A Multi-Server Architecture for Distributed Virtual Walkthrough

Beatrice Ng [†] Antonio Si [*] Rynson W.H. Lau [†‡] Frederick W.B. Li [‡]

[†] Department of Computer Science, City University of Hong Kong, Hong Kong
[*] Oracle Corporation, Redwood Shores, CA 94065, USA
[‡] Department of CEIT, City University of Hong Kong, Hong Kong

Abstract

CyberWalk is a distributed virtual walkthrough system that we have developed. It allows users at different geographical locations to share information and interact within a common virtual environment ($\mathcal{VE}$) via a local network or through the Internet. In this paper, we illustrate that when the number of users exploring the $\mathcal{VE}$ increases, the server will quickly become the bottleneck. To enable good performance, CyberWalk utilizes multiple servers and employs an adaptive data partitioning techniques to dynamically partition the whole $\mathcal{VE}$ into regions. All objects within each region will be managed by one server. Under normal circumstances, when a viewer is exploring a region, the server of that region will be responsible for serving all requests from the viewer. When a viewer is crossing the boundary of two or more regions, the servers of all the regions involved will be serving requests from the viewer since the viewer might be able to view objects within all those regions. We evaluate the performance of this multi-server architecture of CyberWalk via a detail simulation model.

Categories and Subject Descriptors

H.5.1 [**Information Interfaces And Presentation**]: Multimedia Information Systems –Artificial, augmented, and virtual realities

General Terms

Performance, Experimentation

Keywords

Multi-server architecture, data partition and replication, distributed virtual environments

1 Introduction

CyberWalk [6, 7] is a distributed virtual walkthrough prototype system that we have developed. It allows users at different geographical locations to share information and interact within a common virtual environment ($\mathcal{VE}$) via a local network or through the Internet. This $\mathcal{VE}$ may represent a physical museum, library or university. It may also represent a place to be constructed, such as a planned theme park. CyberWalk employs a standard client-server architecture. Information on virtual objects, including their locations and shapes, are maintained in a central database server. When a viewer (user) walks through a $\mathcal{VE}$, geometry information of the virtual objects located within a visible distance from the viewer will be conveyed to the viewer's client machine. The information will then be processed and rendered into images to be viewed. In general, the virtual objects could be dynamic as well, changing their locations and orientations within the $\mathcal{VE}$. However, since the current CyberWalk prototype supports only static objects, we focusn $\mathcal{VE}$'s with static objects only in this paper.

Our goal in this project is to provide good performance for a virtual walkthrough application such as CyberWalk, both in terms of responsiveness and resolution, under the existing constraints of relatively low Internet bandwidth and the large memory demand of virtual objects. In [6], we have introduced a multi-resolution caching mechanism which allows frequently accessed virtual objects to be cached at a client's local storage at a resolution requested by the client. The caching mechanism is also complemented with a prefetching mechanism which attempts to predict the movement of a viewer and transmits the objects to the client in advance. We have verified that the caching and prefetching mechanisms provide impressive improvement in system performance.

In this paper, we show that when the number of viewers exploring the $\mathcal{VE}$ increases, the server will quickly become the bottleneck through serving queries from the clients. CyberWalk addresses this problem by employing parallelism using an array of object servers. We propose an *adaptive data partitioning* technique to partition the whole $\mathcal{VE}$ into multiple regions. All objects within a region will be managed by one server. Requests from viewers for any object within a region will be served by the server managing that region. This reduces the number of viewers that each server needs to handle. When a viewer is crossing the boundary of two or more regions, all the servers of the relevant regions will be serving requests from the viewer since the viewer may be able to view objects within all those regions. When a server is overloaded by a large number of requests due to too many clients accessing its region, the region will be partitioned and part

ACM VRST'02, November 11–13, 2002, Hong Kong.

of it will be transferred to a lightly loaded neighbor server. We illustrate in this paper the performance improvement of CyberWalk using this parallel architecture.

The rest of the paper is organized as follows. Section 2 describes relevant research work. Section 3 introduces the parallel server architecture of CyberWalk. Section 4 presents the load balancing mechanism through the adaptive region partitioning scheme. Section 5 discusses the performance of CyberWalk via several simulated experiments. Finally, section 6 briefly concludes the paper.

2 Related Work

In this section, we introduce existing multi-server technology for distributed virtual environments. We also present an overview of our CyberWalk prototype system.

2.1 *Distributed Servers*

Traditionally, there are two types of architectural support for multiple clients to participate in a $\mathcal{VE}$, *peer-to-peer* and *client-server*. Earlier systems, such as NPSNET [12] and DIVE [3], are implemented in a peer-to-peer architecture. This approach has minimal communication overheads. However, it may not scale well to handle many simultaneous clients due to the saturation of network bandwidth in handling broadcast or multicast messages from the clients. To improve scalability, systems such as BrickNet [20], Community Place [17] and MASSIVE-3 [14], are implemented in a client-server architecture. With this approach, each client sends messages to the server for further propagation to other clients and/or other servers in the same $\mathcal{VE}$. The advantage of this approach is that the server may perform message filtering to minimize the amount of messages needed to be handled by each client and to be propagated through the network. The major limitation, however, is that as the number of clients accessing a $\mathcal{VE}$ increases, the amount of messages needed to be sent among them increases dramatically. The server loading in managing the $\mathcal{VE}$ and handling the messages also increases significantly. Another problem is that the server may potentially become a single point of failure.

