A Grid-enabled Multi-server Network Game Architecture

Tianqi Wang, Cho-Li Wang, Francis Lau
Department of Computer Science and Information Systems
The University of Hong Kong
Email: {tqwang, clwang, fcmlau}@csis.hku.hk

Abstract

Multiplayer network games attract more and more interest nowadays. In order to make the system scalable, the multi-server architecture is introduced. In such systems, as the dynamic behavior of the users in the virtual world may result in significant workload imbalance between servers, a dynamic load sharing and transferring mechanism is crucial to achieve both quick response time and high throughput. In this paper, we present a multi-server model based on the emerging grid technology for supporting multiplayer network games. The core of this model is based on a new abstraction "gamelet", which represents the execution logic within a partitioned game world and is featured by its high mobility for supporting dynamic load sharing. A prototype of a 3D multiplayer game has been implemented based on gamelet concept. Preliminary results show that our approach can help make a more dynamic and cost effective multi-server network game system.

1. Introduction

The development of high-speed network and processor has contributed to the fast growth in multiplayer network games (MNG) in recent years [1]. A MNG system is one kind of distributed interactive systems, where users located at different places of the world can interact with each other by sharing a consistent virtual environment [2]. Such systems usually aim at providing a sense of realism by incorporating realistic 3D graphics and providing real-time interaction for a large number of users.

There are two key parameters to evaluate the performance of a typical large-scale MNG system: the capacity (CP), which is the maximum number of concurrent users that can interact with each other within the world, and the response time (RT), which is the amount of time between a user issues a command and receives the results. An ideal MNG system should provide lifelike virtual world and real-time interaction for a large number of users in a consistent manner. This goal puts intensive requirements on both computing power and network bandwidth.

In literature, there are two types of architecture supporting such distributed interactive systems: peer-to-peer (P2P) and client-multi-server (CMS). In the P2P architecture, there is no central repository of the world state. Instead, each user maintains its own copy of the world based on messages from all the others. These systems, such as NPSNET [3] and DIVE [4] usually rely on IP-multicast to reduce the total bandwidth consumption. However, since IP-multicast is not widely supported, these systems are limited only to some special purpose applications. Another limitation is that a peer that is weak in computing power or network bandwidth cannot join a world with complex scene contents and lots of users. The reason is that it won't be able to handle network packages, perform consistency protocol [5], calculate world state, and render the scene quickly enough to keep a constant refreshing rate. Therefore such P2P systems usually either limit the world complexity or restrict the number of users that can directly interact with each other. Examples include MiMaze [6] which is a simple multiplayer game that has very low computation requirement, and Microsoft's Age of Empires [7] which limits the number of active players to eight.

To overcome these limitations, a multi-server architecture is adopted in many large-scale distributed interactive systems where a cluster of servers are responsible to do the computation intensive jobs and perform various administration control while remote clients only do the least work such as dead reckoning and scene rendering [8]. This approach makes the complex, large-scale virtual world widely accessible by clients with various configurations such as Tablet PCs and even mobile phones. However, servers may potentially become the bottleneck due to the dynamic behavior of the users in the virtual world. How to perform dynamic load sharing among the servers remains challenging.

Several systems adopting multi-server approach with load sharing support have been developed. Examples include Ring [9], CyberWalk [10] and Asheron's Call [11] etc. Ring supposes a building model as a virtual environment and the whole world is easily partitioned into subdivisions. However, each subdivision is assigned to a fixed server. When a large number of users converge in the same subdivision, the assigned server may still be easily overloaded. CyberWalk is a web-based multi-server distributed virtual walkthrough environment. The world is regularly subdivided into a large number of rectangular cells. Each server manages a region of the whole world that consists of an integer number of cells. The size of each region can be adjusted by transferring boundary cells among neighboring regions. Similar approach is also used in Asheron's Call. It divides a wide-open world into several areas, each of which is managed by one server and the size of the area can be adjusted according to the server workload. However, in these systems if there are several hotspots in the concerned regions/servers, a cascading effect may occur which can seriously affect the performance. Therefore, in a very dynamic environment where the changing of the workload is difficult to predict, the above load-sharing scheme based on data partition can hardly be effective. The reason behind is that these schemes only work within the pre-allocated resources at the time the systems are first loaded into the servers.

