

Gamelet: A Mobile Service Component for Building Multi-server Distributed Virtual Environment on Grid ^{*}

Tianqi Wang, Cho-Li Wang, Francis Lau

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

Abstract. A DVE system provides a computer-generated virtual world where individuals located at different places could interact with each other. In this paper, we present the design of a grid-enabled service oriented framework for facilitating the building of DVE systems on Grid. A service component named “gamelet” is proposed. Each gamelet is characterized by its load awareness, high mobility, and embedded synchronization. Based on gamelet, we show how to re-design the existing monopolistic model of a DVE system into an open and service-oriented system that can fit into current Grid/OGSA framework. We also demonstrate an adaptive gamelet load-balancing (AGL) algorithm that helps the DVE system achieve better performance. We evaluate the performance through a multiplayer online game prototype implemented on Globus Toolkit. Results show that our approach can achieve faster response time and higher throughput.

1 Introduction

A Distributed Virtual Environment (DVE) system is a software system through which people that are geographically dispersed over the world can interact with each other by sharing a consistent environment in terms of space, presence and time [1]. These environments usually aim for a sense of realism and an immerse experience by incorporating realistic 3D graphics and providing real-time interactions for a large number of concurrent users. DVEs are now widely used in virtual shopping mall, interactive e-learning and multiplayer online games.

A typical DVE system should be able to support a life-like world and real-time interactions for a large number of users in a consistent way. This translates into intensive requirements on both the computing power and the network bandwidth. Multi-server architecture is a popular solution to tackle these problems [2, 10]. In addition, several techniques [3] such as dead reckoning, packet aggregation, and area of interest (AOI) management have been proposed to reduce the network bandwidth consumption and the servers’ computing load.

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

In recent years, some projects have tried to build DVE systems over Grid. Cal-(IT)2 Game Grid [8] provides the first ever grid infrastructure for games research in the area of communication system and game service protocols. Butterfly Grid [9] tries to provide an easy to use commercial grid-computing environment for both the game developers and game publishers. However, a few new challenges remain to be solved. One is how to re-design the existing monopolistic model of a DVE system into an open and service-oriented system that can fit into current Grid/OGSA framework [7]. Another lies in how to ensure a certain qualities of service from the perspective of end users. In a DVE system, each user is represented as an entity called “avatar” whose state is controlled by the user’s input commands. Since users can move and act freely, it may incur significant workload imbalance among servers, which in turn causes unpredictable delays in computing the global world state and delivering the state to the client machines. This makes it very difficult to achieve real-time interactions.

In this paper, we propose a flexible and scalable service-oriented framework based on a component called “gamelet” to support the building of a DVE system on Grid. Based on the gamelet framework, we further propose an adaptive gamelet load-balancing (AGL) algorithm that is able to make the DVE system more scalable and cost-effective.

The rest of the paper is organized as follows. In Section 2, we discuss the gamelet concept and system framework. In Section 3, we present the prototype design. Section 4 presents the performance evaluations. Section 5 presents some related work. Section 6 concludes the paper.

2 Gamelet-based System Framework

2.1 Gamelet Definition

We define *gamelet* as a mobile service component that is responsible for processing the workload introduced by a partitioned virtual environment. A group of related objects, including both static and dynamic objects, form one partition and will be assigned to one gamelet for processing. For example, in a room-based virtual environment, one or several rooms can form one partition. In a session-based virtual environment, like a virtual meeting system, users joining the same session can form one partition. For system with a persistent wide-open virtual environment, overlapping partition technique [11] can be used to support smooth visual interactions among the participants across multiple partitions. A gamelet has the following unique characteristics:

- Load Awareness. A gamelet is able to detect and monitor the CPU and network load it created on its hosting server. It also provides a rich set of API for load inquiry.
- High Mobility. A gamelet can move freely and transparently to a new grid node within our framework. The mobility is achieved at the application level, and the world states are transferred using object serialization technique.

- Embedded Synchronization. In DVE, there will be some visual interactions among two partitions. Two gamelets should be able to communicate reliably to synchronize their world states whenever necessary.

With gamelets, we can accelerate the application development circles as most existing partition schemes adopted by the multi-server DVE systems can easily adapt to the gamelet framework. Moreover, a gamelet can be migrated among the server nodes to support load balancing at run-time.

