An Architecture to Support Scalable Distributed Virtual Environment Systems on Grid *

Tianqi Wang, Cho-Li Wang, Francis C.M. Lau

Department of Computer Science, The University of Hong Kong, Hong Kong
{tqwang, clwang, fcmlau}@cs.hku.hk

Abstract. A Distributed Virtual Environment (DVE) system offers a computer-generated virtual world in which individuals located at different places in the physical world can interact with one another. In order to achieve real-time response for a large user base, DVE systems need to have a scalable architecture. In this paper, we present the design of a grid-enabled service oriented framework for facilitating the construction of scalable DVE systems on computing grids. A service component called "gamelet" is proposed, whose distinctive mark is its high mobility for supporting dynamic load sharing. We propose a gamelet migration protocol which can ensure the transparency and efficiency of gamelet migration, and an adaptive gamelet load-balancing (AGLB) algorithm for making gamelet redistribution decisions at runtime. The algorithm considers both the synchronization costs of the DVE system and network latencies inherent in the grid nodes. The activities of the users and the heterogeneity of grid resources are also considered in order to carry out load sharing more effectively. We evaluate the performance of the proposed mechanisms through a multiplayer online game prototype implemented using the Globus toolkit. The results show that our approach can achieve faster response times and higher throughputs than existing approaches.

1 Introduction

A Distributed Virtual Environment (DVE) system is a software system of which users located at different places can interact with one another to share a consistent virtual environment [16]. In a DVE system, a user is represented by an entity called "avatar" whose states are a function of user inputs. Many applications of DVE systems exist, including military team training [6], virtual shopping malls [11], interactive e-learning [14], and multiplayer online games [3]. An ideal DVE system should be able to emulate a realistic world and support real time interactions for a large number of concurrent users in a consistent fashion. The calculations and propagations of state changes of the simulated world, however, would easily translate into intensive requirements on computing power, network bandwidth, etc., making the design of a large-scale DVE system very challenging.

* This research is supported in part by the China National Grid project (863 program) and the HKU Foundation Seed Grant 28506002.

Recently, much effort has been put into building DVE systems on computational grids [1] [2]. A grid is a system that coordinates resources belonging to different organizations using standard, open, general-purpose protocols and interfaces to deliver nontrivial qualities of service [9]. In a grid, aggregated resources can potentially offer qualities of services that have only been possible with supercomputers in the past. A grid could be used to support real-time interactions in a DVE system having a large number of concurrent users. However, unlike a cluster, a grid generally consists of heterogeneous and dynamic resources. Grid nodes vary in computing power and the bandwidth between grid nodes may change from time to time. Such characteristics of the grid give rise to several new challenges. One challenge is to re-design the conventional monolithic model of a DVE system so that it becomes an open service-oriented system that can fit into the current grid frameworks (e.g., OGSA). Another challenge is to ensure that certain qualities of service, e.g., real time response, be maintained. Since the aggregate behaviors of users in a DVE system can easily translate into significant workload imbalances which could lead to unpredictable delays in world states computation, dynamic load sharing is very much needed. Traditional approaches for load sharing and migration that have worked well in a cluster environment might not be directly applicable to the grid environment. The load sharing and migration design should be adaptive to the resource heterogeneity and dynamic nature of the grid.

In this paper, we propose a flexible and scalable service-oriented framework that can facilitate the construction of multiserver DVE systems on a grid. The framework adopts a service-oriented approach that can fit well in an OGSA-compliant grid environment. A service component called "gamelet" is introduced, which serves as the basic building block. A gamelet is characterized by its load awareness, mobility, and embedded synchronization for supporting a DVE system with a partitioned virtual environment. Existing multiserver DVE systems based on spatial partitioning scheme can easily be mapped into our gamelet-based framework. Our proposal includes a gamelet migration protocol and an adaptive gamelet load-balancing (AGLB) algorithm. The design principles are such that they would meet the special requirements, e.g., latency tolerance and resource adaption, of building a DVE system in a grid environment. We show that with our design, load balancing and migration can be performed effectively and efficiently.

The rest of the paper is organized as follows. In Section 2, we present briefly the background of DVE systems. In Section 3, we discuss the gamelet concept and the system framework. In Section 4, we present the gamelet migration protocol and the AGLB algorithm. Section 5 discusses the design of a prototype. Section 6 presents the performance evaluation based on the prototype. Section 7 discusses some related work. Section 8 concludes the paper.

2 DVE Systems

A DVE system aims at a sense of realism and an immersive experience by incorporating realistic 3D graphics and providing real-time interactions to users. A DVE system typically has the following features:

- Consistent world. Consistency refers to the similarity of different users' views of the virtual environment. A user has an area of influence in the virtual world, and all the other avatars in the area should be able to view its activities in real time and the perceived states should be the same.
- Realistic graphics. Fast developments of graphics hardware have made it possible for a DVE system to operate with high-resolution 3D objects. Efficient and realistic animations and accurate collision detections are desired features of new emerging DVE systems.
- Distributed users. Although users share the same virtual environment, they might be located at different places and access the system through different devices. The DVE system should create an immersive environment that can provide a highly responsive performance to mask the physical differences of the participants.

