Adaptive Partitioning for Multi-Server Distributed Virtual Environments

Rynson W.H. Lau[†,!] Beatrice Ng[†] Antonio Si[‡] Frederick Li[!]

[†] Department of Computer Science, City University of Hong Kong, Hong Kong
[!] Department of CEIT, City University of Hong Kong, Hong Kong
[‡] Oracle Corporation, 500 Oracle Parkway, Redwood Shores, CA 94065, U.S.A.

Abstract

A distributed virtual environment (DVE) allows users at different geographical locations to share information and interact within a common virtual environment (VE) via a local network or through the Internet. However, when the number of users exploring the VE increases, the server will quickly become the bottleneck. To enable good performance, we are currently developing a multi-server DVE prototype. In this paper, we describe an adaptive data partitioning technique to dynamically partition the whole VE into regions. All objects within each region will be managed by a single server. As the loading of the servers changes, we show how it can be redistributed while minimizing the communication cost. Our initial results show that the proposed adaptive partitioning technique significantly improves the performance of the overall system.

1 Introduction

In a distributed virtual environment (DVE), a user with access to the Internet may explore a place of interest without having to travel there. We have recently developed a client-server based DVE prototype system called CyberWalk [4, 3]. The server maintains the geometry database of the virtual environment (VE) and distributes virtual objects to individual clients on-demand, as viewers (or clients) navigate inside the VE. To optimize the rendering performance, CyberWalk models each object in multiresolution format and each object only needs to be rendered at a resolution just high enough for its viewing distance. To reduce latency and improve availability, CyberWalk proposes a multiresolution caching and prefetching mechanism to cache object models in the client machine and to prefetch objects from the server in advance. However, as the number of clients accessing CyberWalk increases, the performance of the system drops exponentially and the server will quickly become the bottleneck. To overcome this problem, we are currently extending CyberWalk into a multi-server system.

ACM Multimedia '02 Juan Les Pins, France

One popular approach to overcome this problem is to partition the VE into regions, and each of which is assigned to a separate server, distributing the workload among them. This may also prevent the single point of failure problem if clients can be dynamically connected to different servers. Systems adopting this approach includes RING [8], NetEffect [5] and CittaTron [9].

In RING [8], the VE is partitioned statically and each region is assigned to a fixed server. With this approach, a server may still be overloaded if a large number of clients converge to its region. On the other hand, a client may choose to connect to a server statically or dynamically based on its current position. In NetEffect [5], the VE is partitioned dynamically based on the client density of individual regions (communities), and it is possible for one server to handle multiple regions. A master server is responsible for performing the load balancing duty by transferring regions among the servers. In addition, a client may migrate to a different region of another server when its region is overcrowded. However, after the client is migrated to a new region, it needs to wait for downloading the scene of the new region. In CittaTron [9], the VE is partitioned into regions dynamically based on the server workload and the number of clients. Each region is assigned to a unique server. The size of each region may be adjusted during run-time and clients may be transferred among the servers in order to achieve load-balancing. However, factors such as object density and locality are not considered.

Existing multiplayer online games [2] have already implemented with distributed game servers. For example, Quake III Arena [11] and Diablo II [6] offer a list of game servers for clients to join. However, each game server maintains a unique game state, which is not shared among the servers. This is essentially a set of separated client-server systems running the same game and may not be considered as a real multi-server system. EverQuest [7], in contrast, divides the VE into distinct zones to be managed by individual game servers. A client may connect to any zone to join the game. EverQuest allows a client to travel from one zone (game server) to another freely. Ultima Online [12] and Asheron's Call [1] adopts a similar approach as EverQuest, but it may dynamically transfer a portion of the zone from one game server to another.

In general, the load balancing problem in DVE is more difficult than the traditional load balancing problem because more factors need to be addressed here:

- *Object density:* affects the average number of objects

visible to a viewer.

