Issues in a Very Large Scale Distributed Virtual Environment

SO, King-yan Oldfield

A Thesis Submitted in Partial Fulfilment
of the Requirements for the Degree of
Master of Philosophy
in
Computer Science & Engineering

Supervised by:
Prof. John C.S. LUI

大型分佈式虛擬環境的研究項目

蘇敬恩

論文摘要

隨著多媒體的研究和網絡技術的發展，大型分佈式虛擬環境將得到實現。在大型分佈式虛擬環境中，數以千萬計身處於不同地方的用戶，可以透過計算機網絡來在一個同共同的虛擬環境中遊覽、交換資訊和溝通。

當我們要在局域網和廣域網中建立大型分佈式虛擬環境時，我們需要一個能同時應付極大量的用戶，又能處理在整個網絡當中不同結點在通訊上延遲的不同的系統建構。就此，我們把這個問題制定成爲一個劃分問題，並爲此建議了幾個不同的算法來解答這個劃分問題，以致當我們把整個大型分佈式虛擬環境的工作量分配到不同計算機網絡結點時，我們亦能同時減低因爲這個分配所引起對計算機網絡帶寬的額外需求。

理想來說，在大型分佈式虛擬環境中，任何一個用戶在虛擬環境中的一舉一動，都應該立即被傳送到其他的用戶的計算機中。換言之，我們要爲所有用戶提供一個一致的虛擬環境。要保持不同用戶的虛擬環境的一致性，我們需要做透過計算機網絡來做一些同步化的工作。我們就此建議了一些子圖建立的算法，使我們可以透過組播來更有較的在計算機網絡中傳送同步化的資料。我們亦提供了一些同步化的機制來達致不同用戶的虛擬環境的一致性。最後，我們定義了何謂虛擬環境的一致性，並根據這定義和一些系統的參數，用不同的方法演算出該系統的最佳同步化間距。

Issues in a Very Large Scale Distributed Virtual Environment

submitted by

SO, King-yan Oldfield

for the degree of Master of Philosophy
at the Chinese University of Hong Kong

Abstract

Advances in the multimedia and networking technologies allow the realization of the distributed virtual environment (DVE) system. A DVE is a distributed system that allows many users who are located in different nodes in the network to concurrently explore and interact with each other in a high-resolution, 3-dimensional, graphical virtual world.

In general, a DVE can be operated in either a local area network (LAN) or a wide area network (WAN). It is important to have an architecture which can handle both the large number of users and the variation on the network delay throughout the whole network. We formulate the load balancing problem as a partitioning problem, and then we derive some algorithms to solve it, so that as the load is distributed, the total network bandwidth requirement is reduced at the same time.

Each user in a DVE is represented by an avatar and any action taken by an avatar should be observable immediately by all other avatars in the same virtual environment and to provide such a consistent view, synchronization is required. To allow the efficient use of the underlying network resource, construction algorithms for some communication subgraphs are described. Then, we propose a mechanism to perform object state synchronizations. Afterward, we define the notion of consistency and finally, we describe several methods to derive the optimal synchronizing interval based on the consistency requirements and the system parameters.

Acknowledgments

I would like to thank Prof. John C.S. Lui, my supervisor, for his guidance and patience. He never fails to provide me with his insightful advices and comments on my research. He is always here to give whatever support to his students. He also helped me a lot on my job searching. I am proud to be one of his students.

I would also want to express my thanks to my colleagues in SHB 1026, for the discussions on our research, the informal reviews on our research papers, the wonderful games and chats. I will never forget the time I have in there.

Finally, I would like to thank all of my family members and my girl friend, Ling, for their support, care and love. My parents can really provide me with an excellent environment so that I can concentrate on my studies, their dedication to their children is one of the most wonderful gift from my God. I praise the Lord my dearest God.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Communication is a process by which information is exchanged between individuals through a common system of symbols, signals or behavior. Communication is important for any individual inside a society to establish relations and produce mutual benefits.

1.1 Evolution of Communication Technologies

The earliest forms of communication among human beings have the limitation that they can only be used when two individuals are sufficiently close to each others. Examples include verbal conversations, communication through bodily gesture, etc. Later with the commencement of the postal service (including the delivery of letter by pigeon, etc), people can communicate with each others remotely. However, there is usually a large time lag after the initiation of the communication to the arrival of the information at the receiver side. Only after the invention of the telegram and later the telephone system, *real time* communication was then possible.

Modern communication technologies like the telegram, telephone system, the facsimile, the video conferencing system, are proved to be very useful in the modern society by their wide popularity. These technologies are designed to suit various needs and they have been enhanced in such a way that they are becoming faster and cheaper. Information is becoming extremely vital in the business world and the efficient transmission of information is very important.

However, the above mentioned communication technologies imposed a limitation on the format of the data being transmitted, for example, we can only communicate through voice with a telephone. Even with the video conferencing technology, we are still limited in the transmission of images and voice. This limitation, however, does not exist in a computer based communication network.

1.2 The Internet

The Internet is the largest global computer network in the world, and it is also the first globally available computer network. The number of computers connected to the Internet keeps increasing everyday.

With the rapid expansion of the Internet, computer has become one of the most important tools for communications and information exchange. Notice that the communication technologies mentioned previously can also be implemented in a computer network. On the Internet, we have the email service to provide postal like services, we have the `talk` utility in UNIX to provide real time conversations based on text, for example.

With the World Wide Web, documents with graphics or even multimedia data can be incorporated together by using the Hypertext Makeup Language (HTML) and then transmitted by using the Hypertext Transmission Protocol (HTTP) through the Internet. Not only text or images can be transmitted, but it can also support real time audio, video data, etc., by means of extra plug-ins in the web browsers. The web starts a complete revolution in the communications based on the computer network, and its rate of growth shows its usefulness and popularity.

Since *multimedia* data can be transmitted through a computer network and therefore there is virtually no limitation on the format of data being transmitted. For example, a designer can send a three dimensional geometrical model of an aircraft prototype to an aeroplane developer through the computer network, while in the old days, we can only send the blueprint of the design on the paper, say by using facsimile. The growth of the Internet opens up the potential ability on the further use of this flexible and efficient means of communications.

1.3 The Distributed Virtual Environments

With the advances in computer graphics, multimedia systems, parallel/distributed systems and high speed networking technologies, it is now possible for the computer scientists and engineers to build a distributed virtual environment (DVE) system [26].

A DVE is a distributed system that allows many clients who may be located in different nodes in the network to concurrently explore and interact with each other in a high-resolution, 3-dimensional, graphical virtual environment (or virtual world). Clients who are exploring the virtual environment can (1) extract relevant information about the virtual environment (e.g., by sending database query to the DVE system and inquire about the state of any object in the virtual environment), (2) communicate in real time with other clients who are also exploring the same virtual environment. Like any other computer technology, DVE will change the way how people work, interact and share the information. In the near future, people may regularly log in a DVE system just like we read our email today. Through the DVE system, they can enter a highly graphical computer generated virtual world and they can meet and interact with other people.

1.3.1 Features of DVE

The DVE provides the following special features which is either unique or is especially attractive with respect to any other existing technology:

1. The DVE breaks the barrier of geographical separation. Users from different part of the world can join the same virtual world through a DVE system on the computer network. It saves the time for long-distance traveling, and it certainly reduces the cost.

2. The DVE provides a computer generated environment such that it might not necessarily be real. Part of the environments can be computer simulated and it is useful for training, for examples, astronauts can be trained inside a DVE with a computer generated space-craft, soldiers can learn how to react to different situations in a simulated war-field environment.

3. The DVE produces a common environment to the users so that they can be more involved in the interaction. Unlike the video conferencing technology, every user is not bounded inside their own room. They share a common virtual world and yet, it poses no threat to any of the participant.

4. The DVE creates a virtual world such that it can be treated as the real world. Nearly everything can be done in reality inside the virtual world. A user can, for examples, shake hand with other participants, and even play cards, etc.

1.3.2 Current and Potential Applications

The Distributed Interactive Simulation (DIS) is a similar concept to the DVE and it is now widely used in military applications. It provides a realistic computer generated environment and it supports thousands of simultaneous users in the training of the armed forces. Immersive simulators are used for teaching the army on how to operate military vehicles such as the planes, helicopter and tanks in a virtual world with simulated enemies. Strategic planning can be done with the help of these DIS applications.

As for an example of a civilian use of a DVE system, we consider the following situation to show how the DVE technology may affect our daily life and business operation. An architect from France, a civil and a structural engineer from the Los Angeles, a financial planner from Hong Kong and an interior designer from Tokyo, who are having a business meeting concerning about the design and financing issues of a new high-rise office complex which will be built in London. Under the DVE system, these people can interact in a virtual world of the new high-rise office complex that they are proposing to build. Each participant in this business meeting can virtually walk through the high-rise office building. They can interact and carry out the discussion without leaving their own homes. For example, in this virtual environment of high-rise office complex, each user in the meeting is represented by a 3D object, which is known as an *avatar*, and each participant can walk around in this virtual office building, and in the process, they can rearrange any 3D object (e.g., furniture, paintings, selecting different kinds of wall papers, ... etc) in the environment. Any change to a 3D object in this virtual environment will also be *visible* to all the participants. Also, during the meeting in this virtual 3D environment, the participants will be able to interact with each other in real time, as well as to inquire the information about the virtual world that they are exploring.

Note that there are many other types of application that can be built in the form of the DVE system, for example:

1. Education: Students and teachers can interact in a virtual classroom. For example, the teacher can guide a group of students and together, they can explore a virtual museum and the teacher will be able to explain the historical implication of each object in the exhibit to the students.
2. Collaborative Group-ware Application: Either in a business or engineering development environment, each participant in the meeting can see the changes of

the business models or engineering designs, and be able to inquire the specific information about the product that they are trying to market or to build.

3. Internet Shopping: A virtual shopping center such that buyers and sellers can interact and negotiate about the price of the selling item and at the same time, people do not need to leave their homes and still be able to visualize the products in a high-resolution, three dimensional format. It is definitely more intuitive and immersive than the web-based counterparts.

4. Entertainment: Network computer games can be built in the form of a DVE and more players from different part of the world can participate even if they do not know each others. Imagine a football game which involves twenty-plus people around the world!

5. Tele-presence: Robotic facilities in remote site can be controlled through the corresponding simulated objects inside the virtual world of the remote site. Sensors can be installed in the remote site so that the state of the objects inside the virtual world can be kept synchronized with the real world objects.

6. Cooperated Interior Designs: Designers from different part of the world can jointly conduct interior design work inside a room of a virtual building.

1.3.3 The Challenges

To deploy a DVE system in a local area network (LAN), a private network or even in wide area networking environment, such as the Internet, we need to design a cost-effective, scalable DVE system. There are many research issues that need to be addressed. For example:

- Designing an efficient back-end database engine so as to give good throughput and response time for any query submitted by the clients who want to know some information about the virtual environment that they are exploring.