The virtual environment in a DVE system may consist of static world content, e.g., buildings and rooms, as well as dynamic world content, e.g., avatars. State changes of the dynamic world content including positions and velocities of avatars should be transmitted to clients as quickly as possible in order to achieve a feeling of reality. Inconsistencies in state changes must be swiftly detected and resolved in real time. Such activities can be very memory and computing power demanding, especially in an unreliable network environment where messages may get lost [7]. In fact, real-time, accurate collision detection and response calculation of complicated 3D objects are known to be computationally intensive [13] [15]. Hence, it is now common to adopt a multiserver architecture whereby the workload would be distributed across several servers based on a certain partitioning scheme. The servers are responsible for generating global world states as well as performing various administration management tasks while clients only perform operations such as dead reckoning and scene display [17]. Here are some examples of server task: managing input commands, computing world states, rollbacks and state synchronizations, server-side optimizations for area of interest (AOI) management, packet compression, cheat detection and denial-of-service attacks prevention [4].

When too many users happen to crowd into a certain area in the virtual environment, a hot spot will form. Too many hot spots in a DVE system can easily result in server overloading. Therefore, workload mobility and dynamic load sharing are highly desirable features in the design of scalable DVE systems. These features, however, cannot be easily implemented in a grid environment as traditional DVE systems are not at all grid-ready due to their monolithic design and non-compatibility with an open and service-oriented grid computing

environment. The dynamics of grid resources complicate the situation even further since factors such as network latency and resource heterogeneity need to be considered if load sharing and migration are to be performed.

3 Gamelet-based System Framework

3.1 Gamelet Definition

The core building block of our system is called a "gamelet". It is a mobile service component responsible for processing the workload introduced by a partitioned virtual environment. The gamelet concept is closely related to that of traditional problem decomposition for parallel execution: each partition corresponds to a gamelet running in a grid node. A gamelet is designed with the following features:

- Load Awareness. A gamelet currently residing in a server is able to detect and monitor its own workload, including the CPU load of the server, the network load (in terms of bps), the total number of users, etc. The gamelet is at all times ready to provide the necessary information to be used by a third party for load migration decisions.
- High Mobility. A gamelet can be controlled by an authorized remote component through standard interfaces. Supported methods include initiation, destruction, processing, and migration.
- Embedded Synchronization. A gamelet is able to communicate with other gamelets to synchronize their world states whenever necessary. The synchronization protocol is embedded in the gamelet.

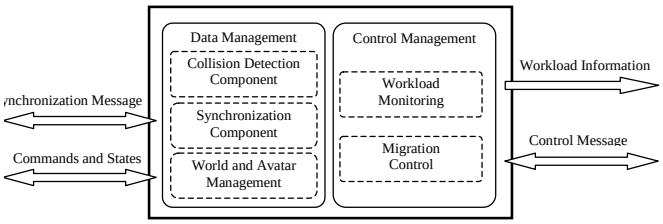

Fig. 1. Gamelet Structure.

A gamelet can be created or destroyed dynamically to support runtime load sharing for a gamelet-based DVE system. Since a gamelet can be migrated to any grid node, the result is a scalable DVE system that sits on top aggregated distributed resources. The system is no longer limited by locally available resources.

Figure 1 shows the structure of a gamelet. The data management section includes three components. The collision detection component is responsible for computing when objects collide with one another and how they should respond. The synchronization component is responsible for ensuring data consistency among multiple partitions. The world and avatar management component is responsible for managing objects in the virtual environment and communicating with clients. The control management section includes two components. The workload monitor component records the CPU load, network load, number of users, and any other monitoring variables. The migration control component implements methods to be used in gamelet migration and remote enquiries of a gamelet's workload states.

3.2 Grid-enabled System Framework

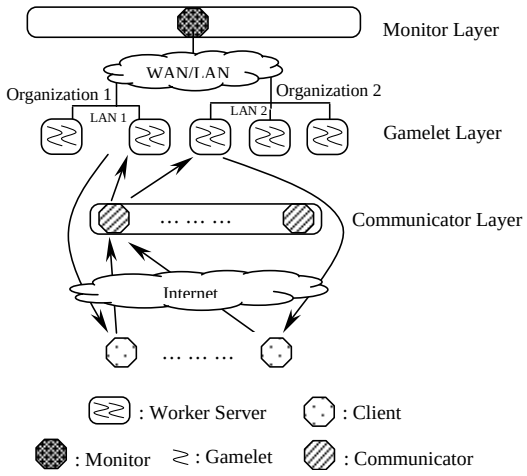

Fig. 2. Three-layer Model.

We propose a three-layer framework for gamelets (Figure 2). The function of the monitor layer is to periodically check the different workload parameters of the worker servers, and to make decisions on how to adjust the workload among the servers. The load sharing strategies are tunable and are embedded in the monitor component. For example, the user may define a threshold to limit the CPU workload difference among the servers. A monitor is also responsible for managing the life cycle of a gamelet, e.g., creation and destruction of a gamelet instance.

The gamelet layer consists of a set of worker servers where gamelets are run. The whole virtual world is logically divided into a set of partitions, each of which is assigned to one gamelet. One worker server can support several gamelets at the same time. A monitor will acquire workload data from the gamelets based on which migration decisions could be made.

The communicator layer consists of a number of communicators and acts as the bridge between the gamelets and the clients. To join the world, a client will first contact a well-known monitor which will assign a communicator to the client, after which the client will always send messages via this communicator. Communicators know the predefined partition rules and will route messages intelligently in case messages are needed by more than one gamelet simultaneously. For example, when an avatar's area of influence is across several gamelets, the user's messages will be routed to all the gamelets. In such a case, the gamelets are responsible for ensuring world consistency. The communicator cooperates with the monitor to perform gamelet migration and will try to make the migration process transparent to the clients. In cases where the communicator is overloaded, the monitor can assign new users with a new communicator so that the workload can be shared.

