Implementation of a Service Platform for Online Games

Anees Shaikh, Sambit Sahu, Marcel Rosu, Michael Shea, and Debanjan Saha
Network Software and Services
IBM T.J. Watson Research Center
Hawthorne, NY 10532

aashaikh@watson.ibm.com, {sambits, rosu, mshea, dsaha}@us.ibm.com

ABSTRACT

Large-scale multiplayer online games require considerable investment in hosting infrastructures. However, the difficulty of predicting the success of a new title makes investing in dedicated server and network resources very risky. A shared infrastructure based on utility computing models to support multiple games offers an attractive option for game providers whose core competency is not in managing large server deployments.

In this paper we describe a prototype implementation of a shared, on demand service platform for online games. The platform builds on open standards and off-the-shelf software developed to support utility computing offerings for Web-based business applications. We describe our early experience with identifying appropriate performance metrics for provisioning game servers and with implementing the platform components that we consider essential for its acceptance.

Categories and Subject Descriptors: C.4 [Computer Systems Organization]: Performance of Systems; C.3 [Computer Systems Organization]: Special-Purpose and Application-Based Systems

General Terms: Design, Experimentation, Management

Keywords: Online games, game hosting, on demand computing

1. INTRODUCTION

Launching large-scale multiplayer online games requires considerable investment and effort to purchase and manage the computing and network infrastructure necessary to deliver a consistently satisfying gaming experience. The traditional approach taken by most publishers and game providers is to install a dedicated infrastructure for each game, resulting in high risk and expense, and potentially poor resource utilization. As the number of online games available to players grows, it is increasingly difficult to predict the success of a title at launch, or its popularity as it ages. Hence, the prospect of a high-cost hosting infrastructure, coupled with the possibility of a game's failure to meet expectations, is not attractive. Moreover, server and network management is typically well outside the core competency of game providers, namely designing games and features to attract and retain players.

This problem has recently gained attention in the context of business applications, where the issue of infrastructure cost strongly motivates new models for utility computing offerings (variously termed "on demand" [4], "utility data center" [6], or "just in time computing" [11]). These models provide the flexibility to scale an application or service in response to user demands by rapidly adding or removing resources (e.g., servers, storage, databases, network bandwidth, etc.) from a pool that may be shared across multiple applications or customers. With an on demand infrastructure, online game providers can similarly benefit by reducing their initial investment, while maintaining the ability to scale rapidly to accommodate players and add new services. For example, an on demand infrastructure based on standard Grid technology [3] was proposed for hosting online games in [10].

In this paper, we describe our work-in-progress to realize some of the major components of an on demand service platform for games using a combination of off-the-shelf commercial and open-source software. Our implementation is based on currently available provisioning software designed for managing Web-based business applications. We adapt this system to monitor game and system performance and automatically provision server resources in response to changing conditions. In addition, to make the platform easier to use, we have implemented a number of reusable auxiliary services that relieve publishers of the task of reimplementing commonly used functions for each game. We demonstrate the feasibility of the platform by provisioning multiple instances of a popular action game.

In addition, we describe in some detail our experience with identifying appropriate performance metrics for provisioning. Our initial experiments show that the best choice of performance metric is highly dependent on game session dynamics. Though each game may require a different set of metrics, we design the platform to localize game-specific information as much as possible. We also offer some initial discussion of the challenges in adapting off-the-shelf software designed for managing infrastructure for traditional Web-based business applications for use with networked games.

In Section 2 we briefly describe the service platform architecture, with a focus on the provisioning system and auxiliary services. The details of the implementation and testbed are discussed in Section 3. An illustration of the effectiveness of various types of performance metrics is shown in Section 4. Section 5 mentions some additional issues that are not addressed by the current implementation, and we summarize our work in Section 6.

2. PLATFORM ARCHITECTURE

In this section we discuss the high-level objectives of the service platform design, and follow with a description of our current prototype architecture.

SIGCOMM'04 Workshops, Aug. 30–Sept. 3, 2004, Portland, Oregon, USA.

2.1 Design objectives

The service platform design is guided by several principles. First, although we wish to relieve game providers from managing the system infrastructure, the platform and associated services should be minimally intrusive to the game applications. The challenge is to design the platform to be general enough to meet the basic requirements of many different types of games (and thus allow it to be shared), yet still allow it to be tailored to some extent to the needs of an individual game publisher. A second principle that follows from this is to provide the features in a modular fashion, allowing game publishers to harness those functions that address their needs and provide the most value. Finally, in order to ensure flexibility and extensibility of the platform, we leverage open standards and open source tools wherever possible.