2.2 System Framework

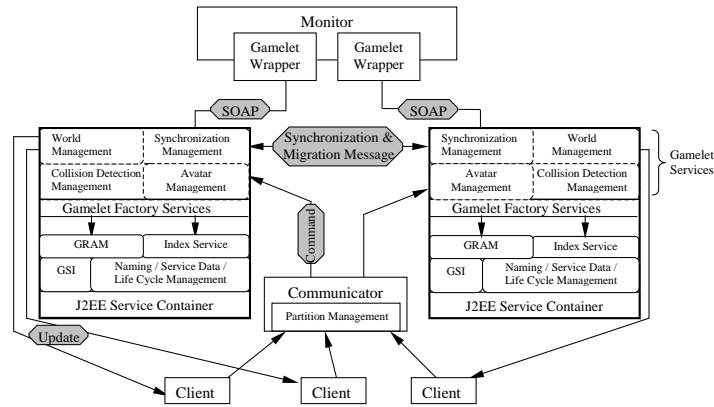


Fig. 1. Gamelet-based System Framework

Figure 1 shows the overall picture of the system framework. It consists of three kinds of components: monitor, gamelet and communicator. A monitor is responsible to collect the workload information of each gamelet periodically and execute the load balancing algorithms. The communicator acts as the gateway for all the incoming packets and will do intelligent message routing, which means it will forward an incoming packet to all the related gamelets according to the predefined partition logic. In such a case, the gamelets are responsible to execute the embedded synchronization protocol to provide a consistent view to the clients. To join the world, a participant will first contact a well-known monitor. The monitor assigns a communicator to the client. The client will always send messages to this communicator. The communicator then forwards the messages to the corresponding gamelets for processing and the updated world states are sent back directly to the clients.

In our framework, GT3 core services provide the following supports. Grid Security Infrastructure (GSI) is used to ensure the secure communication and authentication among the gamelets over an open network. Each gamelet service instance will have 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 requestors. 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. Gamelet Factory service makes use of the Grid Resource Allocation and Management (GRAM) service to enable remote creation and management of a gamelet service. A set of service data, e.g., the cost of the gamelet service, is associated with a Gamelet Factory service through the use of Index Service. Using the Index Service, the monitor can dynamically discover the desired Gamelet Factory services that can meet certain basic requirements in terms of speed, availability, cost and etc.

To enable a flexible control over multiple gamelets, each gamelet has a corresponding wrapper in the monitor. This design encapsulates the complicated grid related stuffs and makes the monitor totally separated from the underline grid libraries. So the monitor component is more portable. The gamelet factory service enables several stateful gamelet services to be created at the same grid node concurrently. 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. The monitor and gamelet form a client-server relationship that is in line with the OGSA client-server programming model [12].

2.3 Gamelet Migration

Gamelet migration enables load balancing. The migration protocol is as follows. When the monitor determines that a gamelet is in need of migration, it first tries to locate a GSH of another Gamelet Factory service and then creates a new gamelet to serve as the target of the migration. Once the new gamelet has been created, the monitor will notify the communicator to store the incoming packets temporarily in a resizable message queue. Then the monitor directs the old gamelet to transfer its world content to the new one. Finally, the monitor will notify the communicator the completion of the migration. Communicator will then forward the stored packets and the succeeding packets according to the updated mapping information. Therefore, the migration process won't result in notable message loss.

2.4 Load Balancing Algorithm

To support the load balancing of a DVE system in the grid environment, we propose a new adaptive gamelet load-balancing (AGL) algorithm. The word *adaptive* has two-fold meanings. Firstly, the algorithm adapts to the network latency among the grid nodes to make gamelet migration decisions. 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 to make gamelet migration decisions.

In AGL algorithm, there is a threshold δ_m used to judge whether a new load balancing process is necessary to be performed for server m . Its value can be adjusted based on the runtime stability of grid resources. Since a grid server may become unavailable due to some reasons, it might need to be removed from the DVE system at anytime. Under such a situation, δ_m can be set to about 0 so that server m will be considered as overloaded until all its gamelets are migrated to other grid servers. Then server m can be removed from the system.

We formulate the gamelet load balancing problem as a graph repartition problem. The graph is built as follows. For the i th gamelet G_i , create a vertex V_i in the graph. For any two vertices V_i and V_j , if there are some intercommunications $C_{i,j}$ between them, create an edge between them with value $W_{i,j} = C_{i,j}$. Other notations used are: $Val(G_i)$ represents the CPU load G_i introduces to the server, defined as a weighted packet sending rate; $Syn(G_i)$ is the synchronization cost of G_i ; $Cost(G_i, m)$ is the cost of migrating G_i to server m ; $Percentage(G_i)$ is the percentage of CPU load that G_i introduces to the server, $Percentage(G_i) \in [0, 1]$. The following lists the main procedure of the AGL algorithm.