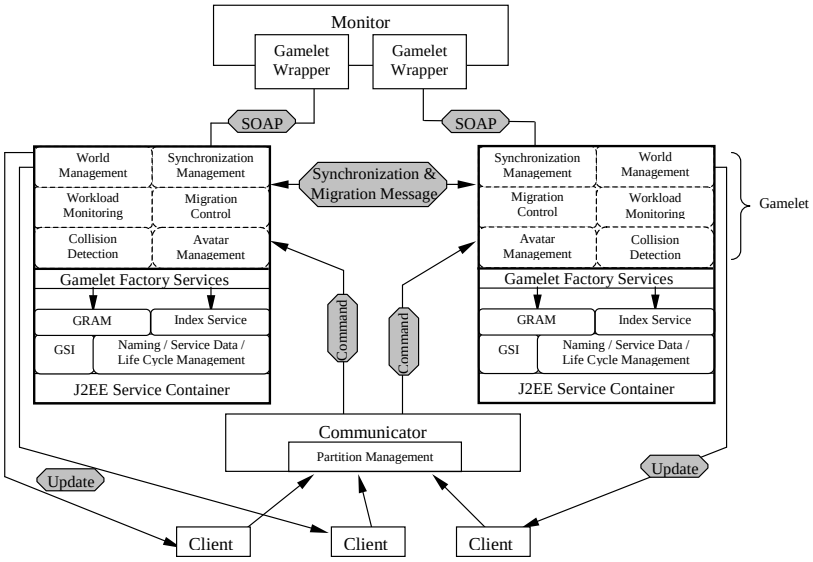

Fig. 3. Gamelet-based System Framework.

Figure 3 shows the detailed architecture of the framework. In order to allow for a flexible control of multiple gamelets, each gamelet is implemented with

a wrapper in the monitor component. The gamelet wrapper encapsulates the complicated grid related protocols. This design separates the monitor component totally from the underlying grid libraries, which can be easily reused in a new grid environment.

The gamelet service component is designed to fit the OGSA-compliant grid service architecture. This architecture comprises several layers. At the lowest layer is the hosting environment such as a J2EE application server. Above the server container is a core service layer (GT3) which provides the following supports. The Grid Security Infrastructure (GSI) is used to ensure the secure communication and authentication among the gamelets in an open network. Each gamelet service instance has a unique Grid Service Handle (GSH) managed by the Naming Service and is associated with a structured collection of information called Service Data that can be easily queried by requesters. The Life Cycle Management service provides methods to control a gamelet throughout its life cycle from creation to destruction.

GT3 base services are based on the GT3 core services. The Gamelet Factory service uses the Grid Resource Allocation and Management (GRAM) service to enable remote creation and management of gamelets. A set of Service Data, e.g., those required by the collision detection methods, is associated with a Gamelet Factory service through the Index Service. The gamelet factory service enables several stateful gamelet service instances to be created at the same grid node concurrently. Each gamelet ensures that the dynamic world states are carefully maintained during the running of the DVE system. A gamelet factory will periodically register its GSH into a registry, from where a monitor can locate a set of gamelet factory services and reliably create and manage gamelet service instances.

4 Gamelet Migration and Load Balancing Algorithm

4.1 Gamelet Migration

Gamelet migration is performed under the cooperation between the monitor and the communicator. The migration protocol is as follows.

A monitor periodically queries the gamelets for their workloads. When the monitor figures that a gamelet is in need of migration, it will try to locate a reference of a new gamelet factory service from the registry, create a new gamelet, and make it ready for the migration act. The monitor then notifies the communicator to store the incoming packets temporarily in a resizable message queue, and directs the source gamelet to package and serialize its dynamic world content. It also transmits the address of the new gamelet to the old gamelet. Afterwards, the old gamelet transfers the dynamic content to the new gamelet. The monitor will be notified when the content transmission is finished and the new gamelet is ready to process the incoming packets. Finally, the monitor notifies the communicator of the completion of the migration, and the old address of the gamelet is replaced by the new one. The communicator will then forward the stored packets

according to the updated client-gamelet mapping table. Newly arrived packets will be routed to the new gamelet for processing.

Since in a grid environment the newly discovered grid node might be located far away from the server where the old gamelet was running, the latency between them might be in the order of several hundred milliseconds. To mitigate this latency problem arising from the gamelet migration process, the communicator component will manage and buffer the incoming packets during the migration period, which will be forwarded to the new gamelet for processing when the new gamelet has completely taken over. This minimizes message losses caused by gamelet migration.

To cut down on the gamelet migration time, grid libraries are preloaded and the procedure to pack the world content is executed concurrently with gamelet creation. Therefore, if the new gamelet is ready to work, the old gamelet can start transferring the world content to it immediately. This design can save much of the gamelet migration time. More on the gamelet creation and migration time will be discussed later. Besides, the client-gamelet mapping table in the communicator component, which is initially defined based on the partition logic of the DVE systems, can be automatically updated when migration takes place. A client always contacts the same communicator without having to know which gamelet is actually working behind the scene. The server side's gamelet migration activities are completely transparent to the client.