These objectives represent a somewhat idealized view, and our current prototype does not fully meet all of them. However, as work-in-progress, the implementation represents an initial step toward realizing a service platform that reaches these goals.

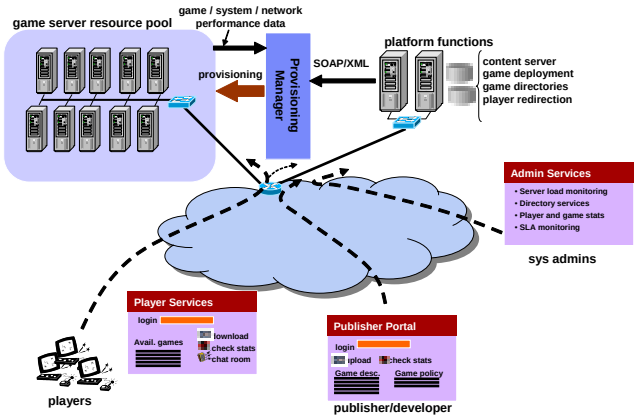


Figure 1: Prototype platform architecture

2.2 Prototype architecture

Figure 1 shows the platform architecture of our prototype, including players, game publishers, system administrators, and the platform itself. The game servers are collected into a resource pool which is shared across multiple game applications from multiple publishers. These servers may also perform auxiliary game functions such as login management, lobby services, or matchmaking on behalf of a particular game. The game servers are managed and automatically provisioned by the provisioning manager (PM), whose main function is to collect performance and availability metrics from the game servers and network infrastructure and respond to shortages or overallocations by adding or removing server and network resources. Notice that the PM must collect information specific to a game application in addition to system statistics about the server platform, such as CPU utilization, memory usage, and bandwidth consumption. The PM also implements a set of data and performance models which describe the information available from each game, along with its implication on game performance.

In addition to managing and provisioning game servers, the platform also provides a number of services and interfaces for platform users. These functions are performed by a separate group of servers which handle interactions with the platform, aside from actual gameplay. When a publisher wishes to deploy a new game, for example, the request is sent to the platform servers which process it and contact the PM to provision servers as necessary. Other functions supported on the platform servers include directories of games and players, support for redirecting players to the appropriate server to play the game, and meta-data for the content distribution service. These services and their current implementation are described in more detail in Section 3.3.

Players access the game servers without any modification, directly via the game client code located on their machines. In addition, they can access player services on the platform using the Web-based player portal. The game client software must also have access to the content distribution service to obtain patches and new game content. Notice that since the servers running a particular game will change over time, the players must access the redirection service upon initial connection to the game. Game publishers and developers interact with the platform primarily through their own portal, whose backend is implemented on the platform servers. Requests to deploy a new game with specific policies (e.g., minimum server requirements) and queries for information about running games are sent by the portal to the platform servers. Similarly, system administrators interact with the platform servers to get information about the operating conditions on the platform. Requests from each of the portals may also trigger commands to the PM server to query or provision game servers.

3. PLATFORM IMPLEMENTATION

The major components of our prototype implementation are the provisioning manager, server resource pool, platform server, and workload generators. For the experiments described in this paper, we use a simplified testbed with two game servers (see Figure 2). One server in the resource pool is designated as "active"; the other server is assigned to the free pool of "idle" servers. In general, the platform supports a large number of servers, organized in pools consisting of similar machines. We generate synthetic player traffic using remote bots (described further below) from a small number of workload machines. In the experiments described in this paper, the client traffic is generated on the LAN; we are also introducing network delay elements in the testbed (e.g., NistNet) to recreate wide-area conditions. Several of the platform functions described in Section 2 are performed by a single platform server, which also doubles as a software repository containing, for example, game server software and operating system images. The machines comprising the service platform are dual-processor 3.0 Ghz Xeon-based servers with 2 GB RAM. The game servers are 2.4 Ghz Pentium 4 PCs with 512 MB RAM. All of the servers in our testbed run a Linux 2.4 kernel, and are interconnected with a 100 Mbps switched Ethernet.