- Designing an efficient communication protocol so that clients who are located in different regions of the world can communicate in real time without consuming too much network bandwidth and with acceptable delay.

- Maintaining that each user will have a consistent view of the virtual world. In other words, if there is an action taken by any user in the virtual world or if the state of any object in the virtual world is changed, every client should be able to view the change "immediately". In order to provide this consistent view, the DVE system needs to perform a *synchronization* action to every user's view periodically.

The first research issue can be answered by designing a back-end database engine which can efficiently process queries (either in relational or spatial form) submitted by the clients, as illustrated by the VINCENT system [13]. For the second research issue, we can utilize some of the recent work on the Internet real time protocols, such as the RSVP [27] and RTP[22], for the network bandwidth allocation and real time communication.

1.4 Our Contributions

In this thesis, we try to solve some of the major problems in the design of a DVE system. Our work can be divided into two parts. In the first part, we formulate and solve the load balancing problem of a DVE system. In the second part, we discuss the synchronization problem which includes the construction of the communication subgraph for the delivery of synchronization messages and the derivation of the optimal synchronizing interval with a given set of consistency requirements.

Chapter 2

System Architecture

In this chapter, we describe the system architecture of our DVE system. Our DVE system consists of two components namely the DVE server and the DVE client. The DVE server is responsible for the maintenance of the state information of the objects inside the virtual world. The DVE client, on the other hand, serves as an interface to the users to the virtual world, it renders the view of the virtual world for the users based on the state of the objects retrieved from the DVE server, it also calculates the changes in the state of the objects based on the actions of the users and reflect these changes to the DVE server. More than one users can join the same virtual world session through different DVE clients.

2.1 The SSDVE and MSDVE Architectures

A straight forward way of implementing a DVE system is to use a single DVE server to maintain the state of the objects in the virtual world. We call this the Single Server DVE (SSDVE) architecture. With the SSDVE architecture, we cannot scale up the size of the virtual world arbitrarily since the computation power of a single server machine is fixed. This poses a limitation on the size of the virtual world and also the number of concurrent users.

However, instead of using a single server machine, we can use more than one DVE server machines to maintain the state of the objects in the virtual world. We call this the Multiple Server DVE (MSDVE) architecture. With the MSDVE architecture, the virtual world is divided into several *partitions* and each of the DVE server machine is responsible for maintaining the state of the objects inside their own partition.

With the MSDVE architecture, any user can join the virtual world through a DVE client to connect to the DVE server which is responsible for maintaining the partition of the virtual world that the user is interested in. The MSDVE architecture is scalable, since we can always add more DVE server machines to increase the total computation power, so that a large virtual world can be maintained. Figure 2.1 illustrates the SSDVE and MSDVE architecture.

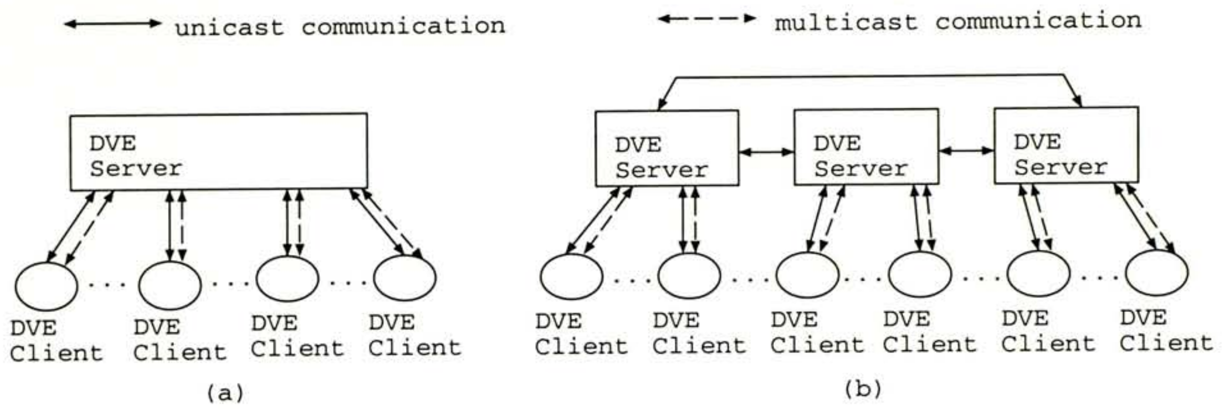

Figure 2.1: System Architecture for (a) SSDVE; (b) MSDVE

2.2 Issues in the MSDVE Architecture

We assume the MSDVE architecture in the rest of this text. The issues in the design and implementation of a very large scale DVE system based on this architecture are covered.

2.2.1 On the Server Side

The first issue we need to address is the work load distribution among the DVE servers. However, when we divide the virtual world into partitions, *inter-server communication* is required. We must derive a partitioning scheme such that as the work load is shared among the DVE servers, the amount of overhead due to this inter-server communication is minimized. The detail about this problem and our proposed solutions are given in chapter 3.

2.2.2 On the Client Side

Since the DVE clients of the same virtual world session can be running in different machines, the views of the virtual world rendered by these clients may not be the same. We would, however, want to keep these deviations to be as small as possible.

We called this property *view consistency.* In chapter 4 and chapter 5, we discuss how to maintain the *view consistency* among the DVE clients which are connected to the same DVE server.

In chapter 4, we describe what is a good communication subgraph for the delivery of the synchronization messages among the DVE clients through the *multicasting* technique. Then, we propose some algorithms to derive a suitable communication subgraph depends on the nature of the application. In chapter 5, we present some synchronization mechanisms and then we show how to derive the parameters for those synchronization mechanisms with a given set of consistency requirements and system parameters.

Chapter 3

Balancing Work Load and Reducing Inter-server Communication

In this chapter, we discuss about the load balancing problem in a DVE system. The computation power required by a very large scale DVE system is huge and therefore, it is neither scalable nor cost effective to implement such a big system in a single computer. With more than one computers, we have to consider the problem about how to use the computing resources in these computers.

In this chapter, a mechanism called the *partitioning* [14] is proposed to build a cost-effective and scalable DVE in a network cluster of computers. The system architecture is first described, and then the partitioning problem is formulated and algorithms are proposed to solve the problem. Finally, experimental results are shown to demonstrate the effectiveness of the algorithms proposed.

3.1 Problem Formulation

With the MSDVE architecture described in chapter 2, the problem is to decide how to divide a virtual world into *partitions* and to assign them to a given set of server computers. We call it the *partitioning* problem.

By dividing the virtual world into different partitions, inter-server communication is required when an avatar from one partition needs to interact with another avatar or objects in another partition. Therefore, the partitioning scheme should be able to divide the work load of maintaining the object states, and at the same time, minimize the inter-server communication induced by such a division.

We first introduce some necessary concepts like the *area of interest* [17] and the *DVE cells*. Then, we introduce two methods to find the expected number of avatars within a DVE cell. Afterward, we describe two different types of network and how they affect our cost metrics in the optimization of the inter-server communication and finally we define the partitioning problem formally at the end of this section.

3.1.1 The Area of Interest

The users of a DVE are called *avatars* inside the virtual world, an avatar can interact with other avatars as well as objects within the virtual world. We define the AOI of an avatar to be the area such that the avatar can interact with the other objects, including the other avatars, within.

Definition: The AOI of an avatar A is defined as the circular region with radius r_A measuring from A, i.e. A is the center of the circle, such that the value of r_A depends on the properties of the avatar A.

In general, the AOI of an avatar can be in arbitrary shape but for the sake of easier analysis, we use a disc as the shape of the AOI.

3.1.2 The DVE Cells

The inter-server communication cost of a partitioning scheme is characterized by the number of avatars which can interact with the other avatars or objects in another partition. In another words, inter-server communication is required when the AOI of an avatar in one partition encloses the other avatars or objects in another partition.

By using the concept of AOI, we divide the virtual world into small regions called the DVE cells which can then be used as the basic unit of the area in the partitioning of the virtual world and this division scheme can greatly simplify the evaluation of the quality of the partitioning scheme.

We divide the virtual world into small square regions and we call them DVE cells. We limit the size of DVE cells for a virtual world of avatars to satisfy the following:

$$\frac{S}{2} \leq r_A^{max} \leq S$$

where $r_A^{max} = \max_{\forall A_i}\{r_{A_i}\}$, and by imposing this restriction to the size of the DVE cells, we ensure that the *AOI* of an avatar will always exceed its home cell, as shown in Figure 3.1. More importantly, no *AOI* can run across two DVE cells. Therefore, the communication cost between two adjacent DVE cells, either horizontally or vertically, if they are assigned to different partitions, can be simplified to consider only the avatars in the neighboring DVE cells instead of considering all the avatars in the whole neighboring partition.

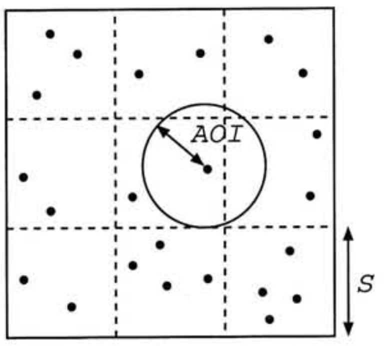

Figure 3.1: Relation of the Size of DVE Cell and AOI_{max}

3.1.3 Expected Number of Avatars

We propose two methods to find the expected number of avatars inside the DVE cells. The first one is an analytical method based on the theory of Markov Chain, and the second one is based on the run time statistics of the virtual world.

Setup time method: Analytical Method

We represent the movement pattern of each avatar by a Markovian Process. Let M_A be the mobility matrix of an avatar A where

$$M_A[i,j] = \begin{cases} \text{rate of avatar } A \text{ moving from cell } i \text{ to cell } j & \text{for } i \neq j \\ -\sum_{k\neq i} M_A[i,k] & \text{for } i = j \end{cases}$$

Given the mobility matrix M_A of avatar A, we can easily compute the steady state probability of avatar A at any given cell in the virtual world by solving the following system of linear equations:

$$\begin{aligned} \pi_A M_A &= \mathbf{0} \\ \pi_A \mathbf{e} &= 1 \end{aligned}$$

where $\pi_A[i]$ is the steady state probability that avatar A is in cell i and $\mathbf{e}$ is the column vector of 1's. Note that if all avatars can provide their mobility matrices upon entering the virtual world, then an effective server-cell assignment can be obtained during the arrival instants of each avatar.

Run time method: Statistical Method

If an avatar cannot provide the mobility matrix, we can approximate the expected number of avatars in a DVE cell by keeping a record on the number of avatars during the run time of the DVE. For example, we can take n snap shots of the DVE on the number of avatars in the DVE cells, and then the expected number of avatars of the DVE cells can be approximated by dividing the total number of avatars which have appeared in that DVE cell by n.