- *Viewer density:* affects the average number of viewers per region.
- *Viewing preference:* represents the viewers' interest on objects, i.e., affects the distribution of viewers in VE.
- *Spatial Coherence:* is a property of objects in the VE. If an object is visible to a viewer, it is likely that its neighbor objects are also visible to the viewer.

The first three factors affect the loading of each server. (Here, we do not consider how to model the viewers' interest.) The forth factor affects our partitioning algorithm.

In this paper, we propose an adaptive partitioning scheme that partitions the VE into multiple regions, each assigned to an individual server. However, the size of each region is dynamically adjusted according to the server loading due to processing the clients' requests and to the network traffics. The rest of the paper is organized as follows. Section 2 presents the multi-server architecture of CyberWalk. Section 3 discusses how we determine the server loading while section 4 discusses the dynamic repartitioning scheme. Finally, we show some initial results and briefly conclude our work in section 5.

2 A Multi-server Architecture

Our solution to the server loading problem is essentially a parallel architecture of multiple servers. As depicted in Figure 1, each server manages its database repository of virtual objects. In traditional parallel architecture, one of the servers is often dedicated to be a coordinator, having complete knowledge of the data managed in each of the other servers. It functions as an interface between a client and the server array in that the coordinator receives a query from a client and forwards it to the server which manages the data required by the query. However, the coordinator could quickly become a bottleneck when the rate of query submission is high. Another problem with this architecture is that any changes made by a client may have to go through many network communication steps before the client receives a reply, seriously affecting the interactivity of the system.

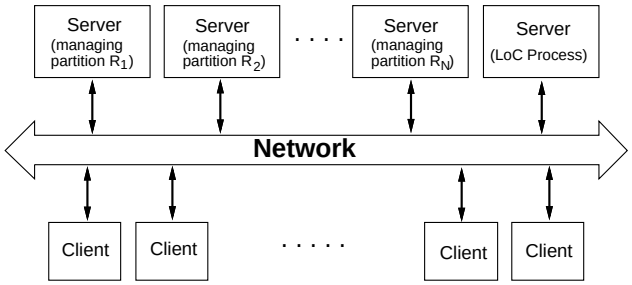

Figure 1: The multi-server CyberWalk.

Here, we partition the complete VE into an array of regions. Virtual objects within a region are managed by a single server only. Each server will be serving only those clients who are exploring objects within the region managed by the server. Hence, each client will be communicating directly to the server which manages the objects visible to the client. However, when a viewer walks near to the boundary of a region, it may be able to see objects located in the neighbor region. In this situation, both the current server and the neighbor server will need to serve the viewer; each will be responsible for distributing the objects that it manages. To achieve this, when the *viewer scope*, which defines the visible region of the viewer, touches the region boundary, the current server will send a message to the neighbor server informing it the information of the viewer, including the machine IP address and the viewer location in the VE. From now on, the client will talk to the neighbor server directly while maintaining the connection to the current server. When the viewer eventually moves into the neighbor region, it will stop communicating with the original server and will only communicate with the new server. An advantage of this architecture is that each server will effectively be serving for a considerable less number of clients.

In Figure 1, the LoC (Loading Collector) process is responsible for collecting the loading information of each server. Each server will periodically inform this process about its workload, and has the flexibility of determining how frequent it does it. If the workload of a server varies more frequently, it will inform the LoC more often. Otherwise, it could inform the LoC much less frequent. The LoC process, thus, has the most up-to-date loading information of each server. When a server is overloaded, it contacts the LoC process and requests for the loading information of neighbor servers. The overloaded server will then select the neighbor server with the lowest workload to absorb the newly partitioned region. It is important to note that the LoC process could be run in any one of the servers. When the server hosting the LoC process fails or is overloaded, a new server could be selected to host the LoC process.

In order to fully realize the advantage of this parallel architecture, we need to address two issues. First, we need to model the workload of a server and determine when a server is overloaded. Second, we need to partition the VE into regions in such a way that they can easily be repartitioned for load balancing. We will discuss these two issues in the following two sections.