In scientific computing area, Grid concept and infrastructure [12] has been developed to address the ever-increasing requirements on the sharing and collaborations among distributed computing resources. Several Grid projects, such as TeraGrid [13], European Data Grid [14] and DOE Science Grid [15] have been successful to provide the high-performance distributed infrastructure for supporting large-scale scientific experiments and analysis. Recently, some efforts have been done to use Grid technology for building large-scale multi-player network games. For example, Butterfly Grid [16] tries to provide an easy to use commercial computing Grid environment for game developers and free them from the complex network issues. They also provide the high-performance networked servers for the publishers to hold their games.

We see the idea to dynamically discover resources and aggregate enough computing power on demand to deliver the desired qualities of service can potentially change the way current multi-player network games are developed and operated. In this paper, we propose a multi-server architecture based on the emerging grid technology for supporting multiplayer network games. We introduce a concept named gamelet, which is an execution abstraction within the partitioned virtual world and featured by its high mobility for supporting dynamic load balancing. Based on gamelet, we can build a more dynamic and cost-effective MNG system, which only uses the minimum resources to deliver the desired qualities of service. We will discuss how we build our prototype and show the performance gain.

This paper is organized as follows. In Section 2, we present our proposed multi-server model. In Section 3, we discuss the gamelet concept. The gamelet-based multi-server architecture is presented in Section 4. In Section 5, we discuss the prototype design and implementation. Performance evaluations are presented in Section 6 and we conclude in Section 7.

2. Multi-server Model

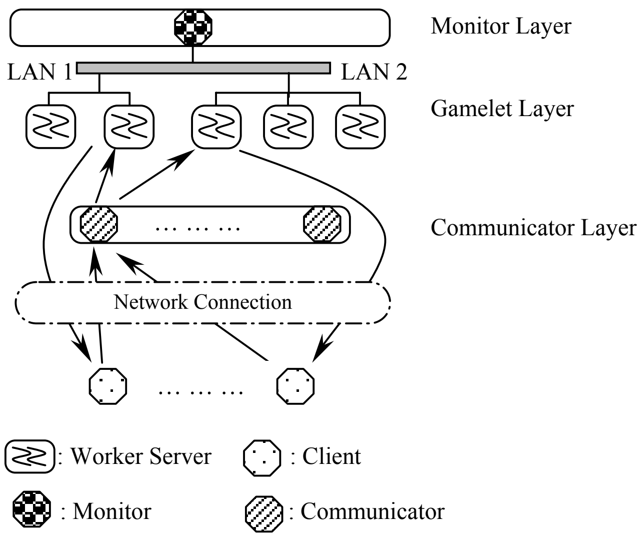

Fig. 1. Multi-server Model

In order to make a scalable system, we adopt a layered design approach. As shown in Figure 1, our multi-server model contains three abstract layers. The monitor layer, gamelet layer and communicator layer. The whole virtual world is partitioned logically into some partitions, each of which will be assigned to one gamelet for processing. Gamelets are running on the worker server. One worker server can hold several gamelets at the same time.

The joining and leaving process is like this: when a user wants to join the game, he first contacts the monitor server. Then the monitor will assign one communicator to the client. After this process, a client will always send messages to this communicator. The communicator knows the partition logic and will forward the messages to the corresponding gamelets. After various processing, the updated world states are directly sent back to the clients. In case the communicator becomes the network bottleneck, it can be fully replicated to share the workload.

A monitor periodically collects the workload parameters of each worker server and makes decisions on how to adjust the workload among the worker servers and when to increase or decrease the worker server number. The load adjustment strategies are tunable and executed by the monitor.

Worker servers do all the computation related to the game logic. In our model, a work server's job includes: receive and process command packets from the clients, buffer the command packets in a timely order, execute commands according to the synchronization requirements, keep track of dynamic objects, and calculate the world states. Area of Interest (AOI) management is also included, which means server only sends the necessary updates that are within that player's potential field of sight. Similar jobs are done by Quake [18] servers.

Since the worker servers can be increased and decreased dynamically, the advantage of the existence of the communicator is obvious. It will make the complicated worker server adjustment activities transparent to the clients. We will discuss how a monitor and a communicator cooperate to perform the gamelet migration process later in Section 4.