3.1.4 Cost Metrics in Different Types of Network

We describe two types of communication network and we show how they affect our cost measurement in the optimization of inter-server communications.

Weight of Edges in Bus-based network

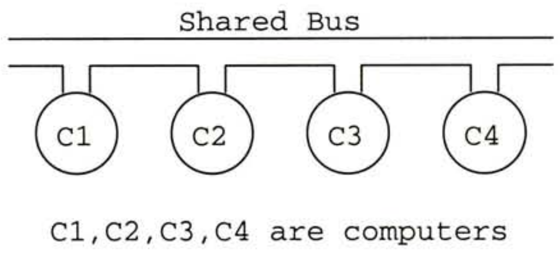

Figure 3.2: Bus-based Network

In a bus-based network, all the communication messages among the DVE servers are sent and received from this bus. Therefore, a good partitioning scheme should try to minimize the *total* amount of communication messages among all the DVE servers.

Weight of Edges in Switch-based network

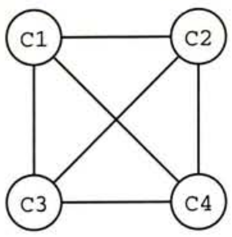

Figure 3.3: Switch-based Network

In a switch-based network, the communication messages among the DVE servers are sent and received virtually through a dedicated link, if this link is full-duplex, a good partitioning should try to minimize the *maximum* size of all the communication messages because messages can be sent and received in parallel and the total delay due to communication is then the time required for the transmission of the message having the maximum size.

3.1.5 Problem Definition

With the concept of the DVE cells, the partitioning problem becomes the assignment of the DVE cells to the partitions or the set of available DVE servers.

We assume that the work load of a DVE server is proportional to the expected number of avatars in the cells within its own partition. The inter-server communication cost of the assignment is proportional to the number of avatars in the *boundary* DVE cells, because only the avatars in the boundary DVE cells can generate inter-server communication.

We give a formal definition of the DVE partitioning problem by transforming it into a graph problem. With a given DVE, we can obtain the expected number of avatars within each cell with the method described in Section 3.1.3. We then transform the DVE system into a graph with its nodes representing the DVE cells and its edges representing the communication link between the adjacent cells.

Let us give an example to illustrate this idea, the DVE in Figure 3.4 is the original DVE, the graph in Figure 3.5 is the graph transformed from the original DVE system.

Notice that not all the weight of the edges of the graph in Figure 3.5 are shown for the simplicity of illustration and drawing. The value on the nodes are the expected number of avatars in the corresponding DVE cells, while the weight of the edges can be calculated in several ways, based on the nature of the network connection within the

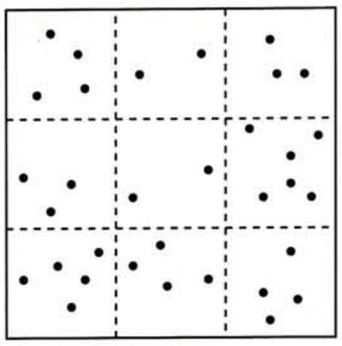

Figure 3.4: The Original DVE

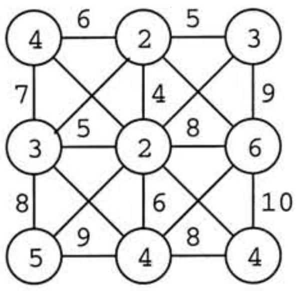

Figure 3.5: The Transformed Graph

DVE server cluster. In the example above, we have summed up the expected number avatars in the two end nodes to form the weight of the edges (i.e. we are modeling a bus-based network), and the weight of the diagonal edges can be calculated similarly.

Before giving the formal definition of the DVE partitioning problem, let us define the following notations:

N	=	Number of cells that compose the whole virtual world
P	=	Number of partitions or servers in the DVE system
c_i	=	DVE cell $i, 1 \leq i \leq N$
n	=	Number of avatars in the DVE system
A_i	=	Avatar i where $i = 1, 2, \ldots, n$
M_i	=	Mobility Matrix for A_i
W_i	=	Work load in cell $c_i, i = 1, 2, \ldots, N$
L_{ij}	=	Communication cost for the link between c_i and $c_j, 1 \leq i, j \leq N$
w_1	=	A non-negative weight of the work load cost on a server
w_2	=	A non-negative weight of the inter-server communication cost

$C^W_{\mathcal{P}}$	=	Work load cost of a given partition configuration $\mathcal{P}$
$C^L_{\mathcal{P}}$	=	Communication cost of a given partition configuration $\mathcal{P}$
$C_{\mathcal{P}}$	=	Total cost for a given partition configuration $\mathcal{P}$

With the above notations, the graph G can be obtained from a given DVE system with the following algorithm:

```
Algorithm Graph_Transform(Input: DVE; Output $G$);
begin
    For each cell $c_i$, create a node $v_i$ in $G$ and assign $W_i$ with
    the expected number of avatars in cell $c_i$;
    For all cell $c_i$ and $c_j$, create an edge $E_{ij}$ in $G$ with end nodes $v_i$ and $v_j$
    if $c_i$ and $c_j$ are adjacent;
    For all $E_{ij} \in E$, computer the edge cost $L_{ij} = W_i + W_j$;
end;
```

We are now in the position to formally define our DVE partitioning problem. Given a graph $G = (V, E)$ with $|V| = N$, $\mathcal{P}$ is the partition that divides V into P (where P is the number of servers) disjoint subsets $V_1, V_2, \ldots, V_P$ such that $V_i \cup V_j = \emptyset$ for $i \neq j$ and $\bigcup_{i=1}^{P} V_i = V$. Given a partitioning scheme $\mathcal{P}$, we can define the work load cost $C^W_{\mathcal{P}}$ on this partition such that:

$$C^W_{\mathcal{P}} = \sum_{j=1}^{P} (\sum_{v_i \in V_j} |W_i - \frac{n}{P}|) \tag{3.1}$$

Note that the term $\frac{n}{P}$ is the work load under the ideal partitioning scheme where there is equal work load in each partition. Therefore, the physical interpretation of the above equation is to represent the deviation of work load from the ideal case.

One might argue that if all the server computers can handle the work load, there is no point to make them share the same amount of work load. Let us consider an extreme case such that the work load of one of the server has reached its limit, and therefore, that server cannot admit any more new users. On the other hand, a system with all the work load equally shared can handle newly admitted users if the total computation power of the servers allowed, and so it is better for us to devise a partitioning scheme which can distribute the work load equally to all the server nodes, rather than just to ensure every server nodes can handle the work load assigned.

Let us define the following function between a cell u and a partition V_l:

$$ADJ(u, V_l) = \begin{cases} 1 & \text{if } \exists v \in V_l \text{ s.t. } E_{uv} \in E \\ 0 & \text{otherwise} \end{cases} \tag{3.2}$$

Then , given a partitioning scheme $\mathcal{P}$, let C_{ij} be the communication cost between partition V_i and V_j and C_{ij} can be expressed as:

$$C_{ij} = \sum_{u \in V_i} W_u \cdot ADJ(u, V_i) + \sum_{u \in V_j} W_u \cdot ADJ(u, V_j) \tag{3.3}$$

Let $C^L_{\mathcal{P}}$ be the communication cost for partitioning scheme $\mathcal{P}$ and it is:

$$C^L_{\mathcal{P}} = \sum_{i=1}^{P} \sum_{j>i}^{P} C_{i,j} \tag{3.4}$$

Therefore, $C^L_{\mathcal{P}}$ represents the total inter-server communication cost given the partitioning scheme $\mathcal{P}$. The overall cost for the partitioning scheme $\mathcal{P}$ can be expressed as:

$$C_{\mathcal{P}} = w_1 C^W_{\mathcal{P}} + w_2 C^L_{\mathcal{P}} \tag{3.5}$$

Finally, the partitioning problem is to find an optimal partitioning scheme $\mathcal{P}^*$ such that

$$C^*_{\mathcal{P}} = \min_{p} \{C_p\} \tag{3.6}$$

3.1.6 Complexity

Before going to the partitioning algorithms, we give a proof of the following theorem:

Theorem 1: The DVE partitioning problem is NP-complete.

Proof: Let us consider a simplified version of the partitioning problem where $w_2 = 0$, which correspond to the case that the network has infinite communication bandwidth and so the inter-server communication cost is negligible. Given a set of nodes in V, we partition them into P disjoint subsets $V_1, V_2, \dots, V_P$ such that $\cup_{i=1}^{P} V_i = V$ and the partitioning cost is:

$$C_{\mathcal{P}} = \sum_{i=1}^{P} |W_{V_i} - \frac{n}{P}| \tag{3.7}$$

where $W_{V_j} = \sum_{v_i \in V_j} W_i$. The main idea of the proof is to transform the subset sum problem [9], which is known to be NP-complete, to the above simplified version of the partitioning problem.

The subset sum problem can be described as follows. Given a set of real numbers $\mathcal{N} = \{a_1, a_2, \dots, a_N\}$ and a real value k, the subset sum problem is to determine whether there exists a partitioning of the set $\mathcal{N}$ into disjoint partitions $\mathcal{N}_1, \mathcal{N}_2, \dots \mathcal{N}_l$ such that the sum of the elements in each $\mathcal{N}_i$ is equal to k.

Given an instance of the subset sum problem, the reduction works as follows. We create a DVE cell for each element $a_i \in \mathcal{N}$, and the work load of the cells are the value of the element a_i. The number of partitions P for the partitioning problem is:

$$P = \frac{\sum_{a_i \in \mathcal{N}} a_i}{k} \tag{3.8}$$

If an input instance of the subset sum problem should return a yes, than it implies that the corresponding partitioning problem can be evenly divided up the work load among P servers. If the result is no, this implies that the partitioning problem will have a load imbalance cost which is greater than zero. Since we can reduce the input instance of a subset sum problem, in polynomial time, to the input instance of a partitioning problem and then use the algorithm of solving the partitioning problem to solve the subset sum problem, therefore, the partitioning problem is also NP-complete. ∎

3.2 Partitioning Algorithms

It has been shown in the earlier section that the general DVE partitioning problem is NP-complete. Therefore, we try to look into a simplified case to gain some insights of the general problem.

3.2.1 A Simplified Case

In our simplified DVE model, we assume that the avatars are *uniformly* distributed over the whole DVE, that is the number of avatars in all the DVE cells are equal. With this assumption, the area of a partition will then be proportional to the number of avatars in this partition.

Another implication is that, the number of avatars lying on the boundary of a partition is proportional to the length of the boundary of this partition. Therefore, by minimizing the *total length* of the boundaries of all partitions, the communication cost of the DVE server cluster is minimized.

The DVE partitioning problem in this simplified case is then become the packing of arbitrary shaped areas into a large area, such that the total length of the boundaries of these areas is to be minimized.

