runtime steering service

There may occur factors during the execution of a game session which affect the performance, such that the negotiated contracts are difficult or impossible to be further maintained. Typical perturbing factors include external load on unreliable Grid resources, or overloaded servers due to an unexpected concentration of users in certain "hot spots". The steering service interacts at runtime with the monitoring service of each Hoster in the management layer for preserving the negotiated QoS parameters for the entire duration of the game session. Following an event-action paradigm, a violation of a QoS parameter triggers appropriate adaptive steering or rescheduling actions using the API provided at the real-time layer.

The QoS parameters of online games to be enforced may be of two kinds:

(1) *Client-related QoS parameters* include requirements between the local game client and the remote server such as latency and bandwidth constraints collected by special sensors embedded in the client application (hidden to the end-user) that monitor the traffic on the real-time connection links. Upon violation of these requirements, users will be transparently migrated to new game servers (of the same distributed session) that maintain the original QoS parameters negotiated by the Scheduler.

(2) *Game session-related QoS parameters* include application-specific requirements such as efficient use of Grid resources, maximum load, or proper load balancing. Typical game session-related steering actions are: starting a new game server (potentially at a different Hoster site) for distributing the user load of an overloaded session; load balancing by migrating users between game servers within one or across several Hosters; switching-off underutilised game servers (after migrating the users) for improved resource utilisation; or cloning game servers upon critical QoS values to hide user migration latencies.

All of these actions must be performed transparently from the user's point of view who should not notice any change or experience minimum delay in the execution environment. To meet this challenge, we consider defining performance contracts based on fuzzy logic [11] (rather than based on exact boolean variables) that allows one to linguistically state smooth transitions between areas of acceptance and violation. The user can express the contracts by defining a membership function with the following signature: $\mu : \mathbb{R} \rightarrow [0, 1]$, where $\mathbb{R}$ denotes the set of real numbers (representing performance values such as current user load). A very simple membership function could be a step function with three linguistic values: fulfil (indicating proper execution), critical (indicating that the contract is in danger and needs steering), and violate (when the contact has already been broken). Fuzzy logic is a powerful technique that will provide flexibility in performing proactive steering actions before the actual contracts are violated, for example through anticipated migration or cloning when the load of the game sessions enters a critical stage.

2.3 Resource management layer

The resource management layer, mostly developed by the University of Innsbruck (apart from monitoring developed in Münster), focuses on the development of generic Grid management services within each Hoster, while dealing with the challenges raised by the main requirements of online games: scalability to thousands of online user connections, optimised allocation of resources, and monitoring and steering to maintain the QoS parameters in dynamic environments like the Grid. The traditional way of allocating resources in a Grid environment employs opportunistic matchmaking models based on runtime resource requests and offers [9], which are known to converge to local minima

as we demonstrated in previous work on scientific workflows [4]. We improve on the state of the art by designing a management layer consisting of three services described in the following.

2.3.1 Resource allocation

Typically, each Hoster in the management layer owns one Resource Allocation service responsible for allocating local resources to a large number of connecting clients. The Resource Allocation service receives from the Scheduler connection requests formulated in terms of QoS requirements such as minimum latency, maximum bandwidth or maximum load, and returns either a positive answer if it can accommodate the client, or a negative answer otherwise.

It is important to notice that, for online games, the resource allocation problem is significantly different than in scientific computing. Scientific applications are typically owned by single users and consist of a set of computationally intensive jobs to be mapped to individual processors. Online games, in contrast, are characterised by a large number of users that share the same application instance and interact within the same or across different game servers. The atomic resource allocation units for users are, therefore, no longer coarse-grain processors, but rather fine-grained threads and memory objects, which are obviously harder to address and more sensitive to external perturbations.

2.3.2 Monitoring

High-level monitoring services implemented by the University of Münster are aiming at observing the QoS parameters previously negotiated by the Hoster as performance contracts which must be preserved throughout the user's participation in the game session. Several monitoring parameters are summarized in particular profiles. To support a wide range of game types, we distinguish between two different types of profiles: basic profiles common to all games, and the custom profiles which allow the compilation of customised parameter sets adapted to the requirements of any particular game.

The basic profiles, already developed and implemented by the University of Münster, support monitoring of low-level QoS parameters, as well as of game-related metrics crucial for guaranteeing an adequate game play experience to the users. The *system profile* and *network profile* contain generic low-level system parameters which do not depend on the game inner state and represent an assessment of the game server resource use. The *entity model profile*, *client data profile*, and *game world profile* monitor information related to internal game mechanisms like entity positions, messages sent or received, or end-user activity information such as idle, spectator, or active.