The client side job will include: encapsulate user operations into data packets, send out the packets to the communicator and use its cache of the game states plus any updates from the server to render the virtual world. Client also does simple collision detection, deck reckoning and various self-healings to make the game more enjoyable and fluent.

```
Gamelet {
  Data:
    { World Contents
    { Performance Parameters
  Processing:
    { Computation Part
    { Control Part
}
```

Fig. 2. Gamelet Structure.

3. Gamelet

We define "gamelet" as an execution abstraction to process the workload introduced by certain related objects within the virtual world. Figure 2 shows the structure of a gamelet, which consists of data component and processing component. In the data component, world contents store the current world states and performance parameters record the CPU load, network load, total user number and any other monitoring variables. In the processing component, computation part includes all the processing that a game server is supposed to do as described in the previous section. Control part contains methods that manage gamelet migration and perform performance calculation. Objects in the whole world, including both static objects and avatars, are grouped according to their logical relationships. The relationships are application specific and can be in terms of space or time. For example, all users participating in a football match or a virtual meeting session can form a group and be effectively assigned to the same gamelet to process. A gamelet has the following characteristics:

- Load Awareness. A gamelet is able to detect and monitor its performance parameters by itself and these values can be retrieved to support load-balancing strategy.

- Remote Control. A gamelet has standard methods that enable an authorized monitor to control remotely. For example, a gamelet in our current prototype has the initiation, starting, exit methods as well as various methods supporting remote retrieval of performance parameters.

- Embedded Synchronization. A gamelet should support world content synchronization to prevent two gamelets that share some overlapping work from going out of synchronization. Whenever necessary, two gamelets are able to communicate reliably to exchange their the world contents.

Based on our gamelet concept, the whole system will consist of a number of gamelets, each of which can be created and destroyed at run-time. Gamelets will run on the suitable worker servers to corporately deliver the desired qualities of service to the clients.

4. Gamelet-based Multi-server Architecture

Our grid-enabled multi-server architecture is based on Globus Toolkit 3.0 [17], which has become the de facto standard for the Grid technology [12]. For the simplicity of the discussion, we only depict the service components that are most important to our gamelet-based architecture in Figure 3.

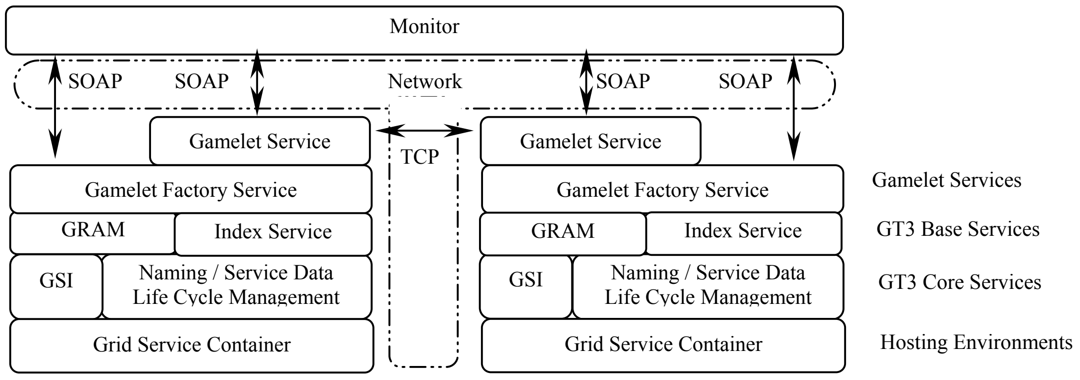

Fig 3. Gamelet-based Multi-server Architecture.

At the lowest level is the hosting environment, which might be a J2EE web container on Windows or Linux operating systems. Running in the service container, GT3 provides two layers of services. The lower level is the GT3 core services. Grid Security Infrastructure (GSI) provides methods to ensure the Grid resources are only accessible to authorized requestors. Each grid service instance will have a unique Grid Service Handle (GSH) managed by the Naming service and be associated with a structured collection of information named Service Data that can be easily queried by the requestors. Life Cycle Management provides functionalities to control a service instance throughout its life cycle starting from the creation till the destruction. The upper level is the GT3 base services. The Grid Resource Allocation and Management (GRAM) service provides a way to submit and monitor jobs remotely, while Index Service (IS) can be used to aggregate and query the Service Data of various service instances.