A distributed $\mathcal{VE}$ system with a multi-server architecture could solve these problems. The $\mathcal{VE}$ may be partitioned 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 [13], NetEffect [8] and CittaTron [16].

In RING [13], the $\mathcal{VE}$ is partitioned statically and each region is assigned to a fixed server. With this approach, some servers may still be overloaded if a large number of clients converge to some particular regions of the $\mathcal{VE}$. A client in RING, however, may choose to connect to a server statically or dynamically based on its current position. In NetEffect [8], the $\mathcal{VE}$ is partitioned dynamically based on the client density of individual communities (regions), and it is possible for one server to handle multiple communities. A master server is responsible for performing the load balancing duty. It periodically checks the client density of all communities and perform load balancing by transferring communities among the servers. In addition, a client may migrate to a different community of another server when its community is overcrowded. However, after the client is migrated to a new community, it needs a waiting period to download the scene of the new community. In CittaTron [16], the $\mathcal{VE}$ 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 in this load-balancing mechanism.

Existing multiplayer online games [2] have already implemented with distributed game servers. For example, Quake III Arena [18] and Diablo II [10] 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 [11], in contrast, divides the entire $\mathcal{VE}$ into distinct zones, and maintains these zones 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 [21] and Asheron's Call [1] adopt similar approach with EverQuest, but they divide the entire $\mathcal{VE}$ into visually continuous zones. For instance, to avoid the lag problem for a client to travel across neighboring zones (game servers), they mirror the content along the zone boundary in the neighboring game servers. In addition, Asheron's Call is technically more advanced than Ultima Online in the way that it may dynamically transfer a portion of the zone controlled by a given game server based on the load of the server.

2.2 *Overview of* CyberWalk

CyberWalk [7, 6] is implemented based on the on-demand transmission concept. It has many advantages over existing systems: including minimal transmission cost through progressive model transmission and the introduction of viewer/object scopes, minimal rendering cost through multiresolution object modeling, and high interactivity and system availability through caching and prefetching:

- *Viewer Scope and Object Scope:* To minimize the amount of data needed to be handled, CyberWalk generalizes the Area-Of-Interest concept [12] to both viewers and objects, referred to as the *viewer scope* and the *object scope*. A viewer scope indicates how far the viewer can see. An object scope indicates how far an object can be seen. Its size is proportional to the size of the object. In general, a viewer may also be considered as an object and assigned with an object scope in addition to the viewer scope. An object can only be seen by a viewer when the two scope overlaps. Hence, objects which scopes do not overlap with the viewer scope do not need to be transferred to the client to save transmission, processing, and memory costs. In CyberWalk, we define a scope as a circular region characterized by a radius as shown in Figure 1.

- *Multiresolution Modeling for Progressive Transmission and Rendering:* Sending the complete models of only those visible objects to the client machine on-demand may still cause a possibly long pause in

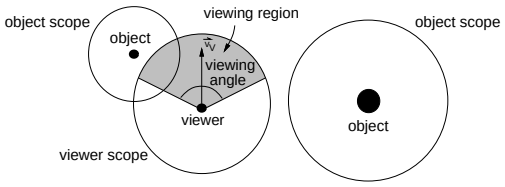

Figure 1: Object scope and viewer scope.

the walkthrough. Instead, our system encodes each object model as a *progressive mesh* [15]. During run-time, the distance between the centers of the viewer scope and of the object scope determines the resolution of the progressive mesh needed to be available in the client machine. Since the visibility of an object usually changes only slightly from frame to frame, only a small number of progressive records need to be transmitted to the client between consecutive frames. At the client, the object models can be progressively reconstructed and rendered from the partially transmitted progressive meshes.

- *Multi-resolution Caching:* In CyberWalk, a multi-resolution caching technique was developed to allow fine-granularity of object caching and replacement. A caching mechanism allows a client to utilize its memory and local storage to cache currently visible objects that are likely to be visible in the near future. A local cache also supports a certain degree of disconnected operation when the Internet is temporarily unavailable.

- *Prefetching:* A prefetching mechanism allows a client to predict objects that are likely to be visible in the future and obtain them in advance to improve response time. In CyberWalk, an EWMA-R (EWMA with residual adjustment) scheme was developed to predict the viewer's movement inside the $\mathcal{VE}$. Considering the fact that most users would be using a PC equipped with a 2D mouse for 3D navigation, we have recently developed a hybrid motion prediction method for predicting the viewer's future movement based on the motion pattern of the 2D mouse [4].