This problem has been evaluated by mathematicians and it is a general belief that hexagons should be the best shape of the partitions under the above metrics. The hexagonal honeycomb conjecture [18], has been proposed and formulated in a number of ways.

Therefore, the DVE partitioning problem in this simplified case can be solved by using *hexagon* as the shape of the partitions. We use it as an approximation to the general case of the problem provided that the deviations on the expected number of avatars are small among the DVE cells. At the same time, it can be used as a baseline for us to evaluate and compare the performance of the other heuristics which attempt to solve the general problem. The hexagonal partitioning scheme in the simplified DVE model is shown in Figure 3.6.

3.2.2 The General Case

The partitioning scheme for the general case should be able to capture the information on the distribution of avatars to maintain the load balancing as well as communication

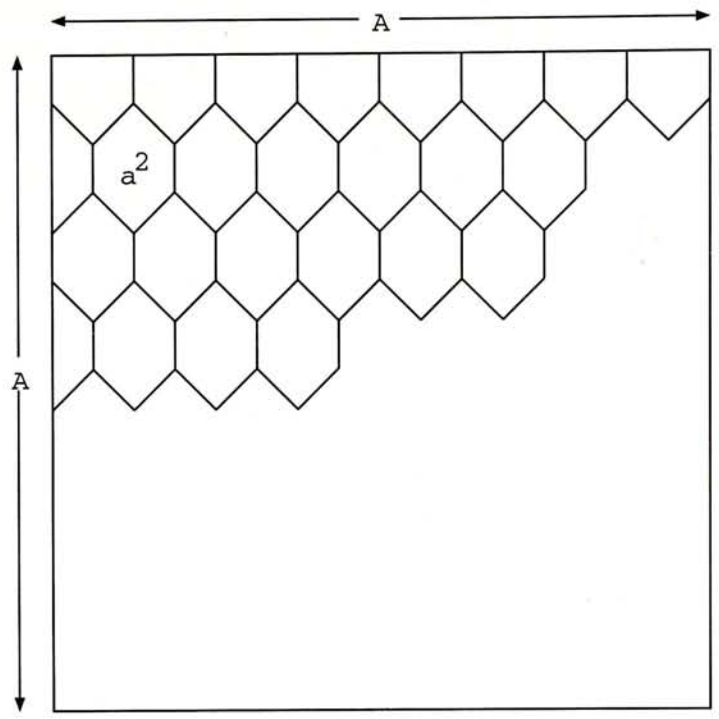

Figure 3.6: Hexagonal Honeycomb Partitioning

cost minimization.

Due to the dynamic nature of the distribution of the avatars in the virtual environment, we have to re-partition the DVE when required, so as to maintain the efficiency as well as cost effectiveness. That is, we have to find out *when* should we invoke the partitioning algorithm again to produce an efficient partitioning. Two simple criteria are that either when the work load is exceedingly skewed or when the communication cost among DVE server increases too much, that is, an imbalance of message cost in a switch-based network or the an dramatic increase in the total amount of messages in a bus-based network.

Exhaustive Partitioning Algorithm (EP)

As the name implies, it enumerates all the possible partitioning schemes and try to find the one with lowest cost. This algorithm is optimal but the complexity is $O(P^N)$ which is not feasible even for a moderate sized DVE.

Baseline Partitioning Algorithm (BP)

Given the NP-completeness nature of the partitioning problem, we try to study a simplified version of DVE first. In the simplified DVE, the avatars are uniformly distributed inside the virtual world. With this assumption, it implies that the number of avatars in the boundary cell, which turn out to be the inter-server communication cost, is proportional to the perimeter of the partitions. And load balancing in this simplified version of DVE can be achieved by assigning equal area of the virtual world to each of the available servers.

Mathematically, the DVE partitioning problem in the simplified version becomes the packing of arbitrary shaped area into a large area such that the total length of the perimeter of the small shaped area is minimized. Mathematicians proposed the hexagonal honeycomb conjecture [18] saying that regular hexagon is the shape of the partitions that we wanted.

For the ease of implementation, we use square instead of hexagon in our experiments below. And a performance drops of 7% can be derived by using square instead of hexagon, theoretically. The baseline partitioning algorithm (BP) is to assume that the underlying distribution of avatars in the given DVE is uniform, even if it is not the case. The complexity is $O(1)$.

Bisection Partitioning Algorithm (Bi-P)

This is our proposed attempt to give sub-optimal solution of the DVE partitioning problem in the general case based on heuristics. We call this algorithm the Bisection Partitioning Algorithm (Bi-P), and it is designed based on the concept of bisection. Without the loss of generality, let us first present the Bi-P algorithm for N cell system, and $P = 2$. Let P_k^n be the partition for the k^{th} server with n cells, initially, we set:

$$P_1^N = V = \{v_1, v_2, \dots, v_N\} \ ; \ P_2^0 = 0 \tag{3.9}$$

Let $\mathcal{P}_i$ be the i^{th} partition configuration and let $C_{\mathcal{P}_i}$, the cost based on the above equation, be the cost of partition configuration $\mathcal{P}_i$. Based on the initial partition, we have $\mathcal{P}_0 = (P_1^N, P_2^0)$ and the corresponding $C_{\mathcal{P}_0}$. We then can find $\mathcal{P}_1$ by moving one cell from P_1^N to P_2^0 and compute the cost $C_{\mathcal{P}_1}$. Note that the cell can be chosen in such a way that the total cost of $C_{\mathcal{P}_1}$ is minimized, which can be achieved by considering each cell in P_1^N and this process takes a linear time with respect to the total number of cells in the system. Formally, we have:

$$\mathcal{P}_i = (P_1^{N-i}, P_2^i) \text{ where } i = 0, 1, \dots, N \tag{3.10}$$

and then $\mathcal{P}_{(i+1)}$ can be derived by:

$$\begin{aligned}\mathcal{P}_{(i+1)} &= (P_1^{(N-(i+1))}, P_2^{i+1}) && (3.11)\\ &= (P_1^{(N-i)} - \{v_j\}, P_2^i \cup \{v_j\}) && (3.12)\end{aligned}$$

for $v_j \in P_1^{(N-i)}$ and $C_{\mathcal{P}_{(i+1)}}$ is minimized.

Note that $C_{\mathcal{P}_0}$ and $C_{\mathcal{P}_N}$ represent the two extremes of the highest load imbalanced cost (i.e., all the cells are assigned to one server and there is no inter-server communication). Therefore, the Bi-P algorithm is to choose a configuration that:

$$\mathcal{P}_{i^*} = \{\mathcal{P}_i | C_{\mathcal{P}_i} = \min_{0 \le j \le N} \{C_{\mathcal{P}_j}\}\} \qquad (3.13)$$

The above algorithm applies for $P = 2$. For larger number of P, we can first use the Bi-P algorithm mentioned above, then choose a partition that has the largest work load and then apply the Bi-P algorithm again until the desired number of partitions is generated. The complexity of the Bi-P algorithm is $O(\sum_{i=0}^{(\log_2 P)-1} \frac{N}{2^i})$.

Cell Shifting Operation

Since only the EP algorithm can generate the optimal partitioning, that means there are rooms for improvements for both the BP and Bi-P algorithms. A post processing technique called cell shifting is proposed to improve the resulting partitioning. In general, cell shifting operation is a process of assigning a cell from one partition to its neighboring partition such that the resulting partition $\mathcal{P}'$ has a lower total cost $C_{\mathcal{P}'}$ than $C_{\mathcal{P}}$. The cell shifting operation terminates when the total cost cannot be further reduced by cell shifting. Therefore, it is intuitively clear that the resulting partitioning after cell shifting operation should be at least as good as the original partition.

3.3 Experiments

In this section [1], we present the experiments for various algorithms that we discussed in the previous section. In experiment 1, we have a small virtual world with the dimension

[1]The experiment is a joint work with Mr. Peter T.S. Tam and Mr. M.F. Chan

of 4×4 cells and the number of avatars is 500 and $P = 2$. Since this DVE is small, we can compare our proposed algorithms with the EP algorithm, which guarantees to yield the optimal partition policy. In the second experiment, the have a large virtual world which is composed of 25×25 cells with the number of avatars being 2500 and $P = 8$.

In general, we use three different methods to generate the position of each avatar and they are:

- *Uniform Distribution:*. Let the position of an avatar be (x, y) and the values of x and y are uniformly distributed between $(0, V_x)$ and $(0, V_y)$ where V_x (V_y) is the horizontal (vertical) dimension of the virtual world.
- *Skewed Distribution:* Given the size of the DVE world is (V_x, V_y), we divide the number of avatars in the DVE systems into four equal sized groups, namely, G_i, $i = 1, 2, 3, 4$. Let (x, y) be the position of the avatar in group G_i. The value of (x, y) is generated in such a way that x is uniformly distributed between $(0, \frac{iV_x}{4})$ and y is uniformly distributed between $(0, \frac{iV_y}{4})$. Under this scheme, most of the avatars will be positioned within the square area defined by the two coordinates $[0, 0]$ and $[\frac{V_x}{4}, \frac{V_y}{4}]$.
- *Clustered Distribution:* Given the size of the DVE world is (V_x, V_y), we generate avatars around $k \geq 1$ clusters. First, we randomly generate k points $(x_1, y_1), \ldots, (x_k, y_k)$ such that x_i and y_i is uniformly distributed between $(0, V_x)$ and $(0, V_y)$ respectively. Then we divide the number of avatars in the DVE system into k equal-sized groups, namely, $G_1, G_2, \ldots, G_k$. For each avatar in group G_i, we generate its position in (x, y) where

$$x = \begin{cases} 0 & \text{if } x_i + dx \times \Omega < 0 \\ V_x & \text{if } x_i + dx \times \Omega > V_x \\ x_i + dx \times \Omega & \text{otherwise} \end{cases}$$

$$y = \begin{cases} 0 & \text{if } y_i + dy \times \Omega < 0 \\ V_y & \text{if } y_i + dy \times \Omega > V_y \\ y_i + dy \times \Omega & \text{otherwise} \end{cases}$$

Note that dx and dy are generated uniformly between $(-1, 1)$ and the parameter Ω depends on the size of the virtual world. For example, we set $\Omega = 0.4$ for the 4×4 cells sized virtual world and $\Omega = 3.0$ for the 25×25 cells sized virtual world.

Experiment 1: In this experiment, the virtual world is composed of 4×4 cells with the total number of avatars equal to 500 and the number of servers $\mathcal{P}$ is equal to two. Figure 3.7 illustrates this virtual world under three different distributions.