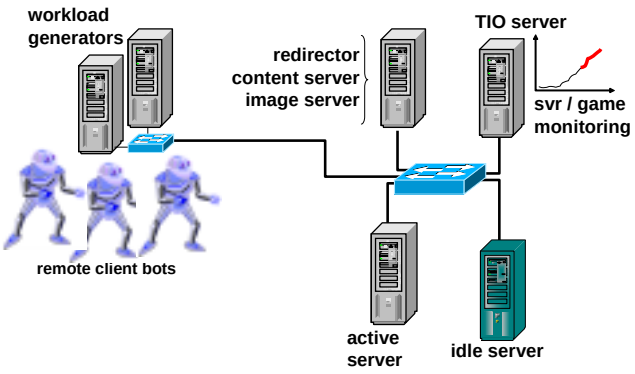


Figure 2: Prototype platform testbed

3.1 Provisioning and data collection

Our provisioning manager is based on IBM Tivoli Intelligent Orchestrator (TIO), an off-the-shelf product that automatically deploys and configures servers, software applications, and network devices in a data center or enterprise environment [5]. TIO is designed to monitor multi-tiered Web-based business application performance and decide how resources should be allocated to different clusters in an on demand fashion. Actual provisioning tasks, such as configuration or installation, are performed using "workflows" which execute sequences of low-level operations in a consistent manner.

The primary function of TIO in our platform is to collect performance and availability metrics from game servers, and decide how to adjust server allocations accordingly. The TIO server polls the active game server for CPU utilization data using SNMP queries. If the utilization crosses a threshold, TIO installs the appropriate game software on an idle server, and adds it to the active set. In addition TIO executes a custom workflow which notifies the redirection server that a new game server is available for joining players. Similarly, when players leave a game server, TIO removes it from the active set and returns it to the idle pool once all of the players have disconnected. The redirection server is an instance of the game server with a slight modification to the communication protocol to allow clients to issue a server assignment query and receive a redirect response to connect to the appropriate server.

Aside from the custom workflows and configuration to install game applications and communicate with platform services, we are using TIO "out-of-the-box." In particular, much of our current effort is aimed at adapting it to use difference performance metrics and more flexible models to decide when to adjust resources (i.e., more sophisticated than a fixed threshold on CPU utilization). We discuss our examination of the suitability of different metrics in Section 4.

3.2 Game metrics and workload generation

We demonstrate our prototype using the open source *Quake II* multiplayer action game, available as both client and server, with a few minor modifications [12]. We removed the hardcoded limit on the number of players so that the relatively powerful servers can be driven into a high utilization regime with a large number of players. We also instrumented the server to export additional performance metrics, beyond those available from the standard *Quake II* server interface (e.g., queried using public tools like *QStat* [7]). One of these is the "slack time," which represents the time left in the server's fixed 100 ms cycle. During each cycle, the game state is computed based on player and object positions and transmitted to clients. Thus, the slack time is an indicator of how close the server is to exceeding its time budget, after which the game state is updated more slowly, noticeably degrading gameplay at the clients. Another metric we introduced is the total traffic transmitted by the server, which is also an indicator of the volume of player and object updates that must be sent to the players. Finally, since TIO uses its own fixed CPU utilization polling interval, along with a smoothing process similar to weighted moving average, we also implemented a finer grained CPU load computation from within the server that is updated every 5 computation cycles (i.e., 500 ms). We refer to this as the "system CPU load."

While many server-side "bots" are available to emulate multiplayer *Quake II*, it is important to recreate actual client traffic for the purposes of demonstrating the service platform. Unfortunately, we were unable to find a suitable open implementation of a *Quake II* client-side bot. As a result, we made some simple modifications to the *Quake II* client to act as a synthetic workload generator. Our "bot" operates externally to the client, sending it native movement and gameplay commands via `stdin`. We also modified the client to sleep periodically to reduce the CPU utilization of the client. This allowed us to instantiate more bots, while still emulating player behavior.

3.3 Auxiliary platform services

The service platform includes a collection of reusable auxiliary services which can be accessed through either the player or the publisher portals. Publisher services help game providers manage specific game titles, for example allowing easy deployment of a new game onto the platform, or enabling distribution of game patches or new content. Player services include common tasks such as authenticating to the game platform (thus allowing things like accounting and billing), discovering other players via directories, and tracking the availability of new games. As described below, our implementation focus thus far has been on two of the key publisher services.

Game deployment service: TIO has the notion of a *Data Center Model* (DCM) that represents the logical and physical assets under its management [8]. In order to allow TIO to automatically deploy server resources with the appropriate software, new games from game providers must be represented in the DCM. In our implementation we represent a new game publisher as a customer with an associated application cluster (i.e., game servers). The cluster is bound in turn to a specific software stack that includes the software necessary to run the game server. For games consisting of components running on separate servers, such as a shared database, game physics computation, etc., each component is represented as a separate application cluster and the associated game software as a separate software stack for the cluster.