4.2 Adaptive Gamelet Load Balancing Algorithm

We use an adaptive gamelet load balancing (AGLB) algorithm for load balancing of the DVE system in a grid environment. The word *adaptive* has two meanings. Firstly, the algorithm adapts to network latencies in making gamelet migration decisions. This is achieved through a cost model which takes into account both gamelet synchronization costs and inter-server latencies. Secondly, the algorithm evaluates each gamelet based on the activities of the clients being managed and adapts to the resource heterogeneity of the grid nodes.

In the AGLB algorithm, the threshold δ_m is used to judge whether a new load balancing process is necessary for server m. Server m will be regarded as overloaded if its CPU usage exceeds δ_m. The value of the threshold can be set beforehand. As a grid server may become unavailable due to management or other reasons, δ_m can be set to zero so that server m will be considered overloaded until all its gamelets have been migrated away. After that the server m can be removed from the system.

Let N be the total number of gamelets and Q the total number of servers. Then, we have the following notations.

- $Graph(V, E)$: the gamelet partition graph. The i-th vertex, V_i, corresponds to the i-th gamelet, G_i. $E_{i,j}$ represents the edge between V_i and V_j. $i, j = 1, 2, 3, \ldots, N$.
- $BW_{m,n}$: the bandwidth between server m and server n, where $m, n = 1, 2, 3, \ldots, Q$. $BW_{m,m}$ is the bandwidth of server m's memory bus.

- $Latency(m, n, t)$: the network latency between server m and server n at time t, where $m, n = 1, 2, 3, \ldots, Q$.
- $C_{i,j}$: the communication traffic between gamelet G_i and G_j. $i, j = 1, 2, 3, \ldots, N$.
- $W_{i,j}$: the weight of edge $E_{i,j}$. $W_{i,j}$ is the number of avatars whose states need to be synchronized between gamelet G_i and gamelet G_j. $W_{i,j} = C_{i,j}/s_{unit}$, where s_{unit} is the size of one avatar's state. $i, j = 1, 2, 3, \ldots, N$. $C_{i,i} \approx 0$.
- $Syn(G_i, m)$: the estimated synchronization cost of gamelet G_i running in server m.

$$Syn(G_i, m) = \sum_{G_j \in \phi} W_{i,j} \times (Latency_{m,n,t} + C_{i,j}/BW_{m,n})$$

  where ϕ is the set of neighboring gamelets of G_i and G_j runs in server n. $i = 1, 2, 3, \ldots, N$, and $m, n = 1, 2, 3, \ldots, Q$.
- $Cost(G_i, m, n)$: the cost of migrating gamelet G_i from server m to server n.

$$Cost(G_i, m, n) = Syn(G_i, n) - Syn(G_i, m)$$

  $Syn(G_i, m)$ is the pre-migration synchronization cost and $Syn(G_i, n)$ is the post-migration synchronization cost; the calculation assumes that G_i has been migrated to server n. $i = 1, 2, 3, \ldots, N$ and $m, n = 1, 2, 3, \ldots, Q$.
- δ_m: the threshold used to judge if server m is overloaded. $\delta_m \in [0, 1]$. $m = 1, 2, 3, \ldots, Q$.
- $CPower_m$: the computing power of server m in terms of floating-point operations per second. $m = 1, 2, 3, \ldots, Q$.
- $Val(G_i, m, t)$: a weighted packet sending rate used to evaluate how much workload G_i introduces to server m at time t.

$$Val(G_i, m, t) = \sum_{P_k \in \Upsilon} (Rate(P_k, m) \times Weight(P_k)),$$

  where Υ is a set of command packets used in the DVE system. $Rate(P_k, m)$ is the receiving rate of command packet P_k in server m. $Weight(P_k)$ is a relative value indicating how much workload packet P_k will introduce to the server. $Weight(P_k) \in (0, 1]$. $i = 1, 2, 3, \ldots, N$ and $m = 1, 2, 3, \ldots, Q$.
- $Percentage(G_i, m)$: the percentage of CPU load that gamelet G_i introduces to server m.

$$Percentage(G_i, m) = Val(G_i, m, t) / \sum_{G_j \in \psi} Val(G_j, m, t),$$

  where ψ is a set of gamelets currently running in server m. $i = 1, 2, 3, \ldots, N$ and $m = 1, 2, 3, \ldots, Q$.
- $ELoad(G_i, m, n)$: the estimated load that gamelet G_i running in server m will introduce to server n.

$$ELoad(G_i, m, n) = Percentage(G_i, m) \times CPower_m / CPower_n$$

  where $i = 1, 2, 3, \ldots, N$ and $m, n = 1, 2, 3, \ldots, Q$.

Whenever there emerge some overloaded servers, the monitor will build a gamelet graph and then execute the AGLB algorithm.

Gamelet Graph Building Algorithm:

1. *For each gamelet G_i in the DVE system, create a vertex V_i in the gamelet partition graph.*
2. *For any two vertices V_i and V_j in the gamelet partition graph, create an edge $E_{i,j}$ with weight $W_{i,j}$*

Adaptive Gamelet Load Balancing (AGLB) Algorithm:

1. *Find server s which has the highest CPU load among the overloaded servers*
2. *For each gamelet G_i running in server s*
4. *For each of the other servers, g*
5. *Calculate $Cost(G_i, s, g)$*
6. *Starting from the smallest one, for each $Cost(G_i, s, g)$ {*
7. *Calculate $ELoad(G_i, s, g)$*
8. *If server g will not be overloaded after receiving gamelet G_i {*
9. *Decide to migrate gamelet G_i to server g; break }}*
10. *If no migration decision is made, add a new grid server to the system*

The strength of the algorithm is that the workload model is more accurate than the other approaches that only take into account the number of clients or density. The reason to use a weighted packet rate is that different participants might have different packet sending rates and different commands might lead to different workloads. Besides, the AGLB algorithm is designed to combine the synchronization costs, the network latencies among the grid nodes, and the heterogeneity of grid resources in making the gamelet redistribution decisions. The algorithm can also dynamically integrate a new grid node into the system when the runtime situation calls for it. All these features make the AGLB algorithm highly effective in a grid environment.