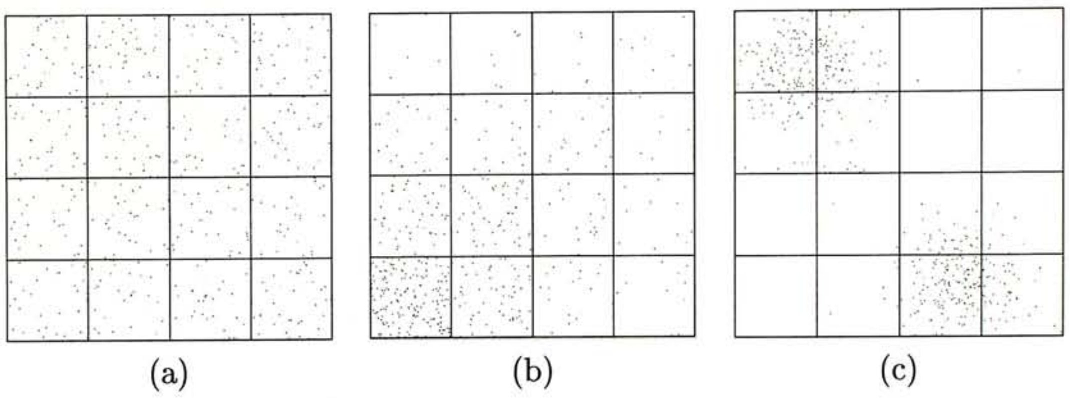

Figure 3.7: Virtual World with a 4 × 4 Cells under (a) Uniform (b) Skewed (c) Clustered distribution

In this experiment, we set $w_1 = w_2 = 0.5$.

When the avatars are uniformly distributed around in virtual world, we have the following result.

DVE with a 4 × 4 cells, 500 avatars under uniform distribution				
Algorithm	$(C^W_{\mathcal{P}})$	$(C^L_{\mathcal{P}})$	Overall Cost $(C_{\mathcal{P}})$	Execution Time T
Exhaustive	22	235	128.5	3.341
Baseline	218	249	233.5	0.164
Bisection	22	243	132.5	0.002

Figure 3.8 illustrates the partitioning of various algorithms.

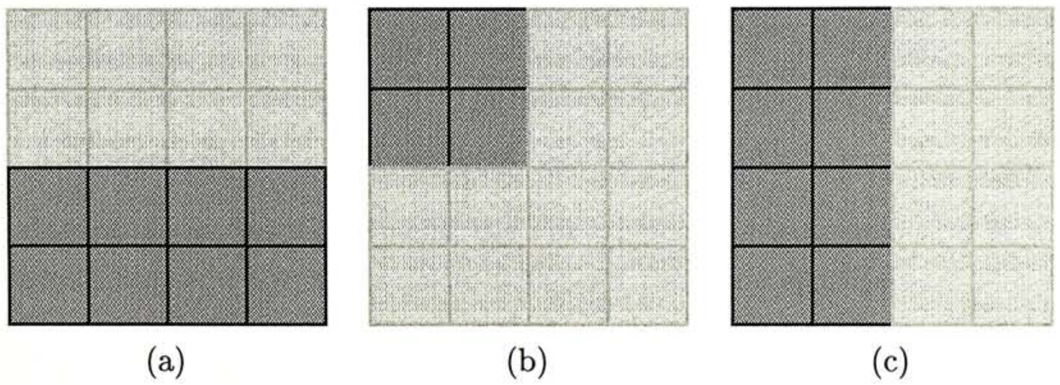

Figure 3.8: Partitioning under (a) Exhaustive (b) Baseline (c) Bisection Policy

When the avatars are skewly distributed around in virtual world, we have the following result.

DVE with a 4×4 cells, 500 avatars under skewed distribution				
Algorithm	$(C_{\mathcal{P}}^{W})$	$(C_{\mathcal{P}}^{L})$	Overall Cost $(C_{\mathcal{P}})$	Execution Time T
Exhaustive	2	396	199	3.341
Baseline	388	205	296.5	0.171
Bisection	6	397	201.5	0.002

Figure 3.9 illustrates the partitioning of various algorithms.

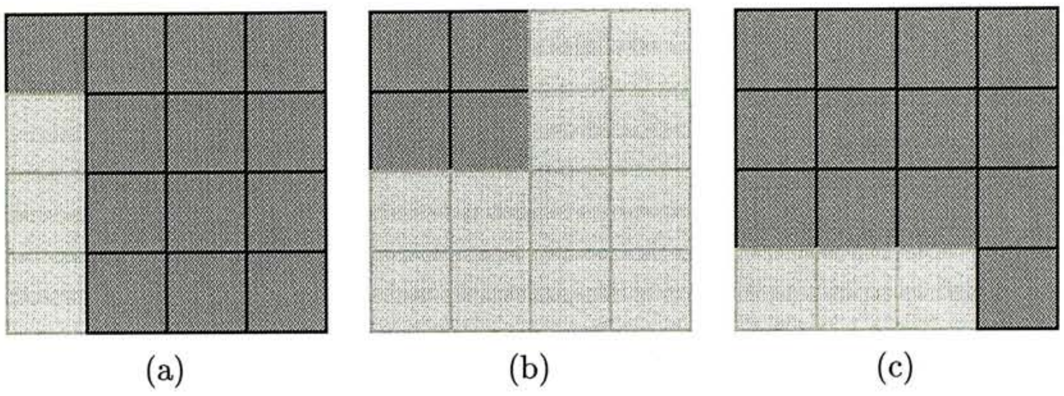

Figure 3.9: Partitioning under (a) Exhaustive (b) Baseline (c) Bisection Policy

When the avatars are distributed in a clustered fashion around in virtual world, we have the following result.

DVE with a 4×4 cells, 500 avatars under clustered distribution				
Algorithm	$(C_{\mathcal{P}}^{W})$	$(C_{\mathcal{P}}^{L})$	Overall Cost $(C_{\mathcal{P}})$	Execution Time T
Exhaustive	2	116	59	3.341
Baseline	4	178	91	0.164
Bisection	2	116	59	0.002

Figure 3.10 illustrates the partitioning of various algorithms.

Experiment 2: In this experiment, the virtual world is composed of a 25×25 cells with total number of avatars equal to 2500 and the number of servers $\mathcal{P}$ is equal to eight. Two algorithms are compared on these worlds, which are the Baseline Algorithm and the Bisection Algorithm.

Figure 3.11 illustrates this virtual world under three different distributions. In this experiment, we set $w_1 = w_2 = 0.5$.

When the avatars are uniformly distributed around in virtual world, we have the following result.

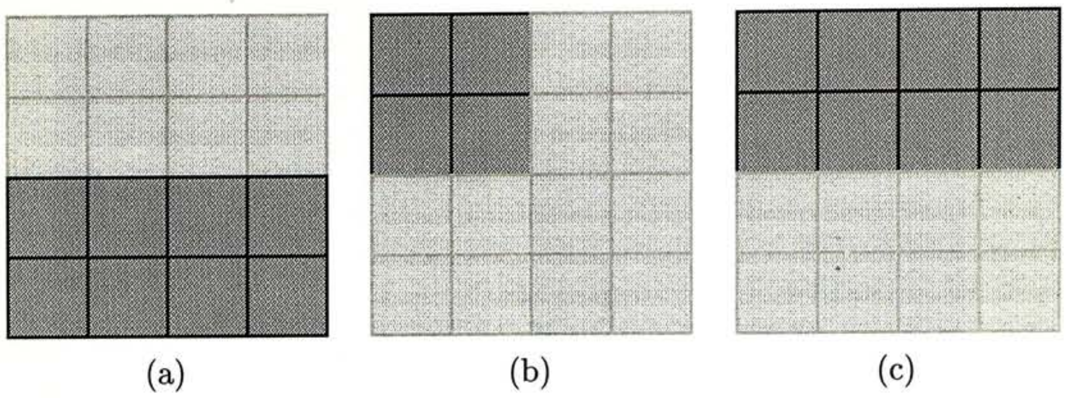

Figure 3.10: Partitioning under (a) Exhaustive (b) Baseline (c) Bisection Policy

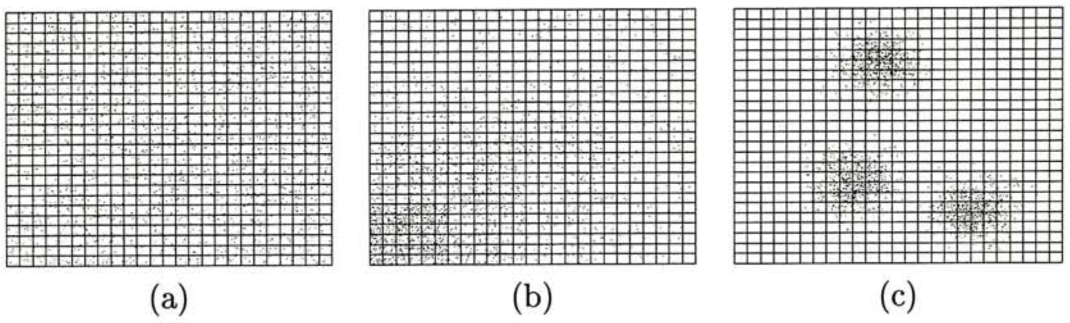

Figure 3.11: Virtual World with a 25 × 25 Cells under (a) Uniform (b) Skewed (c) Clustered Distribution

DVE with a 25 × 25 cells, 2500 avatars under uniform distribution				
Algorithm	$(C^W_{\mathcal{P}})$	$(C^L_{\mathcal{P}})$	Overall Cost $(C_{\mathcal{P}})$	Execution Time T
Baseline	536	1007	771.5	0.177
Bisection	5	1142	573.5	95.315

Figure 3.12 illustrates the partitioning of various algorithms.

When the avatars are skewly distributed around in virtual world, we have the following result.

DVE with a 25 × 25 cells, 2500 avatars under skewed distribution				
Algorithm	$(C^W_{\mathcal{P}})$	$(C^L_{\mathcal{P}})$	Overall Cost $(C_{\mathcal{P}})$	Execution Time T
Baseline	1606	1156	1381	0.183
Bisection	103	1267	685	127.575

Figure 3.13 illustrates the partitioning of various algorithms.

When the avatars are distributed in a clustered fashion around in virtual world, we have the following result.