Deploying a game requires creation in the DCM of a new application object, together with the corresponding clusters and software stacks. In our current implementation, the Web-based publisher portal accepts some simple policy information, such as system requirements and minimum and maximum number of servers, along with an upload of the game software and installation script.

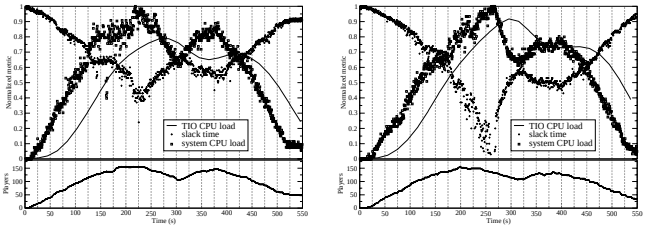
Game content distribution: One of the most important cost factors in operating online games is the significant bandwidth cost of not only game traffic, but also the associated downloads of patches and new game content. Downloads generate significant load on the publisher servers and access network due to the large sizes of the files, the potentially high frequency of patches, and, most importantly, the flash-crowd nature of the downloads as soon as new content is released.

In light of these issues, the content distribution service uses a peer-to-peer architecture in order to deliver content to users quickly while conserving bandwidth at the publisher's content servers. Our implementation is based on the BitTorrent peer-to-peer distribution system [2]. We have completed a partial integration of the *Quake II* client with a modified BitTorrent client. The integrated client has an additional interface to initiate downloads through the peer-to-peer distribution service.

Our modifications to BitTorrent focus on improving its availability and making it network-aware. Our version of the content distribution service provides multiple BitTorrent trackers, thereby eliminating the single point of failure in the standard system. In addition, we enhanced the tracker to provide a list of peer nodes that are nearby to the client, and more likely to offer better download performance (as opposed to the current random list). The key challenge is to determine in a scalable manner which peers are nearest. In our initial prototype, we propose to map peers based on their network prefixes and estimate network distances using Internet coordinate systems.

4. AUTOMATIC PROVISIONING METRICS

In this section, we discuss automatic game server provisioning in our prototype with a focus on evaluating different choices of performance metrics used to trigger resource allocations. Specifically, we consider three key issues: (i) which candidate metric best reflects the load on the game server, (ii) whether provisioning should be based on the most recent measurements, or on a smoothed metric based on past measurements, and (iii) whether different player arrival processes affect the provisioning decision.



(a) $1/\lambda = 1$ s (b) $1/\lambda = 10$ s

Figure 3: CPU utilization and slack time for different mean interarrival times

In our initial implementation we use a relatively simple threshold-based scheme to trigger game server provisioning. Under such a scheme, two different thresholds are typically chosen, one for deciding whether additional resources are necessary, and another to decide when excess resources should be released. These two thresholds should overlap so that frequent reallocations are avoided in the case of transient fluctuation in game performance or server utilization. We adopt TIO's default threshold settings and use our experiments to suggest parameter changes or modifications that enable better performance for games.

4.1 Candidate metrics

We consider several system-level and game-specific performance metrics to better understand the resource requirements in a gaming environment as a function of player population dynamics. We identify four candidates: (i) raw CPU utilization of game servers, (ii) TIO-computed CPU utilization, (iii) the *Quake II* slack time (defined in Section 3), and (iv) total network traffic generated by the *Quake II* server.

The raw CPU utilization metric is a simple instantaneous measure collected roughly every 500 ms from within the game application, while the TIO metric is a smoothed average of CPU utilization collected periodically via SNMP. The slack time is specific to *Quake II*, though it may apply to other games architected in the same way. The fourth metric is useful if the network interface becomes a bottleneck during gameplay, or if managing network bandwidth usage is important in operating the game (e.g., to reduce costs).

Figure 3(a) plots the first three metrics as a function of time during the experiment. Each metric is normalized by the maximum value (i.e., a metric value of 1.0 represents the maximum rather than 100%). The workload is generated by remote bots that connect to the game server (after consulting the redirection server) according to a Poisson process with mean interarrival time $1/\lambda = 1$ second. The number of players connected to the server as the experiment progresses is shown at the bottom of the graph using a separate axis. After the number of active players is sufficient to saturate the server (approximately 200 seconds into the experiment), bots are removed until the load decreases to roughly 50% – 60% CPU utilization. Then bots are added to increase load again during the 300–400 second time period, after which bots begin departing.