5 Prototype Design and Implementation

A prototype of a 3D multiplayer game has been developed. It is designed to simulate a realistic large-scale DVE system, and yet sufficiently generic and flexible to support various kinds of experiments. The virtual environment is an open 3D world of size $100 \times 100 \times 20$, which is divided into 16 equal-sized cubes, and the overlapping length of the neighboring partitions is 5. Each client packet is 32 bytes long, including an avatar ID, a command ID, a position and a timestamp.

The client simulator is a multithreaded program, which simulates a number of randomly distributed clients. Each client sends out a packet every 100 ms with a randomly selected command. The simulator can also specify certain hotspot areas where clients will quadruple their packet sending rate. Figure 4(a) shows the workload distribution of a virtual environment where all the avatars are equally active, and Figure 4(b) shows a virtual environment with three hotspots.

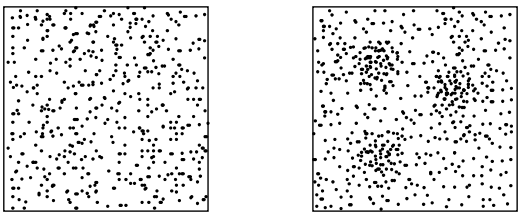

Fig. 4. Workload Distributions.

The communicator uses a data structure called *mapping decision table* for routing client packets to the corresponding gamelets for processing. Gamelets receive packets from the communicator and execute them in time order. After the states of any avatar have been updated, the system calculates the distance between this avatar and all the others. Suppose $DIS_{i,j}$ is the distance between avatar i and avatar j and the area of interest (AOI) of each user is a circle with radius R_{AOI}. For each avatar i, the updated states are only sent to client (avatar) j if $DIS_{i,j} < R_{AOI}$. In the experiment, we set R_{AOI} to 5. The gamelet performs collision detection after each command is executed. Each avatar is represented as a simple 3D object and a bounding box collision detection algorithm is used. To simulate collision detection among complex 3D objects, a certain amount of floating point calculations proportional to $1/DIS_{i,j}$ is added. This is to ensure that the closer two 3D objects are, the more computation is needed to arrive at more accurate states. If an avatar is in an overlapping area managed by several gamelets, each gamelet will calculate its state separately, and then synchronize the state with the other gamelets every 100 ms. Gamelet migration is implemented using object serialization. Only the dynamic states, e.g., an avatar's position and velocity, are packed and transmitted. The communication between a gamelet wrapper and a gamelet instance is through the Simple Object Access Protocol (SOAP) and is based on well-defined interfaces expressed in the Web Service Description Language (WSDL).

There are two performance parameters used in the performance evaluation: response time (RT) and system capacity (SC). Response time is the average measured interval between the time a client sends out a packet and the time it receives the confirmation from a server that the command has been executed. Being executed means that the command, possibly together with other commands, has resulted in the world being updated accordingly, and the results have been sent to all the clients that share the same AOI. Each command is sent to the server in two consecutive packets to minimize the chance of client packet losses. System capacity is defined as the maximum number of participants that the

system can support so that the interactions in the world will have a reasonable average RT ($\leq$ 200 ms) and packet loss rate ($\leq$ 50%).

The gamelet and the monitor are implemented using GT3.0. All components are implemented using J2SE 1.4.2 and run on Linux kernel 2.4.18 with a system configuration of P4 2.0 GHz CPU, 512 MB RAM, and 100 Mbps Ethernet [18]. The network latencies among the grid nodes are within several milliseconds if not specified explicitly.

6 Performance Results

6.1 Gamelet Creation and Migration

We first study the time of gamelet creation and migration when a monitor sequentially creates gamelets in several servers. We find that when a monitor creates the first gamelet in the first server, it usually takes about 7–8 seconds. This is because various GT3 runtime libraries need to be loaded and initialized at both the monitor and the gamelet server. The creation of the first gamelet in the second server takes less than 3 seconds. This is because the monitor has already loaded and created the necessary libraries. The creation of the second gamelet in the same server will only take about 100 ms. However, 100 ms are still substantial for a time-sensitive DVE system. Our proposed gamelet migration protocol is designed to mask the gamelet creation time, whereby the monitor will notify the communicator to stop forwarding client packets for processing only after the new gamelet has been created.

We observe that the gamelet migration time changes from 61 ms to 113 ms when increasing the number of clients from 8 to 256. So a gamelet migration process will introduce a short delay. Three factors account for the delay: interactions between the monitor and the communicator, partition management, and content transmission. The delay time increases with the size of the gamelet, e.g., an increased number of avatars that need to be transferred. The migration process nevertheless will not add to the packet loss rate, since the communicator will store the incoming packets temporarily and forward them later.