In our gamelet-based architecture, through IS the monitor can dynamically discover the desired Gamelet Factory services that can provide certain qualities of service in terms of speed, availability, cost as well as other natures. The monitor utilizes GRAM to create and manage a gamelet. The communication between a monitor and a gamelet is through standard Simple Object Access Protocol (SOAP) and is based on the well-defined interfaces that are described using Web Service Description Language. The Gamelet Factory services will periodically register their GSHs into a registry. From the registry, a monitor can locate a set of Gamelet Factory services, query their Service Data, and reliably create and manage a service instance with the help of IS and GRAM.

After the monitor creates a gamelet, it will periodically observe its performance. In case the monitor makes a decision that a gamelet need to be migrated, it will try to locate a new reference to another Gamelet Factory service and create a new gamelet. Just before the migration begins, it also notifies the communicator to store the incoming packets temporarily. Once the new gamelet is up, monitor will control the old gamelet to transfer the world content to the newly created one through TCP connection. Finally, the monitor will notice the communicator to update gamelet information and the following incoming client packets will be forwarded to the new gamelet for processing.

5. Prototype Design and Implementation

The multi-server prototype used to evaluate our proposed architecture and load transfer scheme follows the multi-server model discussed in Section 2. It is designed to simulate a real large-scale MNG system, but is generic and flexible enough to support various experiments.

We use UDP communication between the client, communicator and gamelet, since UDP can give much shorter response time than TCP. To counteract the unreliable delivery characteristic, each client packet will include two consecutive commands: the current command and previous command. In such a way, each command has two chances to reach the server. So the client can tolerate a lost rate up to 50% provided the lost packets are evenly distributed. Between two gamelets or a gamelet and a communicator, TCP communication is used since the world content need to be reliably delivered.

We define the performance metrics as follows. The average RT is the average time between each client sending out a packet and receiving the confirmation from the server that the command has been executed. Being executed means the command, possibly with other clients' commands together, is considered by the

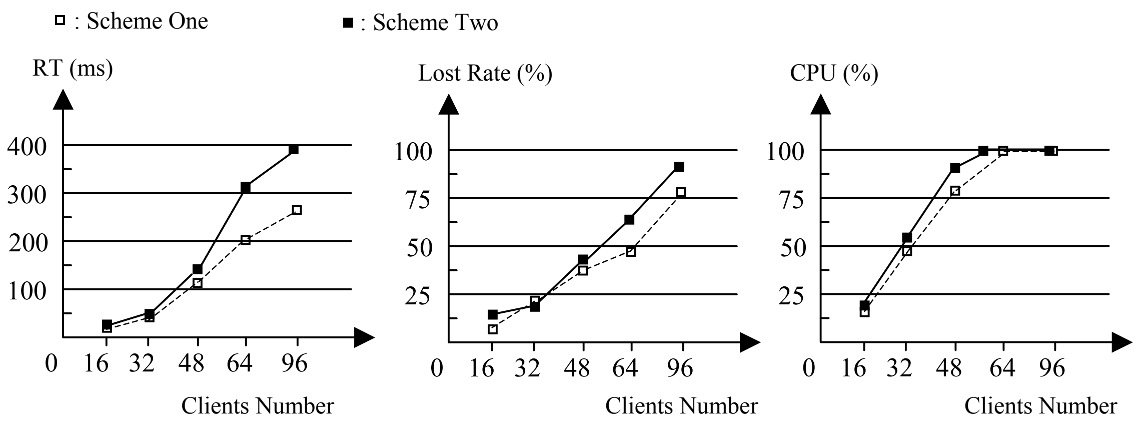

Fig 4. Performance Evaluation One

server, the world is updated accordingly, and the results have been sent to all the clients that share the same AOI. Since the command will have an additional delay if it reaches the server through the second packets, the RT is calculated by this formula:

$$RT = RT*(1\text{-}LostRate) + (RT+TI)* LostRate$$

TI is the time interval between two consecutive packets. We define system CP as the maximum number of concurrent users that can interact with each other within the world with reasonable average RT (<200 ms) and lost rate (<50%).