We observe that the raw CPU utilization metric tracks the workload on the game server quite accurately as the workload changes. The slack time metric offers similarly accurate tracking, roughly mirroring the raw CPU load. However, for this rapid interarrival rate, we observe a lag with the smoothed TIO CPU utilization metric. For example, the increase in the load during the relatively short 300–400 second time period in the experiment is missed by the TIO metric although the load reaches roughly 90% of the maximum. The smoothed metric reflects it only after the workload decreases.

This simple experiment shows that using a smoothed performance metric based on past measurement samples is not a good estimator of the real workload, particularly when game sessions arrive at a high rate over a short interval. A further implication is that the smoothed metric can lead to inefficient resource utilization. For example, suppose the desired allocation threshold is set at a CPU utilization of 0.7. Then using the TIO-computed metric requires the threshold be set at approximately 0.3 to achieve the same allocation and ensure a similar game experience. This would likely result in overprovisioning (almost twice the number of servers) even after the actual load has decreased on the game servers.

We also examined the network traffic generated by the *Quake II* server for the same workload (not shown). We find that for this game, the network interface is not a bottleneck with reasonable network connectivity. While this may be the case for most of today's FPS games, the metric could be of significance in other games.

4.2 Impact of workload

Timescale: We also performed additional experiments to consider how the player interarrival time affects the choice of appropriate metric. We repeated the above experiment for mean player interarrival times of 10 and 30 seconds, again according to a Poisson process. Figure 3(b) plots the same metrics as in Figure 3(a) for 10 second interarrivals. From these experiments we see that with a more slowly varying workload, the smoothed CPU utilization metric is better able to follow the instantaneous CPU utilization metric. Here, the smoothed metric may be preferred as it is less likely to provision too quickly in response to abrupt but small changes in the workload. Yet it still keeps a timely view of the server load. Clearly, the player arrival statistics play a key role in deciding which metrics should be used for the provisioning decision.

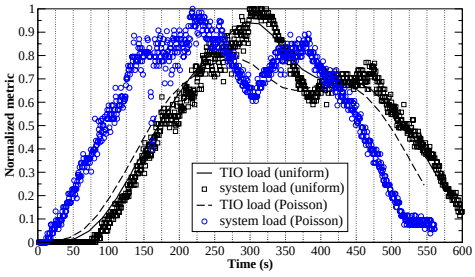


Figure 4: Poisson vs Uniform arrival process

Arrival process: Next, we examine whether the statistical arrival process has an impact on the performance of different metrics. Specifically, we compare the instantaneous and smoothed CPU utilization metrics for Poisson and deterministic game session arrival processes. We choose the same average interarrival time of 1 second between requests for both workloads. Figure 4 shows that with a regular arrival process, the smoothed metric tracks the actual utilization extremely well, even with a very small interarrival time between sessions. We also see that the Poisson arrivals drive the server to peak utilization earlier than the deterministic arrivals.

This is likely due to occasional bursts of arrivals inherent in the stochastic process.

In general, we see that the nature of session arrivals does have a clear impact on which metrics are suitable for game server provisioning. Unlike most Web applications, the soft real-time nature of games requires that sufficient resources always be available to ensure acceptable game performance. This might imply that provisioning decisions should be based on instantaneous metrics, like raw CPU load or slack time, that trigger server allocations as soon as the workload requires it. On the other hand, this can lead to occasional overprovisioning which increases costs for game providers. In this case, a smoothed metric that allocates resources only for persistent workload increases is a better choice, particularly when the smoothed metric is able to track the workload reasonably closely.

5. DISCUSSION

In this section, we discuss some of the additional issues that arise in implementing a shared service platform for games. We also mention a few extensions to the platform currently under development.

Multiplexing games with business applications: Online player populations exhibit a fairly predictable daily usage pattern, with the peak time for gaming typically in the evening and night. Hence, sharing a single server resource pool across gaming applications alone offers little opportunity for statistical multiplexing gains. If we expand the set of applications sharing the server resources to include business applications, however, the opportunities to multiplex and further reduce infrastructure costs become clear. Our adoption of off-the-shelf provisioning software designed for enterprises was strongly motivated by the potential to integrate our gaming service platform with utility offerings for business applications.