6.2 Multi-server with Dynamic Gamelet Migration

We compare the performance of the AGLB algorithm with the popular even-avatar load balancing (EALB) algorithm which tries to balance the number of avatars that each server holds. Two to eight servers are used and the client simulator is configured to generate a virtual environment with three hotspots, as shown in Figure 4. The hotspots span gamelets 1 and 2, gamelets 7 and 8, gamelets 9 and 10, respectively. Table 1 shows the network latency configurations among the servers in the experiments. The network latency is simulated by adding some delay in the gamelet component. In the AGLB algorithm, a server is regarded as overloaded when its CPU usage is larger than 90%.

Latency(ms)	S1	S2	S3	S4	S5	S6	S7	S8
S1	1	100	20	20	20	20	20	20
S2	100	1	200	150	100	100	100	50
S3	20	200	1	20	20	20	20	20
S4	20	150	20	1	20	20	20	20
S5	20	100	20	20	1	20	20	20
S6	20	100	20	20	20	1	20	20
S7	20	100	20	20	20	20	1	20
S8	20	50	20	20	20	20	20	1

Table 1. Network Latencies Among the Servers.

Evaluation 1 Initially, the 16 gamelets are all in server 1. As the number of clients increases, gamelets are migrated to server 2 according to the AGLB algorithm. We examine two partitioning approaches of the EALB algorithm, called EALB-1 and EALB-2, which are as shown in Figure 5. Note that in the partition graph, a thick edge linking two gamelets indicates that there is a hotspot spanning the gamelets.

52 Clients	CPU Load		Inter-server	RT (ms)		Average RT
2 Servers	S1	S2	Traffic	S1	S2	(ms)
AGLB	91%	82%	43.8 Kbps	245	193	219.0
EALB-1	100%	73%	35.1 Kbps	699	182	440.5
EALB-2	88%	89%	149.1 Kbps	261	252	256.5

Table 2. Performance Comparison with Two Servers.

Table 2 shows the performance comparison of the three approaches when 52 clients are used. We see that the EALB-1 approach generates the least amount of inter-server traffic; it however has the worst average response time. The reason is that it groups most of the hotspot areas in one server, which results in very long response time. Since the EALB algorithm does not consider the activity of each client, the workload of the two servers are not balanced, even though the number of clients that each server contains is the same. We observe that server 1 is more overloaded. Its response time is nearly 700 ms, representing the worst average response time of the three approaches. The EALB-2 approach balances the CPU load of the two servers. Since it separates the hotspots into different servers, however, the inter-server traffic is nearly four times that of the AGLB algorithm. For each avatar whose states need to be synchronized, additional delay will be added to its normal response time. Therefore, although the CPU loads of the two servers are more balanced, the response time of the server under the EALB-2 approach is still worse than that under the AGLB approach.

The AGLB algorithm can generate a well balanced CPU load distribution among the servers while keeping the inter-server traffic at a minimum. This is because it uses a more accurate workload model and at the same time adapts to the network latencies among the servers in making the migration decisions. Table 2 shows that the user-perceived response time of the AGLB approach is the best among the three approaches.

Evaluation 2 We also study the performance of the three different approaches with 4 and 8 servers respectively. The partition results are shown in Figure 6 and Figure 7.

91 Clients	CPU Load				Inter-server	RT (ms)				Average RT
4 Servers	S1	S2	S3	S4	Traffic	S1	S2	S3	S4	(ms)
AGLB	90%	89%	79%	81%	276.8 Kbps	223	321	210	241	248.7
EALB-1	100%	100%	99%	42%	184.1 Kbps	503	572	512	101	422.0
EALB-2	100%	100%	68%	70%	322.9 Kbps	539	654	242	172	401.7

Table 3. Performance Comparisons with Four Servers.

176 Clients	CPU Load								Inter-server
8 Servers	S1	S2	S3	S4	S5	S6	S7	S8	Traffic
AGLB	90%	72%	69%	71%	71%	90%	69%	68%	891.1 Kbps
EALB-1	100%	36%	35%	100%	100%	33%	36%	37%	475.3 Kbps
EALB-2	90%	88%	89%	91%	89%	90%	35%	36%	742.6 Kbps

Table 4. Performance Comparisons with Eight Servers (1).

176 Clients	RT (ms)								Average RT
8 Servers	S1	S2	S3	S4	S5	S6	S7	S8	(ms)
AGLB	270	323	180	180	184	247	175	158	214.6
EALB-1	1402	230	104	1434	1334	116	202	111	616.6
EALB-2	235	402	309	278	214	235	137	110	240.0

Table 5. Performance Comparisons with Eight Servers (2).

Table 3 shows the performance result with four servers. We observe that the EALB-1 approach has the least inter-server traffic. However, server 1, server 2 and server 3 are all overloaded, which is the worst performance among the three approaches. Similarly, with the EALB-2 approach, there are two overloaded servers; so the average response time is as unsatisfactory as that of the EALB-1 approach. The influence of the inter-server latency is worth mentioning. For example, under the AGLB approach, server 2 has similar load to server 1, but the response time of server 2 is about 100 ms longer than that of server 1. This is because the network connection of server 2 is worse than that of server 1. For the partition result of the AGLB approach (Figure 6), the average network

latency is about 30 ms between server 1 and other servers while it is about 180 ms for server 2. This also explains why the response time of server 3 is much longer than that of server 4 under the EALB-2 approach even though server 3 is less loaded.