Adaptive Gamelet Load Balancing Algorithm:

1. Select a server s which has the highest CPU load among the overloaded servers. Server s is considered to be overloaded if its CPU load is larger than δ_s , which is loaded from a configuration file at runtime.
2. Select a server t with the least CPU load as the migration target. Calculate the migration cost of each gamelet G_i in server s :

$$Cost(G_i, t) = Syn'(G_i) - Syn(G_i)$$

$Syn(G_i)$ is the pre-migration synchronization cost of gamelet i calculated by formula:

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

where ϕ is the set of gamelets that have some communication traffic with gamelet i , gamelet i runs in server m and gamelet j runs in server n . $Syn'(G_i)$ is the post-migration synchronization cost calculated by assuming that gamelet i has been migrated to server t .

3. Replace server t with a less loaded server. Repeat step 2 again until all existing gamelet servers have been evaluated.
4. Assign a gamelet with the smallest value of $Cost(G_i, q)$ from server s to server q . Calculate how much percentage of the workload gamelet i contributes to the original server. Estimate how much workload it will add to server q (this can be easily done by quantifying and comparing the hardware configurations of the two servers). The algorithm estimates the percentage of

workload introduced by gamelet i by this formula:

$$\text{Percentage}(G_i) = \text{Val}(G_i) / \sum_{G_s \in \psi} \text{Val}(G_s)$$

where ψ is a set of gamelets that are currently running in server s . If server q will be overloaded after migration, repeat step 4 again and try the gamelet with the second smallest value of $\text{Cost}(G_i, q)$.

5. Repeat step one to four until the estimated CPU load of the original server is under its threshold.
6. If there is still an overloaded server while all the other existing servers cannot afford to share workload, the algorithm will make a decision to discover a new grid node with certain requirements, e.g., a grid node with $> 10\text{Mbps}$ bandwidth and $< 200\text{ms}$ latency time, add it to the system as another server candidate. Wait for some time and go to step 1 again.

One strength of the algorithm is that the workload model is more accurate than other approaches that only take into account the clients number or density. Therefore, it makes the workload sharing more effective. The reason to use a weighted packet rate is that different participants will have different packet sending rate and different commands will lead to different workload too. The algorithm can also adapt to the network latency among the grid servers to make gamelet migration decisions. This is achieved through the use of a cost model that integrates both the gamelet synchronization cost and grid inter-server latency. All these factors enable the AGL algorithm work well in a grid environment.

3 Prototype Design and Implementation

There are four components in our prototype: client simulator, communicator, gamelet and monitor. The size of the 3D world is $100*100*20$. The world is partitioned into 16 equal-sized cells and the overlapping length of the neighboring partitions is 5. The client simulator simulates a number of randomly distributed clients. Each will send out a packet every 100ms. Each data packet has 32 bytes, consisting of the avatar's position, command and timestamp. We define a circle with a radius of 5 to be the area that a client is interested in. The gamelet also do collision detections after each command is executed.

We define the response time (RT) to be the average time interval of each client from sending out a packet to receiving the confirmation from the server that the command has been executed and the results have been sent to all the related clients. System capacity (CP) is defined as the maximum number of participants that the system can accommodate so that the interactions in the world will have a reasonable average RT (≤ 200 ms) and loss rate ($\leq 50\%$).

The gamelet and monitor are implemented on top of GT3.0 [14]. All components are implemented using J2SE 1.4.2 and run on Linux kernel 2.4.18 with P4 2.00GHz CPU, 512MB RAM and 100Mbps Ethernet [13]. The network round-trip time among the nodes is within several milliseconds if not specified explicitly.

4 Performance Results

4.1 Gamelet Creation and Migration

We study the performance of gamelet creation and migration when a monitor sequentially creates gamelets in different servers. We find that when the 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 initialed at both the monitor and gamelet server side. But the creation of the first gamelet in a second server takes 2-3 seconds. This is because the monitor has loaded and created necessary libraries. The creation of the second gamelet in the same server will only take about 100 ms.