3 Determining Workloads

When a server is overloaded, an appropriate server needs to be identified to share the workload. A mechanism to monitor and compare the workload of the servers is therefore needed. When CyberWalk is first started, the VE is partitioned into N regions based on object density and each server will be managing similar number of objects. A server, S_i, is associated with a *workload indicator*, ω_i, indicating its workload. When this workload indicator indicates that server S_i is overloaded, region R_i which is managed by S_i will need to be partitioned. A neighbor server S_j with the lowest workload indicator will be selected and the newly partitioned region will be allocated to S_j. This inherently directs all future viewer requests on the newly partitioned region to S_j and thus, reduces the workload of S_i.

The workload of a server is modeled by two factors: CPU loading (CPU_{server}) and network loading ($\mathsf{Network}_{server}$). CPU_{server} captures the processing overhead in identifying and retrieving the object models requested by the viewers and is modeled by the percentage of objects processed with respect to the maximum number of objects CPU_{MAX} that a server can process in one second:

$$\mathsf{CPU}_{server} = \frac{number\ of\ objects\ processed\ per\ second}{CPU_{MAX}}$$

$\mathsf{Network}_{server}$ captures the transmission overhead required to send the object information to the viewer and is modeled by the amount of object data sent to the viewer per second:

$$\mathsf{Network}_{server} = \frac{number\ of\ bytes\ sent\ per\ second}{effective\ network\ bandwidth}$$

where the effective network bandwidth is the actual network bandwidth allocated to the server.

When either the CPU loading factor or the network loading factor exceeds a predefined threshold, the server is considered as overloaded and its region will be partitioned.

The reason for S_i to offload its workload to only its neighbor servers instead of any arbitrary server is that when the viewer scope crosses the boundary of two servers, both servers will be responsible for transferring objects to the client. If a server offloads its newly partitioned region to any arbitrary server, a viewer will potentially be communicating with a large number of servers, increasing the communication overheads. This highlights the major difference between our problem and the traditional load balancing problem.

4 Adaptive Region Partitioning

Although we initially partition the VE into multiple regions based on object density, some regions may become more popular than others during runtime. For example, a region may contain more interesting objects and attract more visitors. This dynamically changing popularity characteristics of the VE may affect the workload distribution among the servers. A partitioning scheme which could result in uniform workload distribution among the servers is largely needed. We term our partitioning scheme, *adaptive region partitioning* as the partitioning of regions will be adapted to the workload among the various servers.

Initially, the complete VE is regularly subdivided into a large number of rectangular cells. Each cell, $c_{i,j}$, contains a set of objects, i.e., $c_{i,j} = \{o_{c_{i,j},1},\ o_{c_{i,j},2},\ \ldots\ o_{c_{i,j},|c_{i,j}|}\}$. The VE is also partitioned into a set of N *regions*, i.e., VE $= \{R_1,\ R_2,\ \ldots\ R_N\}$, with each region containing an integer number of *cells*, i.e., $R_i = \{c_{i,1},\ c_{i,2},\ \ldots\ c_{i,|R_i|}\}$. Since virtual objects are unlikely distributed uniformly within the VE, some cells of the VE may contain more objects than others. We partition the VE based on *object density*, such that each region contains approximately the same number of objects, i.e., $\sum_{k=1}^{|R_i|} \mid c_{i,k} \mid \approx \sum_{l=1}^{|R_j|} \mid c_{j,l} \mid\ \forall\ R_i, R_j \in$ VE. Hence, each region may contain different number of cells.

To simplify the calculation of CPU_{server} and $\mathsf{Network}_{server}$, we maintain in each cell two variables: CPU_{cell} and $\mathsf{Network}_{cell}$, indicating the CPU loading factor and the network loading factor caused by the cell. When we need to have an updated CPU loading factor of a region, for example, we may simply add up the CPU_{cell}'s of all the cells within the region. Since CPU_{cell} and $\mathsf{Network}_{cell}$ change much less frequent than CPU_{server} and $\mathsf{Network}_{server}$, this approach can significantly reduces the costs in computing the two loading factors.