3 Parallel Architecture of CyberWalk

To study the impact of the number of clients on the performance of a centralized server, we have conducted a simulated experiment to measure the average *latency time* and *response time* of all viewers on a single server walkthrough system. Latency time measures the average time required to retrieve the base meshes of all visible objects, i.e., from the time the viewer makes a move to the moment when there is a coarse image of all visible objects. Response time measures the average time that the viewer needs to wait from the moment the viewer makes a move to the moment when all the visible objects are available at the client at their optimal resolutions. Latency time inherently captures the visual perception perceived by each viewer, while the response time measures the absolute performance of the system. The number of clients experimented ranges from 2, 4, 8, 16, 32, 64, and 128.

The size of the $\mathcal{VE}$ was set to 100×100 square units (or *cells*). 6,000 virtual objects are uniformly distributed among the cells. The radius of each object scope is randomly generated, ranging from 0.7% to 1% of the size of the $\mathcal{VE}$, with a mean of 0.85%. Each object is modeled by a progressive mesh. The number of progressive records associated with each object model ranges from 4500 to 7000 with a mean of 5,300 records. Each progressive record has a size of 40 bytes while each base mesh has a size of 3KB. The database is approximately 1.2GB.

The viewer's viewing angle is set at 120 degrees. The radius of the viewer scope is also generated in the same way as that of the object scope. We assume that the network bandwidths of the server and of each client are 20 Mbps and 2Mbps, respectively. The client movement in the $\mathcal{VE}$ is determined by sampling the movement of a real user interacting with CyberWalk using a mouse [4]. It is modeled as follows. The viewer moves circularly starting and ending at the same location. Each movement step includes a translation of 15 units along the viewing direction, followed by a rotation of the viewing direction of 12 degrees. At each step, the viewer rotates his/her head by ±20 degrees. In addition, the moving direction changes with an angle of 10 degrees, after every 4 movement steps. This is termed changing circular moving pattern (CCP).

Both caching and prefetching mechanisms are activated in each client. Since in our previous study [6], we have found that 1% cache size is able to achieve a 86% of cache hit, the size of the storage cache of each client is set to 1% of that of the database, i.e., 12MB. Both the server and the client programs run on PCs with a Pentium III 800 MHz CPU and 512 MB RAM. The result of the experiment is depicted in Figure 2. Figures 2(a) and 2(b) depict the overhead of the server when the number of clients ranges from 2 to 128. Since the overhead of the server when the number of clients is less than 16 is not noticeable in these two figures, Figures 2(c) and 2(d) show an enlarged view into this region. From the figure, the overhead of the server observed by each client increases super-linearly as the number of clients increases. For instance, the latency time for 128 clients is as high as 21 seconds. This is very non-favorable in a virtual walkthrough situation. Assuming that a 0.2 second latency time is the maximum threshold that a client is able to tolerate without experiencing a substantial low visual perception, we observe from the figure that all clients will experience a low visual perception when there are more than 10 clients.

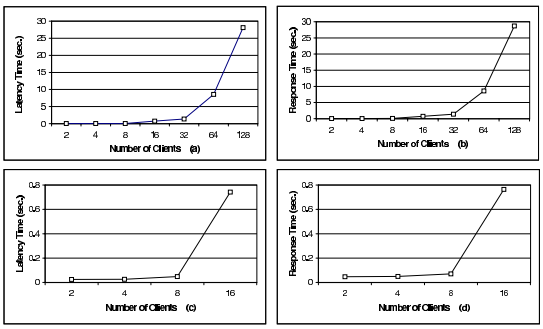

Figure 2: The effect of number of clients on the performance of the server.

To address this problem, CyberWalk employs a parallel architecture of multiple servers. A parallel architecture is fundamentally a share-nothing architecture [9]. As illustrated in Figure 3, each server manages its own

memory and database repository of virtual objects. In traditional share-nothing architecture, one of the servers is often dedicated to be a coordinator, which has complete knowledge of the data managed in each of the other servers. The coordinator receives a query from a client and forwards the query 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 becomes high. There is another problem with such architecture which is regarded as very important in our application. Any changes made by a client may have to go through many network communication steps before the client receives a reply. This extra delay can seriously affect the interactivity of the system, especially if the servers are not physically located in the same LAN.

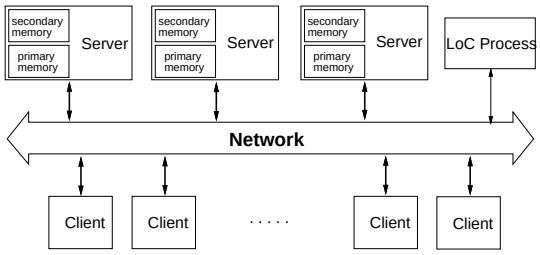

Figure 3: Parallel architecture of CyberWalk.