Designing on demand games: Our current implementation is directly applicable to server-based games like first-person shooters, as well as other gaming functions that scale up simply by adding additional independent server resources without requiring coordination or synchronization between the servers (e.g., lobby servers for console games). Massively multiplayer role-playing games, however, are currently architected as server clusters hosting a copy of the game world (i.e., shard), each with a large, but isolated group of players. In such games, scalability is on the granularity of shards. Recently proposed distributed game architectures offer communications support to enable a single large game world to be partitioned across multiple servers, allowing all players to interact seamlessly (e.g., see [1, 9]). In such architectures, an on demand infrastructure can dynamically adjust server resources as the player population in different areas of the game changes.

Security issues: The fact that provisioning in our current implementation is done on the granularity of an entire server machine simplifies the security issues arising from more fine grained resource sharing. In practice, however, it may be desirable to host multiple games on a single server, but this clearly requires sufficient protection between the games running on the same server. Additionally, while the peer-to-peer distribution model is appealing from the standpoint of lowering bandwidth costs, it also introduces added liability for game providers who use it: players are expected to allow unknown and untrusted machines to connect to their own machines, thus increasing the risk of exposure to malicious users.

Ongoing work: We are actively enhancing the service platform for use with other types of games and with new services. For example, we are currently adding support to demonstrate automatic provisioning for persistent world multiplayer role-playing games, as well as some of the new distributed game architectures described above. To realize some of the player services mentioned in Section 2, we are adding a number of directory services based on LDAP that account for player preferences when connecting to game servers. Finally, we are continuing to investigate the suitability of different game metrics (i.e., at the application level) for other types of games.

6. SUMMARY

In this paper, we described the development of a shared, on demand infrastructure for hosting large-scale multiplayer game. Our work is inspired by utility computing models for business applications that provide the ability to add IT resources from a pool that may be shared across multiple applications or customers. In our prototype, we leverage commercial off-the-shelf provisioning software and adapt it for game applications. We also implement a number of services that make it easy for publishers to use the platform. We demonstrated the platform with the *Quake II* game.

We study the problem of identifying appropriate performance metrics for provisioning game server resources. Our initial experimental results suggest that the nature of session arrivals is a key factor in determining the suitability of different metrics. Also, there is a clear tradeoff between using smoothed and instantaneous metrics. Smoothed metrics can prevent unnecessary overprovisioning in the presence of short-term workload changes, but are not as accurate in tracking the server utilization when sessions arrive at a high rate.

Our experiences illustrate the feasibility of applying utility computing concepts to an infrastructure for hosting multiplayer online games. Through our ongoing implementation efforts, we are able to identify where existing utility computing technology and open standards may be leveraged, and where game-specific customizations are required.

7. REFERENCES

[1] A. Bharambe, M. Agrawal, and S. Seshan. Mercury: Supporting scalable multi-attribute range queries. In *Proceedings of ACM SIGCOMM*, August 2004.

[2] B. Cohen. Bittorrent. `http://bitconjurer.org/BitTorrent/`, 2004.

[3] Global Grid Forum. `http://www.ggf.org`.

[4] IBM. On demand business. `http://www.ibm.com/ondemand`.

[5] IBM. Tivoli intelligent thinkdynamic orchestrator. `http://www.ibm.com/software/tivoli/`, 2004.

[6] IDC. HP utility data center: Enabling enhanced data center agility. `http://www.hp.com/large/globalsolutions/ae/pdfs/udc_enabling.pdf`, May 2003.

[7] S. Jankowski. Qstat: Real-time game server status. `http://www.qstat.org`.

[8] E. Manoel et al. *Provisioning On Demand: Introducing IBM Tivoli Intelligent ThinkDynamic Orchestrator*. IBM ITSO, December 2003. `http://www.redbooks.ibm.com`.

[9] P. Rosedale and C. Ondrejka. Enabling player-created online worlds with grid computing and streaming. `http://www.gamasutra.com/resource_guide/20030916/rosedale_01.shtml`, September 2003.

[10] D. Saha, S. Sahu, and A. Shaikh. A service platform for on-line games. In *Proceedings of Workshop on Network and System Support for Games (NetGames)*, Redwood City, CA 2003.

[11] Sun Microsystems. N1 Grid – introducing just in time computing. `http://wwws.sun.com/software/solutions/n1/wp-n1.pdf`, 2003.

[12] The QuakeForge Project. Quake2forge. `http://www.quakeforge.net`, 2004.