The gamelet migration process will introduce a short time of delay. The delay time depends on the size of the gamelet, e.g., the total number of avatars that need to be transferred. The interactions between the monitor and the communicator add about 30-40 ms to the pure gamelet content transmission time. However, the migration process won't bring obvious influence to the packet loss rate, since the communicator will store the incoming packets temporarily and forward them later.

4.2 Gamelet Migration Algorithm

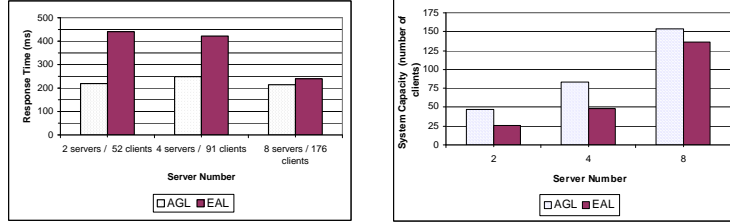
We compare the performance of our proposed AGL algorithm with a popular used algorithm named even-avatar load-balancing (EAL) algorithm, which tries to even the number of avatars that each server holds. Table 1 shows part of the network latency configurations among the servers in the experiments. The latency is below 1 ms within the same server.

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

Table 1. Network Latency Among the Servers

Initially, the 16 gamelets are all in server 1. As the number of clients increases, servers are added into the system one by one according to their indices. A server is regarded as overloaded if its CPU load is larger than 90%. In the experiment, the client simulator will generate a virtual environment with 3 hotspots.

Figure 2(a) shows the average RT under the two different approaches with 2, 4, and 8 servers respectively. The graph shows that the AGL algorithm can get much better performance. Table 2 shows the performance result under 4 servers. Under EAL approach, S1, S2 and S3 are all overloaded. Therefore, the RT of the three servers are very long leading to much worse performance on average. The influence of the inter-server latency is worthy of mention. E.g., under AGL approach, S2 has similar load with S1, but the RT of S2 is about 100 ms longer



(a)

(b)

Fig. 2. Gamelet Migration Algorithms Evaluations

than that of S1. This is because network connections with S2 is not as fast as that with S1.

91 Clients 4 Servers	CPU Load				Inter-server Traffic	RT (ms)				ART ms
	S1	S2	S3	S4		S1	S2	S3	S4	
AGL	90%	89%	79%	81%	276.8 Kbps	223	321	210	241	248.7
EAL	100%	100%	99%	42%	184.1 Kbps	503	572	512	101	422.0

Table 2. Performance Comparisons with 4 Servers

176 Clients 8 Servers	CPU Load								Inter-server Traffic
	S1	S2	S3	S4	S5	S6	S7	S8	
AGL	90%	72%	69%	71%	71%	90%	69%	68%	891.1 Kbps
EAL	90%	88%	89%	91%	89%	90%	35%	36%	742..6 Kbps

Table 3. Performance Comparisons with 8 Servers (1)

176 Clients 8 Servers	RT (ms)								ART (ms)
	S1	S2	S3	S4	S5	S6	S7	S8	
AGL	270	323	180	180	184	247	175	158	214.6
EAL	235	402	309	278	214	235	137	110	240.0

Table 4. Performance Comparisons with 8 Servers (2)

Table 3 and Table 4 show the CPU load and RT when 8 servers is used under the two approaches. This time the EAL approach happens to even the load among the servers (though it can not guarantee), and the inter-server traffic is even smaller than that of AGL algorithm. However, the average RT is still not as good as that of AGL algorithm. We find that the RT of S2, S3 and S4 is much worse under EAL approach. This is mainly because much of the inter-server traffic is transferred through low network, e.g., S2 and S3 under EAL approach. This

situation is avoided under AGL algorithm since two gamelets with large amount of traffic will tend to be assigned to two servers with better connections provided they can not be in the same server due to the CPU balancing requirement.

By decreasing the number of the clients in the virtual world, we can get the CP of the servers under the two approaches. The system with AGL algorithm has much larger capacity than the EAL algorithm. The reason lies in two aspects. Firstly, the AGL algorithm can estimate the CPU load each gamelet introduces to the server more accurate than the EAL algorithm. Secondly, AGL algorithm tries to minimize the intercommunications among the servers that have bad network connections. As shown in Fig. 2(b) the proposed AGL algorithm can improve the CP by 80%, 72% and 13% comparing with EAL approach when using 2, 4 and 8 servers respectively, therefore making a more scalable and cost-effective system.