In CyberWalk, we partition the complete $\mathcal{VE}$ into a two-dimensional array of regions. Virtual objects within a region will be 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. Under normal circumstances, when a viewer V is exploring a region R_1, the server of region R_1, S_1, will be responsible for serving all requests from V. When V is crossing the boundary from R_1 to R_2, i.e, when the viewer scope of V, $\bigcirc_V$, touches the boundary of R_2, the servers of all the regions involved will be serving requests from the viewer since the viewer might be able to view objects within all those regions. In other words, both servers S_1 and S_2 will be serving requests from V. To achieve this, when $\bigcirc_V$ touches the boundary of R_2, server S_1 will send a message to server S_2 of the format $\langle id_V, loc_V, \vec{v}_V \rangle$, where id_V is the ID of V, which is inherently the IP address of the client, loc_V is the location of V, and $\vec{v}_V$ is the viewing direction of V. Once server S_2 has received such a message from server S_1, it can determine the objects in R_2 visible to V and transmits the corresponding progressive records and/or base meshes directly to V. V will then maintain direct communication channels with both S_1 and S_2 as long as $\bigcirc_V$ overlaps with both regions. When V eventually moves into region R_2, V will stop communicating with server S_1 and will only communicate with server S_2.

An advantage of this architecture is that each server will effectively be serving for a considerable less number of clients. When the number of clients inside a region has substantially increased to the extent which affects the performance perceived by a viewer, that region may be further partitioned and the newly partitioned region may be allocated to a less busy neighbor server.

As depicted in Figure 3, one process, referred as the LoC (Loading Collector) process, is dedicated to collect the loading information of each server. Each server will periodically inform the LoC process about its workload, and has the flexibility of determining how often it will inform the LoC process about its workload. If the workload of a server varies more frequent, 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. Approaches to determine such duration has been studied in [5] and is thus beyond the scope of this paper. 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 a neighbor server with the lowest workload to absorb the newly partitioned region.

The LOC process could also be piggybacked as a point of contact for newly-jointed client to discover to which server the client needs to contact in order to explore the relevant part of the $\mathcal{VE}$. Note that the LoC process could be hosted in any one of the servers. When the server hosting the LoC process fails or 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, a mechanism to monitor the workload of the servers is needed in order to identify the appropriate server to share the workload when a server is overloaded. On the one hand, we would like to transfer as much workload to the target server as possible so that the overloaded server could continue to serve new requests effectively and efficiently. On the other hand, we do not want the target server to get overloaded soon after it has received the new workload.

Second, we need to properly partition the $\mathcal{VE}$ into regions. Since the virtual objects may not be uniformly distributed among all regions, some servers will need to manage more objects than others if the $\mathcal{VE}$ is simply partitioned in a uniform manner. This will also cause some servers to experience a heavier workload than others. In addition, some regions may contain more interesting objects and attract more visitors. This skewed popularity characteristic of virtual objects may further 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.

4 Adaptive Load Balancing

In CyberWalk, the complete $\mathcal{VE}$ 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 $\mathcal{VE}$ is also partitioned into a set of N_R *regions*, i.e., $\mathcal{VE} = \{R_1, R_2, \ldots R_{N_R}\}$, while each region contains an integer number of *cells*, i.e., $R_i = \{c_{i,1}, c_{i,2}, \ldots c_{i,|R_i|}\}$. Figure 4 depicts a partition of 9 regions, each containing 9 cells and managed by one server.

There are several ways to partition the $\mathcal{VE}$ into regions. The simpliest way is to evenly partition it into regions, referred as the even scheme. Each region will cover the same geographical size of the $\mathcal{VE}$ and contain the same number of cells, i.e., $\mid R_i \mid = \mid R_j \mid \ \forall R_i, R_j \in \mathcal{VE}$. However, since the virtual objects might not be distributed uniformly within the $\mathcal{VE}$, some regions may contain more objects than others. This could be addressed by partitioning the $\mathcal{VE}$ based on "object density", referred as the density scheme. By "object density", we refer to the number of

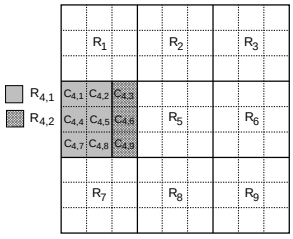

Figure 4: Region partitioning.

objects per cell of the $\mathcal{VE}$. In this scheme, each region will contain approximately the same number of objects, i.e., $\sum_{k=1}^{|R_i|} \mid c_{i,k} \mid \approx \sum_{l=1}^{|R_j|} \mid c_{j,l} \mid \forall R_i, R_j \in \mathcal{VE}$. However, each region may cover a different geographical size of the $\mathcal{VE}$, and thus may contain different number of cells.

Notice that when the virtual objects are distributed uniformly within the $\mathcal{VE}$, the density scheme and the even scheme will result in a similar partition. The density scheme attempts to achieve a uniform workload among all servers by ensuring that each server will manage the same number of objects. This is based on an assumption that each object has a similar degree of interest to the viewers and thus, a similar probability of being accessed. In practice, however, viewers may show particular interests in certain objects and explore certain regions more frequent than others. Hence, the density scheme may not necessarily result in a uniform workload among the servers. We address this issue by our *adaptive region partitioning* scheme, in which the partitioning of the regions will be adapting to the workload among various servers.