Table 4 and Table 5 show the CPU load and the response time of eight servers under the three different approaches, respectively. It is easy to explain why the EALB-1 approach has the worst performance. The reason is that the server load is not well balanced: three servers are overloaded. The AGLB approach generates the most amount of network traffic but can still provide the fastest response time. Comparing the load distribution of the servers under the AGLB and the EALB-2 approaches, we cannot see any obvious differences. In fact, a large amount of network traffic does not necessarily lead to a longer response time. The determining factor is how much of the traffic is transferred through the slow network link. Under the AGLB approach, only server 7 and server 8 need to synchronize with server 2 which has relatively better network connections; but under the EALB-2 approach, there are four servers that need to synchronize with server 2, which leads to a worse performance on average.

By decreasing the total number of clients in the virtual world, we can get the maximum number of clients the servers can support while providing a satisfactory response time. Figure 8 gives the system capacity comparisons of the three approaches. The system with our proposed AGLB algorithm has much larger capacity than the EALB approaches. The reason is two-fold. Firstly, the AGLB algorithm can estimate the CPU load that each gamelet brings to the server more accurately than the EALB approaches. The fact that a client in a hotspot is more active than others is taken into account, which helps produce a better balanced load distribution; otherwise, the server holding several hotspots can be easily overloaded, which will greatly influence the average performance of the DVE system. Secondly, the AGLB algorithm tries to minimize the inter-communications among the servers that have bad network connections. This is achieved by incorporating the network performance in the gamelet migration cost model. With these reasons, we can easily explain the system capacity differences shown in Figure 8 for a fixed number of servers under the three different approaches. We can see that our proposed AGLB algorithm can improve the system capacity by 80%, 72% and 62% when compared with the EALB-1 approach, and by 17%, 62% and 13% when compared with the EALB-2 approach using 2, 4, 8 servers respectively. The AGLB algorithm therefore contributes to a more scalable and cost-effective system.

7 Related Work

Several DVE projects have dealt with load balancing in their design. CittaTron [12] is a multiserver networked game for a wide area network with load balancing and reduction mechanisms. It partitions the virtual world into several regions, each of which is assigned to a server. Although CittaTron provides dynamic load balancing support, its algorithm does not consider the activities of

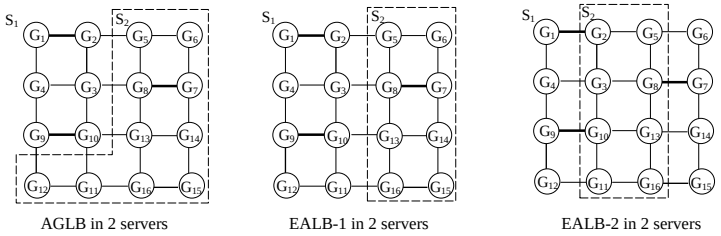

Fig. 5. Partitioned States Under Different Approaches Using Two Servers.

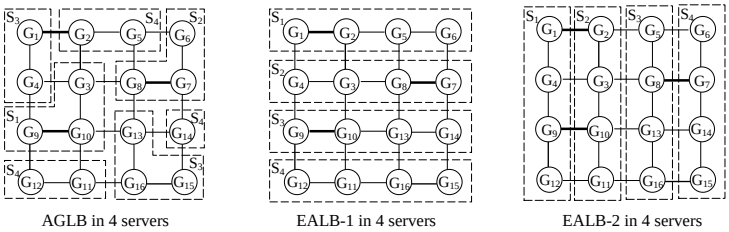

Fig. 6. Partitioned States Under Different Approaches Using Four Servers.

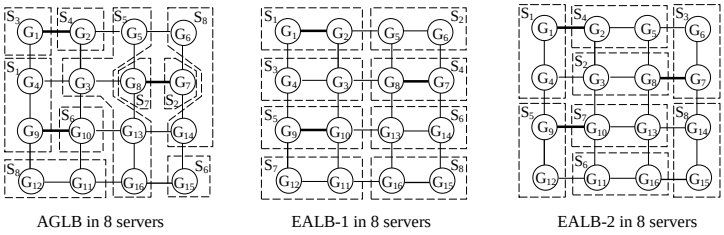

Fig. 7. Partitioned States Under Different Approaches Using Eight Servers.

the clients. Therefore, its mechanism may not be effective in a highly dynamic virtual environment where different users may introduce different workloads in different regions. NetEffect [8] is an architecture for supporting large-scale 3D virtual environments. The whole virtual world is separated into several communities that are managed by multiple servers. A user can can only interact with users in the same community, which limits the interactivity of the system. Whenever the load of a peer server exceeds a certain threshold, one or more communities may be transferred from the heavy loaded server to a less loaded server dynamically. However, this load migration process is not transparent to clients, and a user has to sign in the system again following each migration. CyberWalk [5] is a web-based multiserver distributed virtual walkthrough environment which adopts a standard client-server architecture. The virtual world is

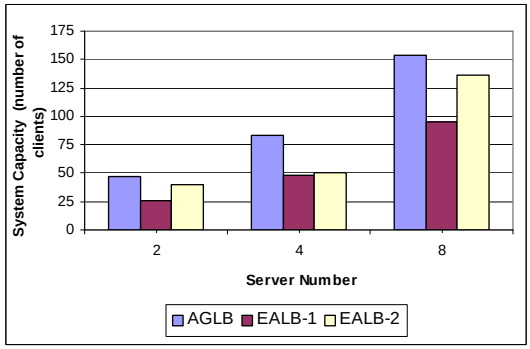

Fig. 8. System Capacity Comparisons.

partitioned into many equal-sized cells. Each server manages a region of the virtual world consisting of a number of cells. An adaptive load balancing algorithm is used to decrease the workload of an overloaded server by adjusting the number of cells managed by the server. The load balancing strategy works well if objects and clients are evenly distributed in the virtual environment. However, if there exist hotspots in neighboring regions, the load balancing strategy may suffer from a cascading effect which could seriously affect the system performance.