5 Related Work

Most of the existing multi-server approaches are based on data partition scheme and can only perform limited local load sharing strategies. Examples includes CittaTron [4], NetEffect [5] and CyberWalk [6]. CittaTron is a multi-server networked game whose world is partitioned into several regions, each of which is assigned to one server. A load adjustment scheme is introduced by transferring some users from the highest loaded region to the neighboring regions. However, this approach can hardly be effective since only the user number is considered. In NetEffect, the virtual world is partitioned into separated communities that are managed by multiple servers. A heavy loaded server can transfer some of its communities to a less loaded server dynamically. Unfortunately, in such a system the load balancing process is not transparent to the clients. A user has to log into the system again after each migration. CyberWalk is a web-based multi-server distributed virtual walkthrough environment. The region managed by each server can be adjusted by transferring the boundary areas among neighboring regions. However, if there is more than one hotspots in the concerned regions, a cascading effect may occur which will seriously affect the system performance. Therefore, all the above load-sharing approaches are not suitable for a very dynamic DVE system where the participants can move and act freely. It is more difficult for these approaches to work well in a dynamic grid environment.

6 Conclusions

In this paper, we present a novel service component called gamelet for building DVE systems over Grid. Existing multi-server DVE systems based on partition scheme can be easily mapped into the gamelet-based framework. Our gamelet migration protocol and the load balancing algorithm help enable a more scalable DVE system. We have evaluated the performance of our proposed approach through a multi-player game prototype. Results show that our approach can build a more scalable system.

In our future work, we will study how various synchronization protocols will influence the performance of gamelet-based DVE system and what are the possible improvements. Currently, we assume a simple two-way synchronization scheme. However, more complicated synchronization protocol is needed for applications that have both high consistency and response time requirements. One possible direction is to study how the communicator can help gamelet do synchronization more efficiently.

References

1. S. Singhal and M. Zyda, *Networked Virtual Environments: Design and Implementation*, Addison Wesley, July 1999.
2. T. Funkhouser, "Network Topologies for Scalable Multi-User Virtual Environments", *Proceedings of the 1996 Virtual Reality Annual International Symposium, IEEE VRAIS 96*, San Jose, CA, USA, 1996, pp. 222-228.
3. J. Smed, T. Kaukoranta, and H. Hakonen, "A Review on Networking and Multiplayer Computer Games" Technical Report 454, Turku Centre for Computer Science, 2002.
4. M. Hori, K. Fujikawa, T. Iseri, and H. Miyahara, "CittaTron: a Multiple-server Networked Game with Load Adjustment Mechanism on the Internet", *Proceedings of the 2001 SCS Euromedia Conference*, Valencia Spain, 2001, pp.253-260.
5. Tapas K. Das, Gurminder Singh, Alex Mitchell, P. Senthil Kumar, and Kevin McGee, "NetEffect: a network architecture for large-scale multi-user virtual worlds", *Proceedings of the ACM symposium on Virtual reality software and technology*, Lausanne, Switzerland, 1997, pp.157-163.
6. N. Beatrice, S. Antonio, L. Rynson, and L. Frederick, "A Multiserver Architecture for Distributed Virtual Walkthrough", *Proceedings of ACM Symposium on Virtual Reality, Software and Technology 2002*, Hong-Kong, 2002.
7. I. Foster, C. Kesselman, J. Nick, 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, 2002.
8. California Inst. for Telecommunications & Infor. Tech. <http://www.calit2.net>
9. Butterfly Grid. <http://www.butterfly.net/>.
10. Beatrice Ng, Frederick W. B. Li, Rynson W. H. Lau, Antonio Si, and Angus M. K. Siu, "A performance study on multi-server DVE systems", *Information Sciences Informatics and Computer Science: An International Journal*, v. 154, n.1-2, 2003, pp. 85-93.
11. J.Y. Huang, Y.C. Du, and C.M. Wang, "Design of the Server Cluster to Support Avatar Migration, *IEEE Virtual Reality 2003 Conference*, LA, USA, 2003, pp.7-14.
12. S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, T. Maguire, T. Sandholm, P. Vanderbilt, and D. Snelling, *Open Grid Services Infrastructure (OGSI) Version 1.0*, Global Grid Forum Draft Recommendation, 2003.
13. HKU Gideon300 Grid. <http://www.srg.csis.hku/gideon/>, 2004.
14. Globus Toolkit 3.0. <http://www.globus.org/>.