2.3.3 Capacity Planning

The load of a game session depends heavily on internal events such as the number of entities that find themselves in each other's area of interest and interact altering each other's stare. Alongside internal events, there may also occur external events such as the user fluctuation over the day or week with peak hours in the early evening [5]. Hence, it becomes crucial for a Hoster to anticipate the future game load which brings the following benefits: (1) enhanced confidence in guaranteeing the fulfilment of global QoS parameters for the entire distributed game session (like smooth response time or proper load balancing) through ahead-planning of resource allocation; (2) anticipation of critical game stages by predicting potential contract violations and performing proactive steering actions before the contracts are actually broken; (3) opportunities for more aggressive resource allocation that accommodates a larger number of users.

Projecting future load in highly dynamic Grid environments of game players needs to take into account a multitude of metrics such as processor, memory, and network latencies and load. All these metrics can be deterministically expressed as a (game-specific) function of the human factor, which is the hardest and most unpredictable parameter. We therefore reduce the prediction problem to the task of estimating the entity distribution in the game world at several equidistant time intervals. Our solution goes towards neural networks due to a number of reasons which make them appropriate to being applied on online games: they adapt to a wide range of time series, they offer better results than other simple methods, and they are sufficiently fast compared to other more sophisticated statistical analysis.

We employ a neural network for load prediction that has a signal expander as input to ensure a non-linear expansion of the input space fed to the hidden neuron layers (see Figure 2). The input fed into the fuzzy layer consists of the positions of the players in the game world at several successive equidistant time intervals (Δt). As output signal, network provides a similar space representing a prediction of the players' layout in the next time interval ($t + \Delta t$). Because the input is expanded, the expected output will not be a precise estimation of each player's position, but a world

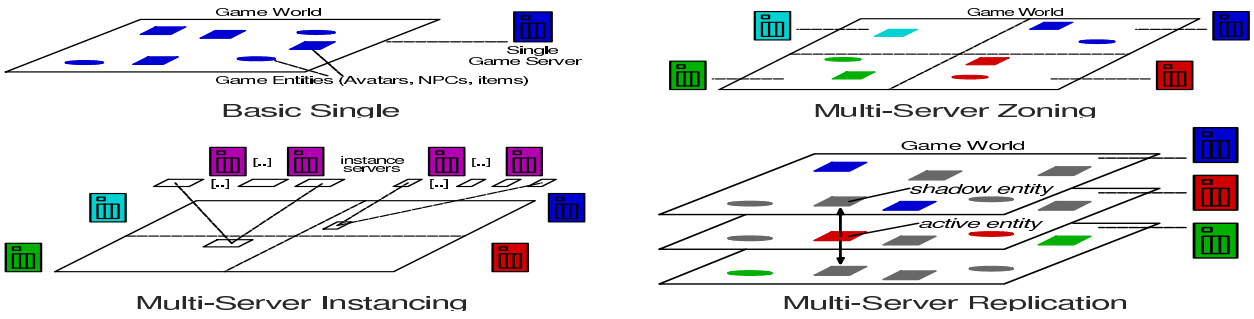

Figure 3: Game distribution strategies.

of subarea estimations. Each zone of the map is analysed by dividing it into subareas and providing a local prediction. An interesting aspect is that the edges of the subareas are overlapped to hide latencies upon player transitions.

Two offline phases are required for utilising the neural network-based prediction. 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 training phase uses most of the previously collected samples as training sets, and the remaining as test sets. The *training phase* runs for a number of eras, until a convergence criterion is fulfilled. A training era has 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.

These two stages are performed only once per world and game type since they are the main game characteristics that determine the player behaviour. Once successfully trained, the network can be serialised and reused at a later time, when the same world and game type combination is played.

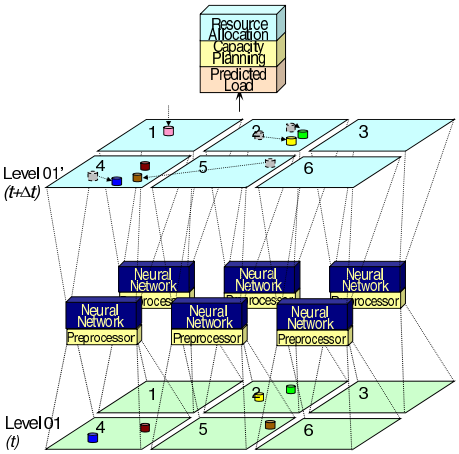

Figure 2: Neural network capacity planning.

2.4 Real-time layer

As part of the real-time layer, the University of Münster is developing the Real-Time Framework (RTF) [6] that hides the Grid management infrastructure for the game servers. For the in-game communication, we designed a custom and highly optimised communication protocol independent from the XML-based protocols within the management infrastructure. To improve the game scalability to a high number and density of users, we distribute game sessions based on several distribution strategies illustrated in Figure 3.