When a server is overloaded, it will consider three factors:

- identifies a neighbor server with minimum workload,
- selects cells that are located at the boundary of these two regions for transfer, and
- minimizes the resulting perimeters of the two regions.

The first two factors should be obvious by now. The reason for the third factor is that if the perimeter of a region can be minimized, it also minimizes the average number of regions that the viewer scope overlaps when the viewer is near to the boundary of a region. In other words, the number of servers needed to serve the client can be minimized.

Our repartitioning procedures are as follows. First, when a server finds itself overloaded, it becomes the *overloaded server*. It then talks to the LoC process to obtain the loading information of its neighbor servers. At the same time, it checks a precomputed list for break points at its boundary. The cause of these breaks is due to the transfer of some cells to neighbor regions as shown in Figure 2. Each server maintains a list to indicate all the break points in its boundary. In the diagram, we assume that S_b is an overloaded server and R_b has two break points, break 1 and break 2. Once the loading information of the neighbor servers is available, S_b will check the workload of the corresponding neighbor server of the break points. If the workload of S_a is smaller, S_b will transfer the row of cells along the boundary starting from break point 1 to S_a, until either the workload of S_b drops below the threshold or the workload of S_a is too high to accept more cells. When a region has no break points, the server will start from a corner cell. It looks for a neighbor server with the lowest workload and starts "peeling" its boundary cells to the server.

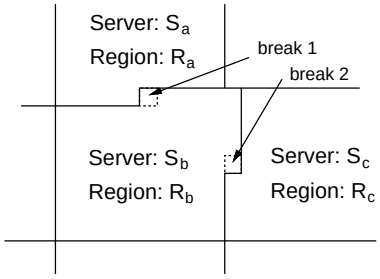

Figure 2: Repartitioning of a region.

There is a potential *flip-flop* problem with this method. When server S_b is overloaded and transfers some cells to server S_a, it may cause server S_a to get overloaded after a short period of time. If server S_a then selects S_b to offload some of its cells, S_b may become overloaded soon. To prevent this problem to occur, we ensure that server S_a will not select server S_b to offload its cells immediately after S_b has offloaded some cells to S_a.

5 Results and Conclusions

We have created a VE to test the new multi-server DVE architecture. It contains a total of 6,000 objects and is divided into 100×100 cells. We set the effective network bandwidth of the server to be 16Mbps while that of the clients to be 1.6Mbps. Both the server and the client programs run on Pentium III PC's with a 800MHz CPU. Figure 3 shows that as we increase the number of servers (and hence regions), the latency time decreases exponentially.

Figure 4 demonstrates how the number of clients accessing the VE and the number of objects in the VE affect the performance of the system. The latency increases as we increase the number of clients or the number of objects. This

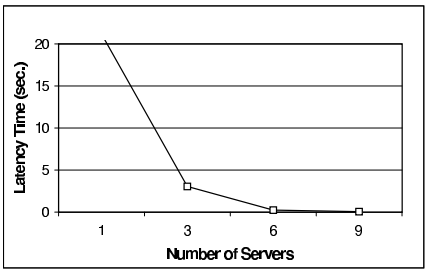

Figure 3: The effect of the number of servers on latency.

is because as we increase the number of viewers, more data requests will be generated. Likewise, as we increase the number of objects in the VE, more objects will be visible to a viewer, causing more information to be processed and transmitted. A more detailed performance analysis of the multi-server architecture can be found in [10].

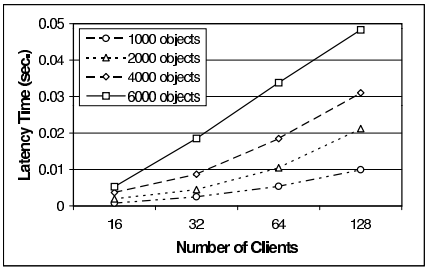

Figure 4: The effect of the number clients/objects on latency.