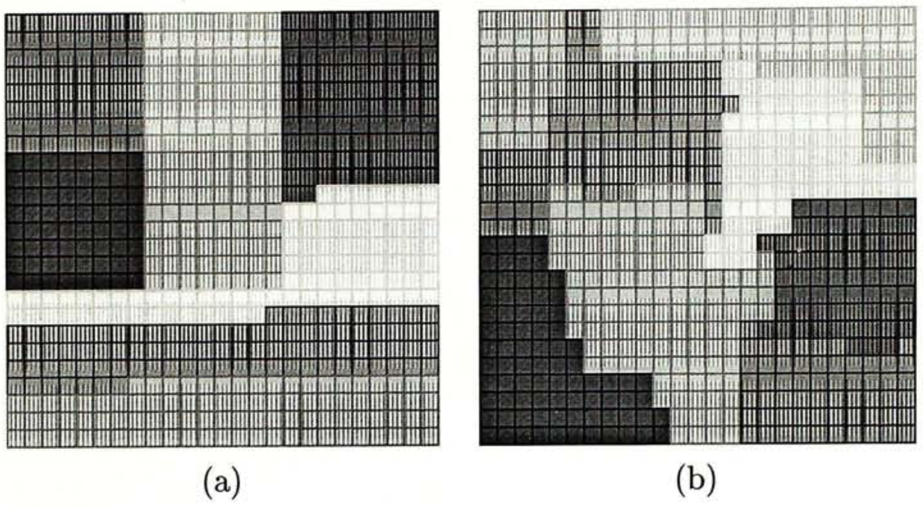

Figure 3.12: Partitioning under (a) Baseline (b) Bisection Policy

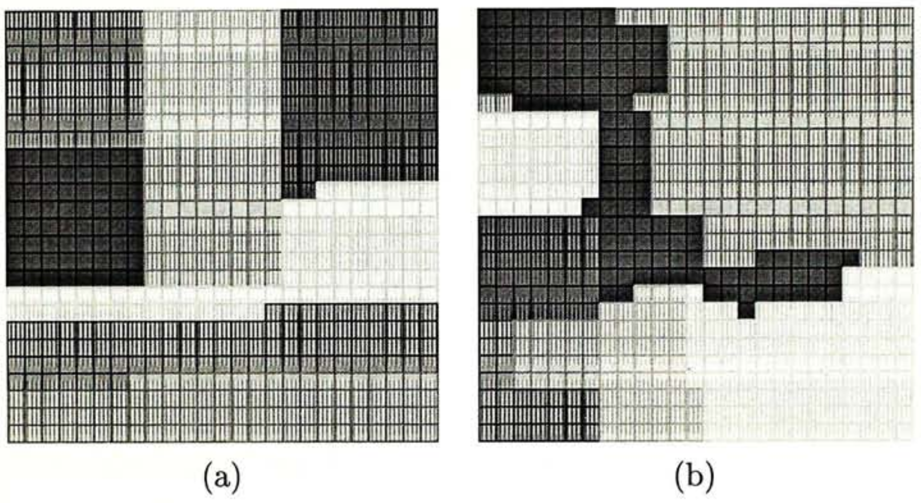

Figure 3.13: Partitioning under (a) Baseline (b) Bisection Policy

DVE with a 25 × 25 cells, 2500 avatars under clustered distribution				
Algorithm	$(C_{\mathcal{P}}^{W})$	$(C_{\mathcal{P}}^{L})$	Overall Cost $(C_{\mathcal{P}})$	Execution Time T
Baseline	2245.75	1402	1823.88	0.176
Bisection	555	2456	1505.5	152.270

Figure 3.14 illustrates the partitioning of various algorithms.

Experiment 3: In this experiment, the virtual world is composed of 25 × 25 cells with the total number of avatars equal to 2500 and the number of servers $\mathcal{P}$ is equal to eight. The virtual worlds are the same as the one found in Experiment 2. And again, we apply two algorithms, the Baseline Algorithm and the Bisection Algorithm to partition those worlds. However, we additionally apply the Cell-Shifting iterative improvement method to the results and obtain better partitions. As usual, in this experiment, we set $w_1 = w_2 = 0.5$.

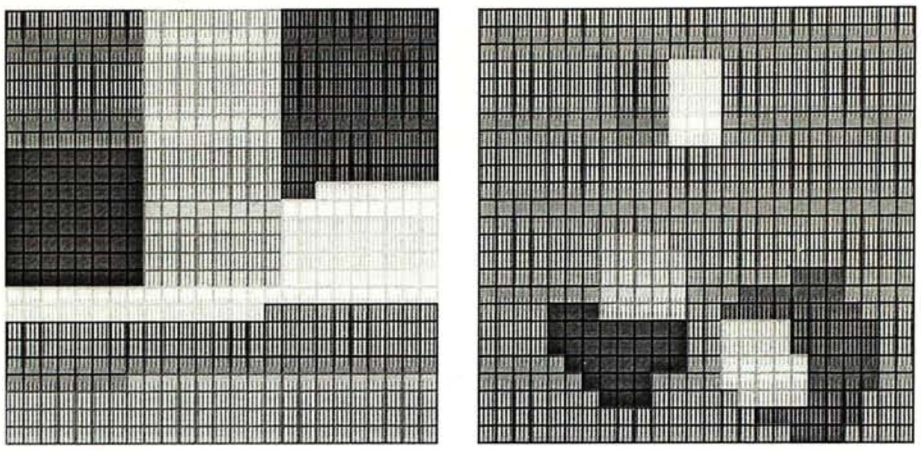
(a) (b)

Figure 3.14: Partitioning under (a) Baseline (b) Bisection Policy

When the avatars are uniformly distributed around the virtual world, we have the following result (additional cell-shifting method).

DVE with a 25 × 25 cells, 2500 avatars under uniform distribution				
Algorithm	$(C^W_{\mathcal{P}})$	$(C^L_{\mathcal{P}})$	Overall Cost $(C_{\mathcal{P}})$	Execution Time T
Baseline	356	1071	715.3	3.874
Bisection	5	1044	524.5	106.315

Figure 3.15 illustrates the partitioning of various algorithms.

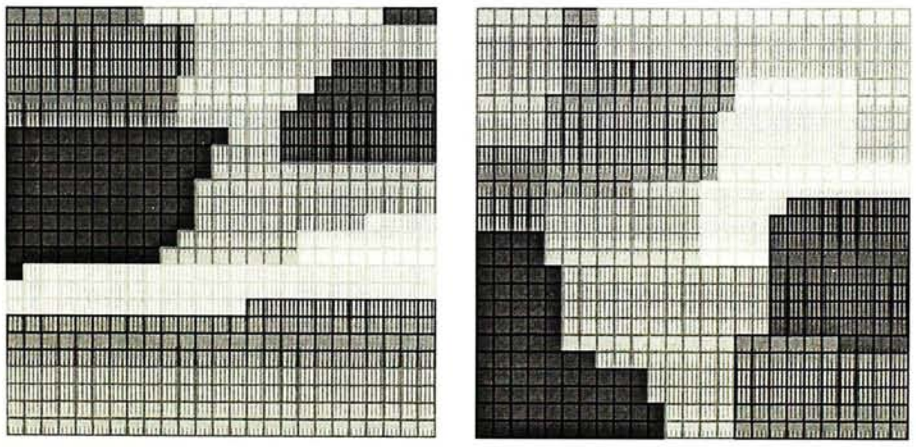
(a) (b)

Figure 3.15: Partitioning under (a) Baseline (b) Bisection Policy

When the avatars are skewly distributed around the virtual world, we have the following result(additional cell-shifting method).

DVE with a 25 × 25 cells, 2500 avatars under skewed distribution				
Algorithm	$(C_{\mathcal{P}}^{W})$	$(C_{\mathcal{P}}^{L})$	Overall Cost $(C_{\mathcal{P}})$	Execution Time T
Baseline	1564	1081	1322.5	3.460
Bisection	103	1211	657	132.125

Figure 3.16 illustrates the partitioning of various algorithms.

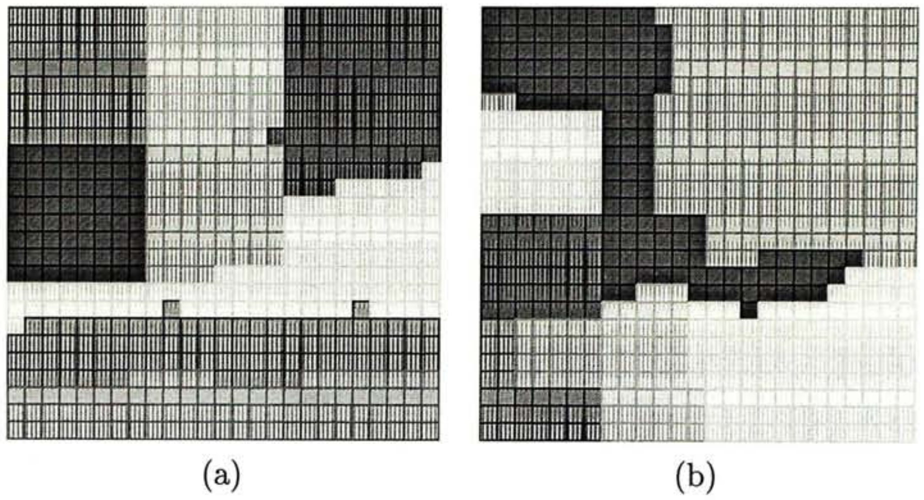
Figure 3.16: Partitioning under (a) Baseline (b) Bisection Policy

When the avatars are distributed in a clustered fashion around the virtual world, we have the following result.

DVE with a 25 × 25 cells, 2500 avatars under clustered distribution				
Algorithm	$(C_{\mathcal{P}}^{W})$	$(C_{\mathcal{P}}^{L})$	Overall Cost $(C_{\mathcal{P}})$	Execution Time T
Baseline	2635.75	995	1815.38	3.690
Bisection	555	2072	1313.5	162.289

Figure 3.17 illustrates the partitioning of various algorithms.

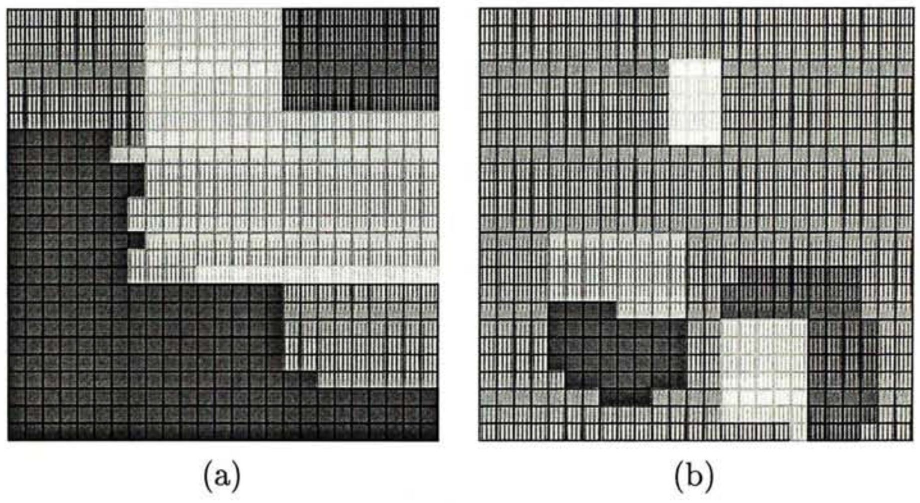
Figure 3.17: Partitioning under (a) Baseline (b) Bisection Policy

Chapter 4

Communication Sub-graph

Before going into the discussion on the synchronization mechanisms for maintaining the *view consistency* of the DVE clients, we first present the ideas on the construction of the underlying communication sub-graph [15] to support the delivery of the synchronization messages through multicasting. We are interested in doing that because we want to use the communication channels efficiently, and more importantly, we want to know the *maximum end-to-end delay*, we call it d_{max}, for the delivery of the synchronization message, in order to derive the *optimal synchronizing interval* in the next chapter.