The concept of *zoning* [2] is based on the distribution of the game world into several geographical zones, each zone being assigned to one game server. Players are able to move freely between zones with little overhead such that no interruptions in the game play are experienced. *Instancing* uses multiple, independently processed copies of highly frequented subareas of the geographical world, each copy being processed by one separate server. If a user moves into one such highly frequented subarea, he is assigned to one of the available copies. *Replication* is a novel technique developed in Münster [7] which assigns multiple servers to a single game world zone with a high load. The responsibility of computing the entities' states in that zone is divided equally among the assigned processors. Each processor has a local copy of the states of all entities in the game world zone which, after the assigned computations are done, is synchronised with the other servers.

The main novel features of the RTF are as follows: (1) Highly optimised and dynamic real-time communication links which adapt to changes in the dynamic Grid environment and allow the RTF, e.g. to redirect the communication to new servers underneath without a required reaction from the game developer; (2) Hidden background preparations like speculative connection establishments, which allow the runtime transfer and redistribution of parts of a game onto additional Grid resources without noticeable interruptions for the participating users; (3) An interface for the game

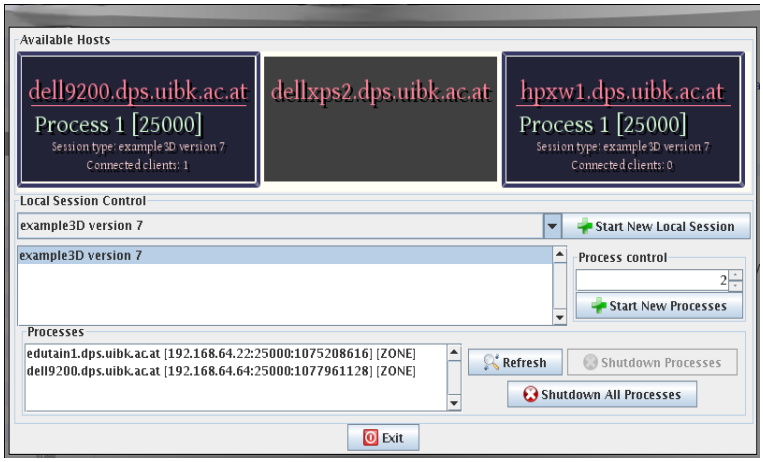

Figure 4: Management portal snapshot

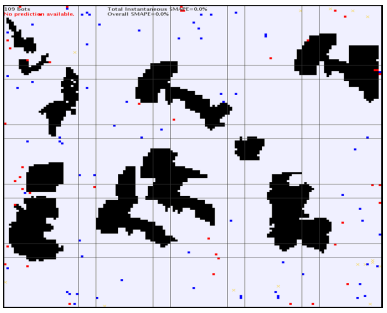

Figure 5: Game simulator snapshot

developer that abstracts the game processing from the location of the participating resources. This is the technical basis that allows the management layer to dynamically reassign the game processing to the available resources; (4) Monitoring data gathering in the background about, e.g., the number of exchanged events, number of in-game objects on a server, and connection latencies. This data is used by the management layer for capacity planing and steering services.

3 Implementation Issues and Experiments

We are developing the scheduling and resource management layers as a set of WSRF-based services deployed in the Austrian Grid environment that aggregates a large collection of heterogeneous parallel computers distributed across several major cities of Austria. We provide a *management portal* developed by the University of Innsbruck and depicted in Figure 4, which offers means of visualising the available machines in the Grid, the game sessions running at each Hoster site, and important monitoring metrics such as user load on each game server. The portal also offers key management features such as start-up and shut-down of sessions for games using RTF as well as stand-alone proprietary game server.

University of Münster implemented the RTF as a C++ library, since C++ is the dominating language in game development mainly due to performance reasons. The C++ library hides the underlying complexity of the distributed Grid from the game developers and allows them to realise their games without a major shift compared to contemporary client-single server development. Additionally, we realised a Java interface to the C++ library for exporting monitoring and management capabilities to the upper layers.

In order to generate load patterns suitable for testing and validating our prediction method, we developed a distributed FPS game simulator supporting the zoning and inter-zone migrations of the entities (see Figure 5). The simulator integrates the real-time layer implemented and generates dynamic load by simulating interaction hot-spots between large numbers of entities managed by one server. In order to simulate an as realistic environment as possible, the entities are driven by several Artificial Intelligence (AI) profiles which determine their behaviour during a game session: *aggressive* determines the entity to seek and interact with opponents; *team player* causes the entity to seek or form a group with its teammates; *scout* leads the entity into uncharted zones of the game-world (not guaranteeing any interaction); and *camper* simulates a well-known FPS game tactic to hide and wait for the enemy. To better emulate player behaviour in a realistic session, each entity is able to dynamically switch between all AI profiles during the simulation (e.g. from aggressive to camper once wounded) with a certain degree of randomness. We configured the entity speed and interactions to a highly dynamic value making the prediction in our belief harder than in existing fast-paced FPS games.