4.1 *The Adaptive Region Partitioning Scheme*

Initially, when CyberWalk is first started, there is no information regarding to the workload of each server. Therefore, the $\mathcal{VE}$ is partitioned into N_R regions based on either the even or the density scheme. Each server S_i is associated with a *workload indicator* ω_i indicating the workload of the server. When ω_i indicates that server S_i is overloaded, region R_i will be further partitioned. A neighbor server of S_i with the lowest workload indicator will be selected and the newly partitioned region will be allocated to this neighbor server. This inherently directs all future viewer requests on the newly partitioned region to the neighbor server and thus, reduces the workload of S_i.

To determine if a region needs to be partitioned, each server S_i will continuously monitor its workload indicator by maintaining two monitor windows called the short duration window $W_S(S_i)$ and long duration window $W_L(S_i)$, with a window size of $\mid W_S(S_i) \mid$ and $\mid W_L(S_i) \mid$, respectively, where $\mid W_S(S_i) \mid < \mid W_L(S_i) \mid$. (Hereafter, we will leave out S_i when the context is clear.) The short duration window monitors the workload of S_i within a very short duration of time. The purpose is to detect sudden bursts of workload on the server either due to network congestion or a sudden increase in interest on region R_i. By contrast, the long duration window monitors the workload of S_i within a much longer duration of time. The purpose is to detect a continuous high workload on the server.

We model the workload of a server by two factors: CPU loading factor and network loading factor. The CPU loading factor captures the processing overhead required to identify and retrieve the object models requested by the viewers and is modeled by the server utilization, which is the percentage of objects processed in one second i.e., $\frac{\#\ of\ objects\ processed\ per\ second}{M_{S_i}}$, where M_{S_i}, is the maximum number of objects a server can process in one second. This information inherently captures the CPU overhead of the server. The network loading factor captures the transmission overhead required to transmit the progressive records to the viewers and is modeled by the amount of object data sent to the user in a second, i.e., $\frac{\#\ of\ Bytes\ sent}{(network\ bandwidth * r_i)}$, where r_i is the ratio of bandwidth allocated to server S_i. From the formulae, each server S_i will monitor the number of bytes sent through the network and determine the duration to transmit the data. The term "network bandwidth * r_i" is used to model the effective bandwidth since in practice, a server is not likely to be able to utilize the full bandwidth of the network.

To determine if a region needs to be further partitioned, each server maintains two *partitioning threshold*s, τ_{CPU} and $\tau_{network}$. When the CPU or the network loading factor monitored by any one of the windows exceeds τ_{CPU} or $\tau_{network}$, respectively, the region will be further partitioned. These two thresholds could be determined dynamically based on the workload of the whole system or be fixed. In the former case, when the system is overloaded, the performance of the whole system will be gracefully degraded as the thresholds adjusted their values higher. In the latter case, the whole system cannot continue to operate as it is thrashed trying to partition every region. A detail analysis of how to determine the thresholds is, however, beyond the scope of this paper.

To partition a region R_i to offload some of its workload, the following steps are involved:

1. When server S_i is overloaded, it contacts the CP for the workload information of its neighbor servers. By neighbor servers, we refer to those servers managing regions that are immediate neighbors of R_i.

2. When S_i receives the workload information of its neighbor servers, it selects the one with the lowest workload as the target server S_T to offload some of its workload. This load transfer is achieved by partitioning R_i and allocating the newly partitioned subregions to S_T.

3. S_i will determine the *target workload*, the amount of workload needed to be transferred to S_T. In our prototype, each server will reduce its workload to 10% below the threshold value.

4. The partitioning is performed on a per cell basis. Each server will maintain a workload indicator for each cell. The partitioning of a region R_i is achieved by de-allocating one or more of its boundary cells, $c_{i,j}$, from R_i and transferring them to the target server, i.e., $R_i = R_i$ - $\{c_{i,j}\}$ and $R_T = R_T \cup \{c_{i,j}\}$. By boundary cells, we refer to those cells at the boundary between region R_i and the region managed by the target server. For example, cells $c_{4,3}$, $c_{4,6}$, and $c_{4,9}$ in Figure 4 are the boundary cells of R_4 to be merged with R_5.

5. Since there will most likely be more than one boundary cell, the one with the workload indicator closest

to the target workload will be selected first and allocated to the neighbor region. For example, assuming that cell $c_{4,9}$ in Figure 4 has the highest workload among the three boundary cells, $c_{4,3}$, $c_{4,6}$, and $c_{4,9}$, it will then be the first to be allocated to region R_5.

The reason for S_i to offload its workload to only its neighbor servers instead of any arbitrary servers is that when the viewer scope crosses the boundary of two servers, both server will be responsible for transferring object models to the viewer. If a server offload its newly partitioned sub-region to any server, a viewer will potentially be communicating with a large number of servers, significantly increasing the communication overheads. In Figure 4, if server S_4 offloads cells $c_{4,3}$, $c_{4,6}$, and $c_{4,9}$ at three different load balancing moments to arbitrary servers, these three cells may potentially be offloaded to three different servers. When a viewer whose viewer scope crosses the boundary between region $R_{4,1}$ and region $R_{4,2}$, the viewer will have to communicate with multiple servers in order to receive object models within the three cells. By contrast, when offloading is performed only to the neighbor servers, all these three cells will be offloaded to server S_5. The viewer will just have to communicate with two servers when its viewer scope crosses the boundary.