Figures 5 and 6 demonstrate how the shape of the regions changes as we introduce viewers into the VE. Figure 5 shows the initial partitioning of the VE into 9 regions managed by 9 servers. The partitioning is based on object density, with each grey dot representing an object. Hence, regions with high object density will have smaller size. Figure 6 shows the new partitioning of the VE after we have introduced a large number of viewers and let them walk around for a while. (Each black dot represents a viewer.) We can see that regions with a lot of viewers are significantly smaller in size now, for example, the center region. We can also see that the size of some regions becomes larger even though they also have some viewers inside, for example, the left middle region. This is because they have taken up some of the loading from their neighbor regions. As they are not overloaded yet, they do not distribute any of their own loads to other regions.

Overall, we have found that our adaptive region partitioning scheme is very effective in distributing the loading among all the servers while minimizing the communication costs. We are currently conducting more experiments on the scheme and implementing a full prototype system to compare the system performance in actual environment.

6 References

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

[2] T. Barron. *MultiPlayer Game Programming*. Premier Press, Inc., 2001.

[3] J. Chim, M. Green, R. Lau, H. Leong, and A. Si. On Caching and Prefetching of Virtual Objects in Distributed Virtual Environments. In *Proc. of ACM Multimedia*, Sept. 1998.

[4] J. Chim, R. Lau, H. Leong, and A. Si. CyberWalk: A Web-based Distributed Virtual Walkthrough Environment. *IEEE Trans. on Multimedia (to appear)*.

[5] T. Das, G. Singh, A. Mitchell, P. Kumar, and K. McGhee. NetEffect: A Network Architecture for Large-scale Multi-user Virtual World. In *Proc. of ACM VRST*, pages 157–163, 1997.

[6] Diablo II, Starcraft. Available at http://www.blizzard.com/.

[7] EverQuest. Available at http://everquest.station.sony.com/.

[8] T. Funkhouser. RING: A Client-Server System for Multi-User Virtual Environments. In *Proc. of Symp. on Interactive 3D Graphics*, pages 85–92, 1995.

[9] M. Hori, T. Iseri, K. Fujikawa, S. Shimojo, and H. Miyahara. Scalability Issues of Dynamic Space Management for Multiple-Server Networked Virtual Environments. In *Proc. of IEEE Pacific Rim Conf. on Communications, Computers and signal Processing*, pages 200–203, 2001.

[10] B. Ng, A. Si, R. Lau, and F. Li. A Multi-Server Architecture for Distributed Virtual Walkthrough. In *To appear in Proc. of ACM VRST*, Nov. 2002.

[11] Quake. Available at http://www.idsoftware.com/.

[12] Ultima Online. Available at http://www.uo.com/.

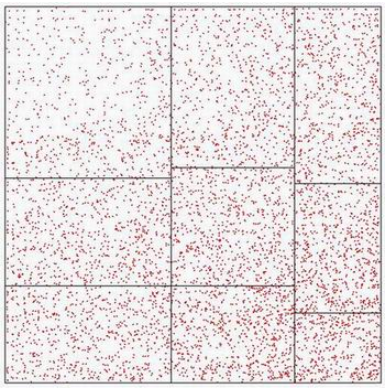

Figure 5: The initial partitioning of the VE based on object density.

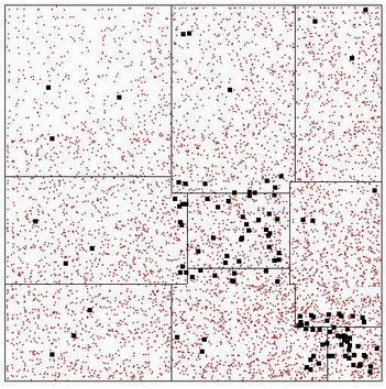

Figure 6: The partitioning of the VE when there is a large number of visitors.