The world in our prototype is a 100*100*20 size 3D environment. Each user (avatar) is represented as a 3D object. We have developed a client simulator in Java that can simulate a certain number of clients, each of which will send out packets at a constant rate. The packet used is 32 bytes, which consists of avatar positions, commands and timestamps. Packet is first serialized into a byte stream before sending out and will be deserialized at the receiver side. The serialized packet size is 151 bytes.

The server will receive the packets from the outside, deserialize and execute them in a timely order. After we update any client state, we calculate the distance between this client and all the others. Suppose we use DIS(I, J) to represent the distance between client I and client J. For a client I, we need only send its updated state to client J if DIS(I, J) < AOI. In current implementation, our avatar uses simple 3D object and we want to simulate the collision detection among complex 3D objects. Our method is to add a certain amount of floating point calculation which is proportional to 1/DIS(I, J) to ensure that the closer two 3D objects are, the more computation is needed to calculate their states.

In our preliminary experiments, we only consider the spatial relationship and simply divide the world evenly into three equal-sized areas (each is a 33*100*20 area). Each area will be assigned to one gamelet to process. Two consecutive gamelet will have an overlapping area of length 2*AOI, since one avatar's AOI might go across two gamelets. The partition information is stored by the communicator and will be used to route the client packets to the corresponding gamelets for processing. The avatar is evenly distributed within the world and each of them performs a random movement every each 100 ms. AOI is set to 10. The gamelet, monitor are implemented on top of GT3.0.1 and runs on Linux kernel 2.4.2 with P3 733 MHz CPU, 256MB RAM and 100Mbps Ethernet. The client simulator and communicator run on Windows 2000 Professional with P4 2.2 GHz CPU, 512MB RAM and 100Mbps Ethernet. All components are implemented using J2SE 1.4.2.

6. Performance Evaluation

We will study the performance of our proposed architecture through various experiments, using from one to three worker servers. The gamelet creation and migration performance is also studied.

6.1. Evaluation One

We first compare the performance of the following two schemes. Scheme one: there is no partition. All the client packets are forwarded to one worker server to process. Scheme two: the world is partitioned into three areas as described in Section 5, and each of them