4.2 *Subregion Allocation*

When a server is overloaded, we transfer its boundary cells to the same target server until the workload of the overloaded server falls below the target workload. In Figure 4, if the transfer of cell $c_{4,9}$ to R_5 cannot reduce the workload of region $R_{4,1}$ to fall below the target workload, additional boundary cells, eg., cells $c_{4,3}$, and $c_{4,6}$, will be also transferred. A potential problem called, the *load balancing cascading effect* may happen. Once a neighbor server is identified as the target server, all boundary cells are allocated to it. This might cause the target server to overload and the target server will in turn, initiate the load balancing process again.

When the cascading effect does happen while server S_i offloads a newly partitioned subregion to a target server S_T, it is possible that server S_T will identify server S_i as the target server, causing a flip-flop effect. We therefore need to ensure that server S_T will not select server S_i as its target server immediately after S_i has offloaded a partition to S_T.

5 Results and Discussions

We have developed a simulation model and conducted experimental studies to quantify and evaluate the performance of the parallel architecture of CyberWalk. The parameters used in our experiments are listed in Table 1. Our experimental settings are similar to those described in Section 3 and we will only describe the settings that are specific to the parallel architecture here. In our simulation model, there are n virtual objects and N_V clients. The modeling of each virtual object has been described in Section 3. The experimental conditions are essentially the same. The $\mathcal{VE}$ is regularly subdivided into 100×100 cells. The n virtual objects are distributed among the cells. The complete $\mathcal{VE}$ is partitioned into N_R regions, with $N_R << 100\times100$. Since each region is managed by one server, N_R servers will be required.

We model two object distribution patterns, D_O, in our experiments: uniform and skew. In uniform distribution, the objects are distributed uniformly among all regions. Hence, each cell contains a fixed number of $\frac{n}{10000}$ objects. In skew distribution, each region will contain a random percentage of objects, ranging from 30% to 200% of the mean value. We study three different region partitioning schemes, P_R: even, density, and adaptive. In even and density, each region will not be further partitioned into subregions regardless of the workload of the server. In adaptive, a region might be further partitioned into subregions using the adaptive partitioning scheme.

To monitor the workload, the window size of the short duration window of each server is set to 1 second and that of the long duration window of each server to 5 seconds in our experiments. Parameter r_i is set to 0.8. Since the maximum network bandwidth of each server is 20 Mbps, with r_i set to 0.8, the effective network bandwidth becomes 16 Mbps. In addition, M_{S_i} is set to 1.4×10^6, which is the maximum number of objects that the server can handle in a second and is obtained through experiments. Since the CPU loading factor measures the utilization of the server, we decide to partition a region when the server is fully utilized to reduce the frequency of partitioning. Hence, threshold τ_{CPU} is set to 1 in our experiments. Similarly, the network partition threshold $\tau_{network}$ is also set to 1 second. Due to space limitation, we will not study the effect of these parameters in this evaluation.

Each viewer will be residing at a random position within the $\mathcal{VE}$ when the simulation starts, and move within the $\mathcal{VE}$ according to the changing circular pattern (CCP) described in section 3. Our simulation program ensures that the path of each viewer is different.

We measure *latency time* and *response time*, as defined in section 3, experienced by each client. In each experiment, each metric is determined by averaging the metrics obtained from all movement steps of all clients. The standard deviations are found to be small. Due to space limitation, we only present four experiments to quantify the performance of the parallel architecture of CyberWalk. The parameter settings are summarized in Table 1.

Para.	Exp. #1	Exp. #2	Exp. #3	Exp. #4
n	6K	1K, 2K, 4K, 6K	6K	6K
N_V	128	16, 32, 64, 128	16, 32, 64, 128	128
N_R	1, 3, 6, 9	9	9	9
P_R	even	even	even, density	density, adaptive
D_O	uniform	uniform	skew	skew

Table 1: Parameter values for the experiments.

5.1 *Experiment #1*

In the first experiment, we study the effect of number of servers on the performance of CyberWalk. In this experiment, n is fixed at 6,000 objects and the number of clients at 128. The region partitioning scheme is fixed at even. The object distribution pattern is fixed at uniform. The number of servers, N_R, ranges from 1, 3, 6, and 9. The measurements of the two metrics are depicted in Figurse 5(a) and 5(b). Since the metrics values for 6 and 9 servers are too small to be noticeable, we zoom in the scale for these two data points in Figures 5(c) and 5(d).

From the diagram, it is obvious that both latency time and response time decrease exponentially with the number of servers. Using the same 0.2 second latency time as threshold for acceptable visual perception, we can observe

from the diagram that 9 (or more than 6) servers are sufficient to serve for a population of 128 clients.