In this chapter, we describe various factors which affect the design choices of the communication sub-graph. Then, we present several algorithms for the construction of the communication sub-graph, depending on the those design considerations.

4.1 Problem Formulation

The underlying network among the DVE clients is represented by a connected graph $G = (V, E)$. Given the current DVE client set $C(t) = \{c_1, c_2, \ldots, c_k\}$, we need to find a connected sub-graph $G' = (V', E')$ such that $C(t) \subseteq V'$ and $E' \subseteq E$. Note that in general, the number of nodes in G' is greater than k. The reason is that some intermediate nodes are needed between the DVE client nodes so as to provide the connectivity as well as to allow further optimization of the communication sub-graph.

4.1.1 Optimization Metrics

Since there are many possible communication sub-graphs G', we may want to optimize the sub-graph construction based on the following metrics:

1. Minimize the maximum *end-to-end delay* between any two clients in G'.
2. Minimize the total *bandwidth consumption* by minimizing the total edge cost of G'.

Formally, minimizing the maximum end-to-end delay refers to:

$$\min_{\forall G'} d_{max}(G') = \min_{\forall G'} \left\{ \max_{\forall i,j \in C(t)} \{d(p_{ij})\} \right\} \tag{4.1}$$

where $d(p_{ij})$ is the delay in the communication path between client i and client j, for $i \neq j$. Minimizing the total network bandwidth consumption refers to:

$$\min_{\forall G'} \left\{ \lim_{t \to \infty} \frac{\sum_{k=0}^{t} N(k)}{t} \right\} \tag{4.2}$$

where $N(k)$ is the total number of messages which are in transit (e.g., the transmitting messages which are in some communication paths in G') at time k.

4.1.2 Design Considerations

In this chapter, we consider two factors that may affect the optimization metrics on the construction of the communication sub-graph. These factors are:

1. The underlying networking environment.
2. The type of membership of the DVE clients.

We classify the underlying networking environment into two major classes: the LAN and the WAN (e.g., the Internet) environment. Under the LAN environment, the transmission bandwidth is high and the data transmission is usually fast with low latency and reliable. For a DVE system to operate on a LAN environment, the number of participants is usually small, for example, from tens to hundreds. Thus, it is possible to use a *centralized* algorithm to construct a communication sub-graph so as to minimize the maximum delay or to minimize the bandwidth consumption.

On the other hand, under the WAN environment which consists of many routers and sub-networks, the communication delay between any two clients is not negligible. The transmission bandwidth in the WAN environment is usually scarce and expensive when compared with the LAN environment. Moreover, the number of DVE clients in the WAN environment can vary a lot, ranging from tens to thousands. Due to the size of the network and the number of possible participating DVE clients, we need a *distributed* algorithm for the construction of the communication sub-graph.

Another factor that influence the design choices of the communication sub-graph is the type of the membership of the DVE clients. We classify the membership of the DVE client into two types, the *static membership* and the *dynamic membership.*

Static membership refers to the case that the set of participating DVE clients is constant throughout a DVE session. For example, in a DVE system for a teleconferencing application, where the set of participants are known ahead of time and it remains constant throughout the conference.

On the other hand, dynamic membership refers to the case that the set of participating DVE clients is time varying. For example, in a DVE system for an Internet game in which the client can *join* or *leave* the DVE system at any time.

Under a DVE system with static membership, we can construct the communication sub-graph before admitting the participating clients and the communication sub-graph will be discarded at the end of the DVE session. Under a DVE system with dynamic membership, the communication sub-graph is initialized at the beginning of the virtual environment session. Whenever one or more clients want to join or leave the virtual environment session, the DVE system may have to *modify* or *re-create* a new communication sub-graph. Therefore, an *incremental* sub-graph construction algorithm is favorable in this case.

4.2 Communication Sub-graph Construction Algorithms

With the consideration of the above factors for the construction of the communication sub-graph, we present the following sub-graph construction algorithms: [1]

1. The minimum diameter sub-graph (MDS)
2. The core-based tree (CBT)
3. The minimum spanning tree (MST)

4.2.1 The Minimum Diameter Sub-graph (MDS)

The minimum diameter sub-graph (MDS) is a natural choice when we want to construct a multicast sub-graph for a communication network. The reason is that the MDS provides a guarantee on the delay bound between any two client nodes. The MDS ensures that for every pair of client nodes in the sub-graph, there exists a path between them which is having a length less than or equal to the diameter of the sub-graph. In the context of our DVE application, the diameter is particularly important, since the diameter is the delay bound of all the messages transmissions.

The definition of the diameter of a graph is the length of a path, where this path is the longest among all the possible shortest paths within the graph. Formally, the diameter of a graph $G = (V, E)$ is:

$$\text{Diameter}(G) = \max_{v_i, v_j \in V} d(p_{v_i, v_j})$$

where p_{v_i,v_j} is the shortest path between the two nodes v_i and v_j, and $d(p_{v_i,v_j})$ is the delay associated with the path. An important point is that the minimum diameter sub-graph G' is not unique.

It is worth to note that when we remove any edge from a graph, provided that it does not destroy the connectivity, the diameter of the graph will either be increased or remain unchanged. On the other hand, if we add some edges to the graph, the diameter of the graph will either be decreased or remain unchanged.

Aside from finding the MDS of a graph G, one interesting question is whether we can find a MDS which also has the minimum total edge cost among all the possible

[1]This part is a joint survey conducted with Mr. T.S.Tam

MDS so that we can also minimize the total network bandwidth consumption. To answer this question, we have the following theorem:

Theorem 1 Given a graph $G = (V, E)$ and for each edge $e \in E$, the edge cost is $d(e) \in N$. For a given positive integer B, the process of finding a spanning sub-graph $G' = (V, E')$ for G (where $E' \subseteq E$) such that the sum of the cost of the edges in E' does not exceed B and the diameter of G' is minimized, is NP-complete.

Proof: Please refer to [21]. ∎

Because of the NP-completeness nature of the problem, we relax our requirement of minimizing the total edges cost, so that we can come up with an algorithm to find a minimum diameter sub-graph (MDS) in polynomial time.

In general, to find the MDS of a graph G, we perform the following steps:

1. Find the all pairs shortest paths between every pair of nodes in the graph $G = (V, E)$.
2. Union the edges within all the above shortest paths found, to form a new set of edges E'.
3. Then, the resulting $G' = (V, E')$ is a MDS.

We have presented the general idea on the construction of a MDS of a graph G, but in the context of the DVE application, the situation is different and we need some *pre-processing* before we can obtain a MDS for a DVE application. The reason for this step is that, because the clients set $C(t)$ is not equal V, the set of nodes in the network. For a given graph $G = (V, E)$ and a set of clients $C(t)$, we are seeking a sub-graph $G' = (V', E')$ which satisfies the following:

1. The set V' must include the set of all clients, i.e. $C(t) \subseteq V'$, in addition, V' must be a subset of V $(V' \subseteq V)$.
2. The longest path among those paths between any two clients should be minimized, i.e. we minimize the following:

$$\max_{\forall i,j \in C(t)} d(p'_{ij})$$

where p'_{ij} is the shortest path between client i and client j with respect to sub-graph G'

The algorithm of finding a MDS communication sub-graph for our DVE application is given below.

```
$V' \leftarrow \emptyset$
$E' \leftarrow \emptyset$
for each pair of clients $i, j \in C(t)$ do
    Find the shortest path $p_{ij}$ between $i$ and $j$ where $p_{ij}$ is
    a set of edges that constitute to a path
    for each edge $e_s \in p_{ij}$ do
        if $e_s \notin E'$ then $E' \leftarrow E' \cup \{e_s\}$
        Let $v_{s1}$ and $v_{s2}$ be the vertices on the two end of the edge $e_s$
        if $v_{s1} \notin V'$ then $V' \leftarrow V' \cup \{v_{s1}\}$
        if $v_{s2} \notin V'$ then $V' \leftarrow V' \cup \{v_{s2}\}$
    end for
end for
The MDS is found in $G' = (V', E')$
```

Theorem 2 The above algorithm guarantees that the diameter of the sub-graph G' is minimum.

Proof: We prove by contradiction. We first assume that the sub-graph G' of G is not a minimum diameter sub-graph. This implies the existence of a sub-graph G'' of G with a smaller diameter. Then we choose the path p'_{ij} between client i and j which is the longest shortest path in G' (i.e., diameter of $G' = d(p'_{ij})$). We also take the shortest path p''_{ij} from G'' for client i and j. Based on the definition of the diameter of a graph, $d(p''_{ij}) \leq$ diameter of G''. Combine with the fact that the diameter of G'' is less than the diameter of G', we have $d(p''_{ij}) < d(p'_{ij})$. However, the path p'_{ij} is guaranteed to be the shortest with respect to the graph G with client $C(t)$, therefore it is impossible to find another path p''_{ij} which is shorter than p'_{ij} and therefore contradiction occurs. ∎

An Example for the MDS Construction

Figure 4.1 shows an example of finding a minimum diameter subgraph. Figure 4.1(a) is a graph that represents the model of a network. Assume that all the nodes are participating nodes, then Figure 4.1(b) is the MDS corresponding to the graph in (a). It is worth to note that the MDS may contain cycles and usually, the total cost of edges is larger than the total cost of edges in the corresponding minimum spanning tree.

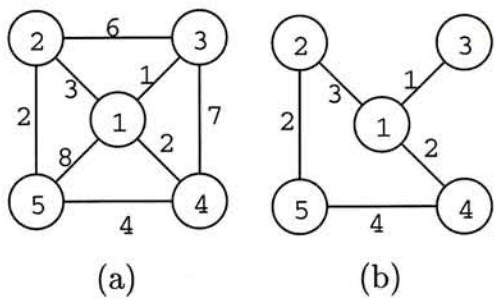

Figure 4.1: MDS Example (a) the Original Graph (b) MDS of Graph in (a)

4.2.2 The Core-based Tree (CBT)

The core-based tree (CBT) was proposed in [2], which is intended to provide a general framework for sub-graph construction in a large scale network where clients are located in different points of the Internet. The features of CBT is that:

1. There is a designated node called the *core*.
2. The path between any node to the core node must be the shortest path.
3. The routing policy for each node (or router) is efficient and easy to implement (e.g., based on the current Internet routing protocol).
4. The core-based tree is constructed in an incremental manner, such that any change to the client membership imposes only little changes to the communication sub-graph.