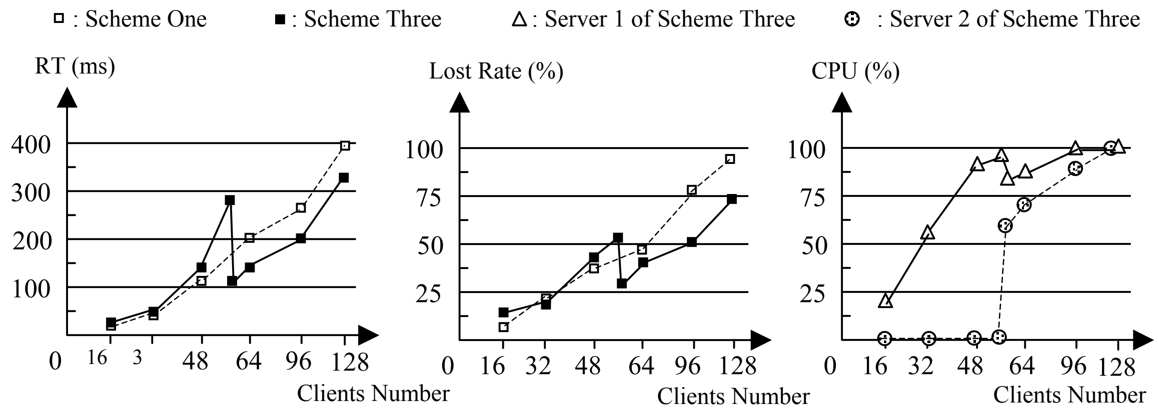

Fig 5. Performance Evaluation Two

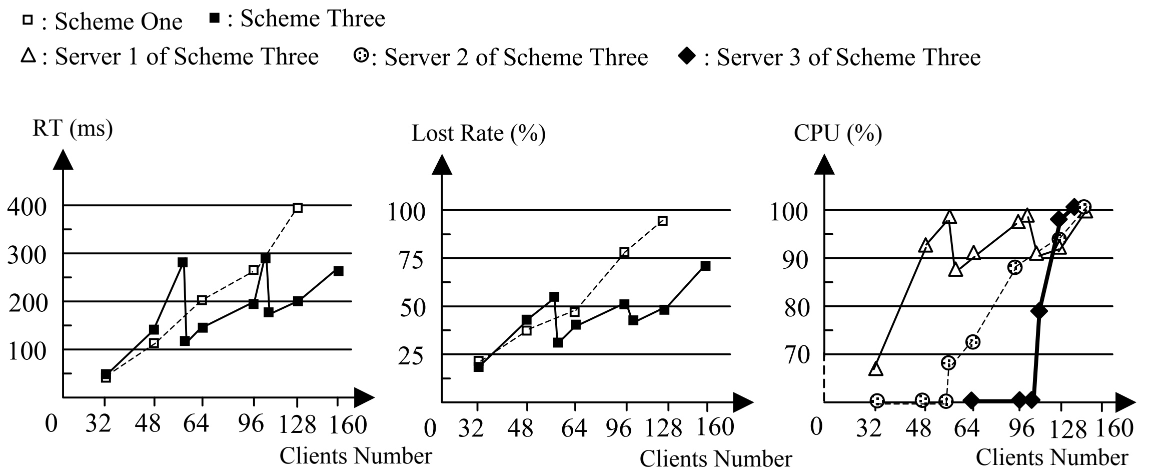

Fig 6. Performance Evaluation Three

is assigned to one gamelet to process. All the three gamelets are running on the same worker server.

From the CPU utilization graph in Figure 4, we find that the CPU load increases as the client number increases. The CPU load of scheme two is slightly larger than that of scheme one. The reason is that in scheme two, each gamelet will do some work that is overlapping with its neighboring gamelets. Therefore, the total work in scheme two will be larger than that in scheme one. However, this difference is not great. Both of them approach 100% when the total client number is between 48 and 64. We observe that just at the point when the client number reaches this threshold, that is when the CPU load approaches 100%, both the RT and lost rate in scheme one and scheme two become unsatisfied. After this point, we can clearly see that the performance of the first scheme is better than the second scheme. However, before this point, we cannot see much difference in both RT and lost rate. So our conclusion is that not until the CPU utilization of the machine hits some threshold, our proposed approach can almost achieve the same performance with scheme one, which is the traditional used approach. The network load of the communicator is no more than 1.9Mbps when there are 96 clients. So network is not the bottleneck and there is no need to replicate the communicator.

6.2. Evaluation Two

Now we consider scheme three: the gamelet can be migrated dynamically and we will use 2 servers at most. The monitor will periodically retrieve the gamelet workload information. Once it finds the CPU load of the worker server that a gamelet runs on is approaching 100%, the monitor will initiate a gamelet migration process. Since the client is evenly distributed in our experiment, load-balancing strategy is also simple. The monitor will simply choose any one of the three gamelets to migration to the new place.

From the CPU utilization graph in Figure 5, we find that the gamelet migration is initiated when the client number is about 56, at which point the CPU load of server 1 is approaching 100%. After the migration is completed, CPU load of server 2 rises from 0 to 65% and CPU load of server 1 decreases to 87%. In the RT graph, we see the RT is relatively higher at the point when the client number is 56 than that shown in Figure 4, however the lost rate is quite similar to that shown in Figure 4. The reason is that when a gamelet migration is initiated, the world content needs to be reliably transferred to the new gamelet. In our experiment, the migration process takes about 40 ms. Since the communicator will store the incoming clients' packets during the gamelet migration process and forward them later, the migration process has little effect on the lost rate. After the migration completes, the client could again enjoy a shorter RT and lower lost rate until the client number is about 96 and the CPU load of server 1 approaches 100% again. The gamelet creation time ranges from about 430 ms to several seconds, however, this process won't count to the gamelet migration time since it can be done before the migration process.

6.3. Evaluation Three

In this experiment, we also consider scheme one and scheme three. However, the gamelet can be migrated to 3 servers at most this time. The results are shown in Figure 6.