5.2 *Experiment #2*

In the second experiment, we study the effect of number of objects on the performance of CyberWalk. In this experiment, n ranges from 1,000, 2,000, 4,000, and 6,000 objects. The number of clients ranges from 16, 32, 64, and 128. The number of servers is fixed at 9. The region partitioning scheme is still fixed at even, and the object distribution pattern is fixed at uniform. The measurements of the two metrics are depicted in Figure 6. The first row shows the metrics versus the number of objects, while the second row shows the metrics versus the number of clients. We can see that both latency time and response time increase with the number of objects. It is very encouraging that the latency time all fall below the 0.2 second performance threshold, meaning that users will be able to experience a good visual perception on the system even with a high population of clients and objects.

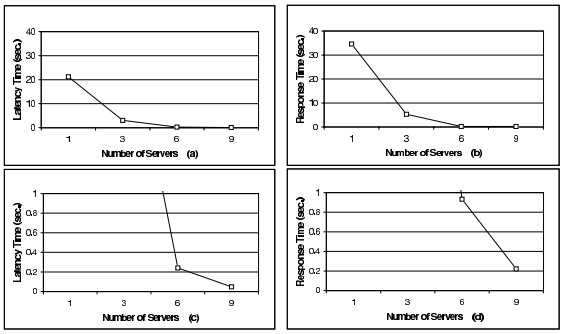

Figure 5: Performance of Experiment #1.

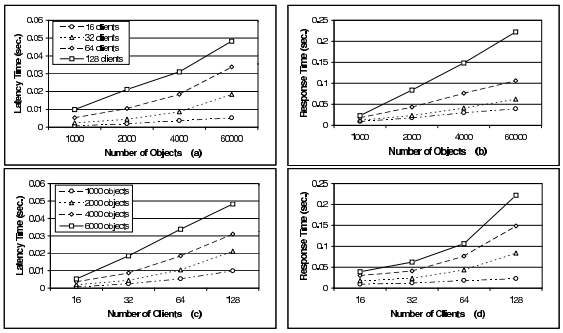

Figure 6: Performance of Experiment #2.

When comparing Figures 6(b) and 6(d), we observe that the number of objects does not have as big effect on the performance as the number of clients. This is because the amount of data transmitted depends on the degree of details a viewer views an object. In a typical walkthrough, only a few nearby objects are really rendered at high detail at any time. Increasing the number of objects will only moderately increase the amount of data needed to be transmitted. In contrast, increasing the number of clients will contribute to a greater amount of data transmitted from the server since each client will need to request for its own copy of all the visible object models.

5.3 *Experiment #3*

In the third experiment, we compare the performance between even and density region partitioning schemes. In this experiment, n is fixed at 6,000 objects. The number of clients will also range from 16, 32, 64, and 128. The number of servers is again fixed at 9. Since even and density partitioning schemes differ only when the objects are not distributed uniformly across the regions, we will only look at the skew object distribution pattern here.

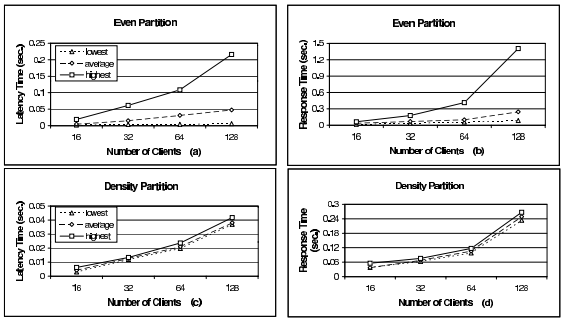

Figure 7: Performance of Experiment #3.

In Figure 7, we show three sets of data in each graph, depicting the lowest overhead, the average overhead, and the highest overhead among the 9 servers. We observe that in even partitioning, there is a big difference in performance among the servers. This indicates that some servers might be overloaded, while others might be under utilized. This is because in even partitioning, the $\mathcal{VE}$ is divided into equal-sized regions and hence each server will need to handle different number of objects, resulting in a non-uniform workload among the servers. Furthermore, we observe that when there are 128 clients, the latency time experienced by the clients will be higher than the 0.2 second performance threshold. In contrast, in density partitioning, the difference in overhead among the servers is minimal. This indicates that density partitioning is able to evenly distribute the workload among the servers. Furthermore, all clients will experience a good visual perception as the latency time of all clients are under the 0.2 second performance threshold.

5.4 *Experiment #4*

In Experiment #3, we have indicated that by partitioning the regions based on object density, all servers will have relatively uniform workload and thus, similar performance. This is based on the assumption that each viewer has a similar degree of interest in each object. However, in practice, viewers usually have different degrees of interest in different objects. Hence, a region may have a higher access probability if the viewers have a high interest in the objects within it, and a uniform object density among various regions does not necessarily result in a uniform workload among the servers.

In the fourth experiment, we study the effect of the adaptive region partitioning scheme in balancing the workload among the servers. We fix the number of clients to 128 and the number of objects to 6,000. We only look at the skew object distribution pattern in this experiment, as the even object distribution pattern has a similar behavior. For comparison purpose, we show results of both density and adaptive region partitioning schemes since the density scheme is shown to be able to achieve certain degree of uniform workload among the servers in Experiment #3. To model different degrees of interests among different objects, the center region is dedicated to

be the hottest region in which V% of the viewers will visit.