To validate our neural network prediction in the first instance, we generated using our game simulator eight different trace data sets with different characteristics by running 17 hours of simulations for each set and sampling the game state at every two minutes. The first four data traces simulate different scenarios of a highly dynamic FPS game,

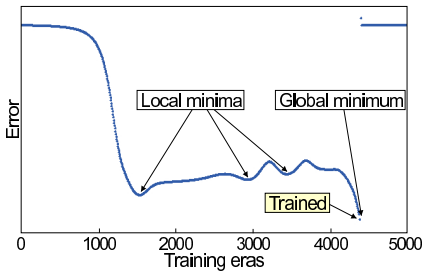

Figure 6: Error trend during neural network training

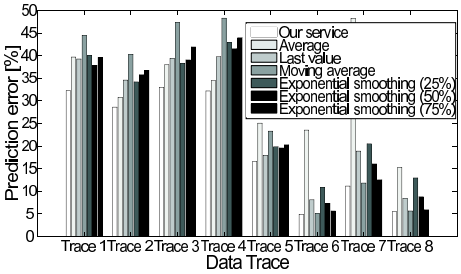

Figure 7: Comparison of neural network prediction against other methods

while the other four are characteristic to different MMORPG sessions. We use this trace data for training the neural network as presented in Section 2.3.3 until the process converges to a global minimum (see Figure 6). The majority of the traces are used as training sets and the rest as test sets.

To quantify the quality of our prediction, we compared the prediction error of the neural network against other fast prediction methods such as moving average, last value, average, exponential smoothing, which are known to be among the most effective in large and dynamic environments as the Grid [10]. Figure 7 illustrates that, apart from producing better results, our neural network prediction has the main quality of being adaptive to time series with different characteristics which related methods fail to achieve. The drawback of these conventional methods is that it is not universally clear during a game play which of them should be applied as the real-time prediction method for the next time step. Our prediction successfully manages to adapt to all these heterogeneous types of signals and always delivers good results, especially in the case of highly dynamic FPS action games (i.e. first four data traces).

4 Conclusions

We presented a joint work between the University of Innsbruck and the University of Münster towards a transparent, four-layer Grid environment designed to improve the scalability and QoS provisioning in online games, as a novel class of Grid applications appealing to a general public.

At the bottom layer, the RTF library provides the core technology for portable development and scalability of games through the combination of zoning and replication techniques. On top of it, we designed middleware services comprising resource allocation, capacity planning, scheduling, and runtime steering for automatic on-the-fly management, scaling, and provisioning of QoS parameters required for a smooth and efficient execution on the underlying Grid resources. We integrated the RTF library into a game simulator capable of delivering realistic user loads required for validating our methods, and we showed experimental results of using a neural network for predicting user behaviour in online games which performs better than traditional last value, averaging, or exponential smoothing methods.

References

[1] BigWorld. BigWorld Technology, http://www.bigworldtech.com.

[2] W. Cai, P. Xavier, S. J. Turner, and B.-S. Lee. A scalable architecture for supporting interactive games on the internet. In *16th workshop on parallel and distributed simulation*, pages 60–67, Washington, DC, USA, 2002. IEEE Computer Society.

[3] Emergent Game Technologies. Butterfly.net, http://www.emergentgametech.com.

[4] T. Fahringer, R. Prodan et al. ASKALON: a Grid application development and computing environment. In *6th Int. Workshop on Grid Computing*. IEEE Computer Society, 2005.

[5] W.-C. Feng, D. Brandt, and D. Saha. A long-term study of a popular MMORPG. In *NetGames'07*, ACM Press., 2007.

[6] F. Glinka, A. Ploss, J. Müller-Iden, and S. Gorlatch. RTF: A Real-Time Framework for Developing Scalable Multiplayer Online Games. In *NetGames'07*, ACM Press., 2007.

[7] J. Müller-Iden and S. Gorlatch. Rokkatan: Scaling an RTS game design to the massively multiplayer realm. *Computers in Entertainment*, 4(3):11, 2006.

[8] R. B. Myerson. *Game Theory: Analysis of Conflict*. Harvard University Press, 1997.

[9] R. Raman, M. Livny, and M. H. Solomon. Matchmaking: An extensible framework for distributed resource management. *Cluster Computing*, 2(2):129–138, 1999.

[10] R. Wolski. Experiences with predicting resource performance on-line in computational grid settings. *ACM SIGMETRICS Performance Evaluation Review*, 30(4):41–49, 2003.

[11] L. A. Zadeh. Fuzzy logic. 21(8), 1988.