There are also projects that implement DVE systems on a grid. Cal-(IT)2 Game Grid [2] provides the first ever grid infrastructure for game research, focusing on the areas of communication and game service protocols. Butterfly grid [1] is an infrastructure for supporting massive multiplayer games (MMG) via on-demand computing. It tries to provide an easy to use commercial grid-computing environment for both game developers and game publishers. The current implementation of the Butterfly grid is based on Globus Toolkit 2 [10]. As the OGSA becomes available, it is expected that Butterfly will migrate to an OGSA compliant service platform. Using the Butterfly Grid's computing resources and framework, game developers can develop, test and deploy games more easily. There are problems, however, remaining to be addressed, one of which is about scalability. Although dynamic user redirection can be performed by the gateway server, no dynamic load migration is supported. Besides, as the new OGSA protocols become better defined and more widely available, how to cast the existing Butterfly model in an OGSA compliant model needs to be further explored.

8 Conclusions

In this paper, we address the challenges of building DVE systems on a grid. We present the design of a grid-enabled service oriented framework for the building of DVE systems. A foundation service component called "gamelet" is pro-

posed. Using gamelets, existing client-server DVE systems with a partitioned virtual world can be readily mapped into a grid environment. We also propose a gamelet migration protocol which is designed to be latency-tolerant. Our adaptive gamelet load-balancing (AGLB) algorithm uses an accurate workload model and can adapt to network latencies when making migration decisions. We have evaluated the feasibility and performance of our approach through various experiments with a generic DVE system prototype. Results show that our approach can achieve reasonably fast response times and high throughputs.

The current system assumes a simple two-way synchronization scheme. More complicated synchronization protocols might be needed for applications that have more stringent consistency and response time requirements. For future work, such synchronization protocols and their influences on the gamelet-based DVE system could be studied. Another possible direction is to study how the communicator component can be improved to help gamelets synchronize more efficiently. For instance, the communicator can try to predict the processing time of the incoming client packets and proactively discard any packets that will be deemed too late to the clients.

References

1. Butterfly grid. http://www.butterfly.net/.
2. California institute for telecommunications and information technology. http://www.calit2.net.
3. Igda online games white paper - 2nd edition. http://www.igda.org/, 2003.
4. N.E. Baughman and B.N. Levine. Cheat-proof playout for centralized and distributed online. In *Proceedings of IEEE INFOCOM*, pages 104–113, Anchorage, Alaska, 2001. ACM Press.
5. N. Beatrice, S. Antonio, L. Rynson, and L. Frederick. A multiserver architecture for distributed virtual walkthrough. In *Proceedings of ACM Symposium on Virtual Reality, Software and Technology 2002*, Hong-Kong, 2002.
6. M. Capps, D. McGregor, D. Brutzman, and M. Zyda. NPSNET-V: A New Beginning for Dynamically Extensible Virtual Environments. *IEEE Computer Graphics and Applications*, 20(5):12–15, 2000.
7. E. Cronin, B. Filstrup, Anthony R. Kurc, and Sugih Jamin. An efficient synchronization mechanism for mirrored game architectures. In *Proceedings of ACM NetGames2002*, pages 67–73, Braunschweig, Germany, 2002.
8. Tapas K. Das, Gurminder Singh, Alex Mitchell, P. Senthil Kumar, and Kevin McGee. Neteffect: a network architecture for large-scale multi-user virtual worlds. In *Proceedings of the ACM symposium on Virtual reality software and technology*, pages 157–163, Lausanne, Switzerland, 1997.
9. I. Foster. What is the grid? a three point checklist. *GRIDToday*, 2002.
10. I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications*, 11(2):115–129, 1998.
11. S. Han, M. Lim, D. Lee, H. Kim, B. Koo, S. Kim, and B. Choi. Scalable network support for 3d virtual shopping mall. pages 336–345, Gyeongju, Korea, 2002.
12. M. Hori, K. Fujikawa, T. Iseri, and H. Miyahara. Cittatron: a multiple-server networked game with load adjustment mechanism on the internet. In *Proceedings of the 2001 SCS Euromedia Conference*, pages 253–260, Valencia Spain, 2001.

13. M. Matthew and J. Wilhelms. Collision Detection and Response for Computer Animation. *Computer Graphics*, 22(4):289–298, 1988.
14. Tohei Nitta, Kazuhiro Fujita, and Sachio Cono. An application of distributed virtual environment to foreign language. In *ASEE/IEEE Frontiers in Education Conference*, volume 1, pages F1G/9–F1G15, Kansas City, MO, United States, 2000.
15. Alex P. Pentland. Computational complexity versus simulated environments. *Computer Graphics*, 24(2):185–192, 1990.
16. S. Singhal and M. Zyda. *Networked Virtual Environments: Design and Implementation*. The MIT Press, Cambridge, MA, 1999.
17. J. Smed, T. Kaukoranta, and H. Hakonen. A Review on Networking and Multiplayer Computer Games. Technical report, Turku Centre for Computer Science, 2002.
18. The University of Hong Kong. System Research Group of the Department of Computer Science. Hku gideon300 grid. http://www.srg.csis.hku/gideon/, 2004.