To collect the loading information, each server notifies the LoC process of its workload indicator every 1 second in our experiment. We measure the performance of the hottest region by varying V from 10%, 20%, 40%, 60%, and 80%. Figure 8 shows the overhead experienced by the center server, i.e., the hottest server, as well as the average overhead of its neighbor servers. We can see that the overhead increases with the number of visiting clients. With the density partitioning scheme, the center server experiences a much higher overhead than the neighbor servers. It is obvious from the figure that the neighbor servers are under utilized, while the center server is overloaded. Using the 0.2 second latency time as the threshold for acceptable visual perception, we observe that the clients will start experiencing a poor visual perception when more than 40% of the clients are visiting the center region. By contrast, with the adaptive partitioning scheme, we observe that the overhead experienced by the center server is similar to that experienced by the neighbor servers. This indicates that the center server could offload its workload to its neighbor servers properly. Hence, each server is able to support a much larger percentage of clients. Even with 80% of the clients visiting the center region, the latency time of the center server is only just above the 0.2 second threshold.

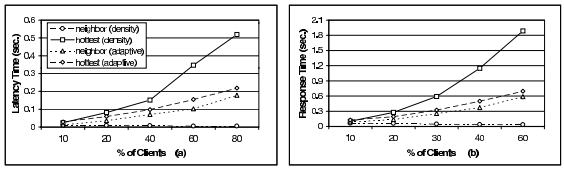

Figure 8: Performance of Experiment #4.

6 Conclusion

In this paper, we have described a parallel architecture to support virtual walkthrough and illustrated its implementation in our CyberWalk prototype. We have pointed out that in traditional client/server architecture, a single server has limitation in serving multiple clients. As one alternative to improving the performance, we propose a parallel architecture and employ multiple servers in serving the clients. We have discussed several ways of partitioning the virtual environment into multiple regions and studied their behaviors under various object distribution patterns. We have also introduced the adaptive region partitioning scheme to address the problem of non-uniform access among the regions. This scheme, in effect, partitions the virtual environment based on the degrees of interests that the viewers have on the regions. We have studied our parallel architecture via simulation. Our studies show that the adaptive region partitioning scheme is very effective in maintaining a uniform workload among the servers. Although we have not conducted experiments on other moving patterns as well, we believe the CCP moving pattern studied in this paper is able to present an overall performance behavior of our approach.

Acknowledgments

The work described in this paper was partially supported by a SRG grant from City University of Hong Kong (Project Number: 7001391).

7 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] C. Carlsson and O. Hagsand. DIVE - a Multi-User Virtual Reality System. In *Proc. of IEEE VRAIS*, pages 394–400, 1993.

[4] A. Chan, R.W.H. Lau, and B. Ng. A Hybrid Motion Prediction Method for Caching and Prefetching in Distributed Virtual Environments. In *Proc. of ACM VRST*, pages 135–142, Nov. 2001.

[5] B. Chan, A. Si, and H.V. Leong. A Framework for Cache Management for Mobile Databases: Design and Evaluation. *Journal of Distributed and Parallel Databases*, **10**(1):23–57, 2001.

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

[7] J. Chim, R.W.H. Lau, A. Si, H.V. Leong, D. To, M. Green, and M. Lam. Multi-Resolution Model Transmission in Distributed Virtual Environments. In *Proc. of ACM VRST*, pages 25–34, Nov. 1998.

[8] 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.

[9] D. DeWitt, S. Ghandeharizadeh, and D. Schneider. A Performance Evaluation of the Gamma Database Machine. In *Proc. of ACM SIGMOD*, 1988.

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

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

[12] J. Falby, M. Zyda, D. Pratt, and R. Mackey. NPSNET: Hierarchical Data Structures for Real-Time Three-Dimensional Visual Simulation. *Computers & Graphics*, **17**(1):65–69, 1993.

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

[14] C. Greenhalgh, J. Purbrick, and D. Snowdon. Inside MASSIVE-3: Flexible Support for Data Consistency and World Structuring. In *Proc. of Int'l Conf. on Collaborative Virtual Environments*, pages 119–127, 2000.

[15] H. Hoppe. Progressive Meshes. In *Proc. of ACM SIGGRAPH '96*, pages 99–108, Aug. 1996.

[16] 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.

[17] R. Lea, Y. Honda, K. Matsuda, and S. Matsuda. Community Place: Architecture and Performance. In *Proc. of VRML '97*, pages 4l–50, Feb. 1997.

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

[19] G. Singh, L. Serra, W. Png, and H. Ng. BrickNet: A Software Toolkit for Network-Based Virtual Worlds. *Presence: Teleoperators and Virtual Environments*, **3**(1):19–34, 1994.

[20] G. Singh, L. Serra, W. Png, A. Wong, and H. Ng. BrickNet: Sharing Object Behaviors on the Net. In *Proc. of IEEE VRAIS*, pages 19–27, 1995.

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