In the CPU utilization graph, we can see that as the client number increases, the third worker server is added when the total client number is slightly larger than 96. Accordingly, in the RT graph there is a short delay during the migration process. Since the clients have already experienced a long RT, this short delay won't lead to obvious user distraction. After the migration completed, the RT and the lost rate are both within the tolerable range until the client number is about 128 and all the worker servers become saturated. So we conclude that our proposed approach will increase the system CP from the original 64 to about 128. Similarly, as the client number decreases, the gamelets will then converge to two and finally one worker server again, but still can provide the reasonable performance. Therefore, our approach is more dynamic and cost-effective, since gamelet can be created and destroyed dynamically and the system only uses necessary resources to provide the desired qualities of services.

7. Conclusions

In this paper, we present a new concept named gamelet, which is an execution abstraction within the world. Based on the gamelet concept and the cutting-edged Grid technology, we propose a multi-server architecture to build more dynamic and cost-effective MNG systems.

We have evaluated the performance of our proposed scheme through our prototype. The results show that our approach enables the system scale dynamically according to the system workload and only uses the minimum resources to provide reasonable performance to the clients. Near future work includes porting the gamelet-based architecture into the HKU Gideon Cluster, which consists of 300 PC nodes, and evaluating the maximum potential power of the gamelet in a more complex environment.

References

[1] A. Jarett, J. Estanislao, E. Dunin, J. MacLean, B. Robbins, D. Rohrl, J. Welch, J. Valadares, *IGDA Online Games White Paper. 2nd Edition*, 2003.

[2] S. Singhal and M. Zyda, *Networked Virtual Environments: Design and Implementation*, Addison Wesley, 1999.

[3] M. Capps, D. McGregor, D. Brutzman, M. Zyda, "NPSNET-V: A New Beginning for Dynamically Extensible Virtual Environments", *IEEE Computer Graphics and Applications*, 2000, pp. 12-15.

[4] O. Hagsand, "Interactive Multi-user VEs in the DIVE system", *IEEE Multimedia*, Vol: 3, No.: 1, 1996, pp.30–39.

[5] E. Cronin, B. Filstrup, Anthony R. Kurc, Sugih Jamin, "An Efficient Synchronization Mechanism for Mirrored Game Architectures", *ACM. NetGames*, 2002, pp. 67-73.

[6] C. Diot and L. Gautier, "A Distributed Architecture for Multiplayer Interactive Applications on the Internet", *IEEE Networks Magazine*, vol. 13, 1999, pp. 6–15.

[7] Age of Empires. http://www.microsoft.com/games/empires/.

[8] J. Smed, T. Kaukoranta, H. Hakonen, "A Review on Networking and Multiplayer Computer Games", Technical Report 454, Turku Centre for Computer Science, 2002.

[9] Thomas A. Funkhouser, "RING: a Client-server System for Multi-user Virtual Environments", *Proceedings of the 1995 symposium on Interactive 3D graphics*, Monterey, California, United States, 1995, pp.85-92.

[10] N. Beatrice, S. Antonio, L. Rynson, L. Frederick, "A Multiserver Architecture for Distributed Virtual Walkthrough", *Proceedings of ACM Symposium on Virtual Reality, Software and Technology 2002,* Hong-Kong, 2002.

[11] Asheron's Call. http://www.microsoft.com/games/zone/asheronscall/.

[12] Kesselman, J. Nick, I. Foster, C., and S. Tuecke, "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration", Open Grid Service Infrastructure WG, Global Grid Forum, June 22, 2002.

[13] C. Catlett, "The Philosophy of TeraGrid: Building an Open, Extensible, Distributed TeraScale Facility", *Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid*, 2002, page: 8.

[14] D.Bosio, J.Casey, A.Frohner, L.Guy et al, "Next Generation EU DataGrid Data Management Services", *Computing in High Energy Physics*, 2003.

[15] W. Johnston, "Issues for Using Computing and Data Grids for Large-Scale Science and Engineering", *Workshop on Clusters and Computational Grids for Scientific Computing*, 2000.

[16] Butterfly Grid. http://www.butterfly.net/.

[17] Globus Toolkit 3.0. http://www.globus.org/.

[18] Quake . http://www.idsoftware.com/.