However, it is important that the CBT does not guarantee a minimum diameter spanning tree, it only guarantees that the path between the core node and any node (i.e. the participating client) is the shortest.

A CBT is constructed in a distributed, incremental manner. The construction procedure of CBT is as follows: at the very beginning, a core is chosen from the set of participating clients either manually or via a bootstrap mechanism as discussed in [2]. The core node is the first and the only node which is on the communication sub-graph at this stage. It would be natural for our DVE application to choose the common DVE server of the DVE clients to be the core.

Then, each participating client will send a `JOIN_REQUEST` message to the core (the IP address of the core node is advertised and well-known). This join message is sent through the shortest path from the participating client node to the destination core and this can be accomplished via the existing Internet routing protocol. Along the shortest path, the join message may reach a node which is either in the current communication sub-graph (i.e., a node which has already joined the CBT) or a node that is not part of the tree.

If the join message reaches a node which is a part of the multicast tree, the forwarding process will stop and the incoming interface (channel) will be added to the *forwarding cache* of the visiting node. Then, the visiting node will send an acknowledgment message (`JOIN_ACK`) back to the participating client node via the incoming interface.

On the other hand, if the join message reaches a node which is not part of the multicast tree, the visiting node will redirect the message to the next hop along the shortest path towards the core node and caches the incoming interface and the incoming node on the temporary storage. After that the node waits for an acknowledgment message. Once an acknowledgment is received, the node will put the incoming interface to the forwarding cache and redirect the acknowledgment to the nodes listed in the temporary storage. Also, it will set the node which sent the acknowledgment to be its parent node. It is clear that under this construction scheme, a multicast tree will be formed and this tree guarantees the shortest path from any participating client to the core node.

Once the CBT is formed, we can start the multicast service. When a client sends a multicast message, it must first consult the content in the *forwarding cache*. The information stored in the forwarding cache of the node v is a list of neighbor nodes of v in the CBT. Therefore, to multicast a message, a client can simply sends the message to all the nodes listed in its forwarding cache. When a node receives a message, it should forward the message to all the outgoing interfaces listed in the forwarding cache, except the incoming interface. In this way, the message can go through the whole tree, and stop at the leaf nodes.

The algorithm for the CBT construction for our DVE application is given below. Initially, the CBT consists of one core node only, and we assume that this core is one of the client $v_c \in C(t)$. The initial CBT is $G' = (V', E')$, where $V' = \{v_c\}$ and $E' = \emptyset$. Besides, there is a temporary storage associated with each node v, and denote it by tmp_v. The temporary storage of each node is empty at the beginning of the CBT construction procedure, i.e. $tmp_v = \emptyset$, $\forall v \in V'$.

A client v who wants to join the DVE can send a JOIN_REQUEST message to the core node v_c by sending the message to its adjacent node v' which is along the shortest path from v to the core node v_c. When node v' receives a JOIN_REQUEST from node v, the following procedure will be executed:

```
if $v' \in V'$ then
    add $v$ to the forwarding cache;
    reply an JOIN_ACK message to $v$;
else
    $tmp_{v'} \leftarrow tmp_{v'} \cup \{v\}$;
    send a JOIN_REQUEST  message to the first computer
        along the shortest path from $v'$ to the core $v_c$;
```

When node v receives a JOIN_ACK message from node v', the following procedure will be executed:

```
$V' \leftarrow V' \cup \{v\}$;
$E' \leftarrow E' \cup \{e\}$, where $e$ is the edge connecting $v$ and $v'$;
add $v'$ to the forwarding cache in $v$;
set $v'$ to be the parent of $v$;
reply a JOIN_ACK to all the neighboring node $u \in tmp_v$;
clear the temporary storage $tmp_v$;
```

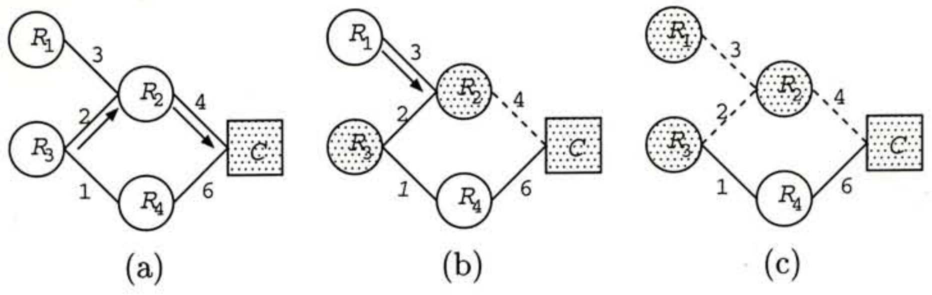

Figure 4.2: An Example of CBT Construction

An Example for the CBT Construction

An example of the CBT construction is depicted in Figure 4.2. Figure 4.2(a) shows a network topology of one core C and four nodes R_1, R_2, R_3 and R_4. The number besides each edge is the cost (i.e., delay) of the edge, i.e. $d(e)$. We use a shadow pattern to indicate that the corresponding node is a part of the multicast tree. In this example, the node R_3 wants to join the CBT and then the node R_1 wants to join the CBT. Initially, only the core C is on the tree, and we use the shaded nodes and doted line to denote the nodes and edges of the tree. At the beginning, node R_3 sends a `JOIN_REQUEST` message toward the core, as shown in the arrows in Figure 4.2(a). The intermediate node R_2 receives the request message, and then processes it. Since R_2 is not on-tree, it stores R_3 on its forwarding cache, after that R_2 sends a request message toward C. The core C replies the acknowledgment to R_2, and in turn R_2 sends an acknowledgment to R_3. Because of the acknowledgment, C is added to the forwarding cache of R_2 and C is marked as the parent of R_2. Similarly R_2 is added to the forwarding cache of R_3 and R_2 is marked as the parent of R_3. After the above process, the forwarding cache of R_2, R_3 and C is: $\{R_3, C\}$, $\{R_2\}$ and $\{R_2\}$ respectively. Also, R_2 and R_3 are now part of the tree as depicted in Figure 4.2 (b). Now, consider the case that node R_1 wants to join the multicast tree. It sends a join request message to R_2. Since R_2 finds itself is already part of the multicast tree, it replies an acknowledgment to R_1 directly and adds R_1 in its forwarding interfaces list. Node R_1 adds R_2 into its forwarding interfaces list after receiving the acknowledgment, and mark R_2 as its parent.

4.2.3 The Minimum Spanning Tree (MST)

Consider the following problem: given a graph $G = (V, E)$ where each edge $e \in E$ is associated with an edge cost $d(e)$. There is a subset of vertices $C(t)$, which is the set of DVE clients. We want to find a subtree G' of G that includes all the vertices in

$C(t)$ and the sum of the cost of the edges in the subtree G' is no more than B.

The motivation for finding the above sub-graph is to guarantee that the total edge cost is less than or equal to B so as to provide an upper bound on the network bandwidth consumption. However, the above problem is the well-known Steiner tree problem, which is known to be NP-complete[9]. Due to this reason, we propose to find the MST sub-graph based on the following procedure:

1. First, build a minimum spanning tree by any well known algorithm like Kruskal's algorithm and Prim's algorithm, the resulting tree is denoted by $G' = (V, E')$.
2. For every v nodes in G', check that whether removing v will partition G' into two parts where both parts consists of any clients in $C(t)$, if not, remove v, and the edges associated with v, from G'.
3. The resulting G' is the communication sub-graph we want.

Note that the MST sub-graph generated by the above algorithm preserves the tree structure but it does not guarantee the total edge cost to be minimum. The reason that it does not provide minimum total edge cost is that only a subset of nodes (e.g., the participating clients) are included in the sub-graph instead of all nodes in the graph G.

Chapter 5

Synchronization

In this chapter, we discuss the synchronization problem in a DVE system. In a DVE system, we have a number of DVE clients and they serve as an interface of the users to the virtual world. In order to render the virtual world for the users, the DVE clients must obtain the information about the virtual world from the DVE server. The DVE clients are also responsible to send any request from the users to the DVE server for processing. The actions taken by the users on the virtual objects inside the DVE system are sent to the DVE server by the DVE clients, too.

Since the DVE clients are processes running in different client machines with different run time environment (e.g., the current work load in the client machine, the bandwidth and latency of the network), they will generate the same sequence of views of the virtual world at *different rate*. Therefore, there are going to be some *consistency* problems among the views rendered by different client machines such that even the users of the *same* virtual world may experience a slightly different view of the virtual world at the same time.

Synchronization is the key to maintain the consistency among the views generated by the DVE clients so that the deviation of the views can be kept under an acceptable level.

We begin with some definitions and descriptions about synchronizations in a DVE system. Then, we present our system model followed by defining what is consistency by means of the system model we described. Finally, we present how to solve the synchronization problem by deriving an optimal synchronizing interval with two consistency measures based on the given set of consistency criteria.

5.1 Synchronization in a DVE System

The virtual world is populated by different virtual objects and they keep changing over time. Before we classify the objects in the virtual world, we need the following definition:

Definition 1 The state of an object is defined by its current coordinate (e.g., the $[x, y, z]$ coordinate in the virtual world) and its current property (e.g., the current color of an object, directional velocity, $\dots$, etc).

For a given object o_i, let $S_{o_i} = \{s^1_{o_i}, s^2_{o_i}, \dots\}$ be the set of all the possible states of the object o_i in the virtual world. In general, objects in the virtual world can be classified into two categories, namely *static* and *dynamic*. A static object has only one state. For example, an object representing a mountain in the virtual world is static in the sense that it will not change its coordinate in the virtual world. On the other hand, a dynamic object can have more than one state. For example, an avatar may traverse from one region to another region in the virtual world or an avatar may choose to move an object from one location to another location. Since all the users who are exploring the virtual world should share the same view, therefore, it is important that all users should have the *consistent view* of all the objects in the virtual world. For example, if a user from client C_a views an object o_i and the object o_i is in state $s^*_{o_i}$ (e.g., coordinate $[x, y, z]$), then another user from client C_b should also display object o_i in state $s^*_{o_i}$ (e.g., the same coordinate $[x, y, z]$).

To *visualize* a dynamic object, let $s_{o_i}(t)$ be the state of object o_i at time step t. If client k wants to view the dynamic object o_i, then the computer in which client k resides has to *render a sequence of states changes* for object o_i, $\{s_{o_i}(t), s_{o_i}(t+1), s_{o_i}(t+2), \dots\}$, starting from the same time step t. To maintain a consistent view for all the clients in the virtual world, this implies that every computers where the clients reside have to *render* the same sequence of state changes, starting from time step t. Note that in general, an absolute consistent view of a dynamic object may not be possible due to the following reasons:

- Not all client can start the rendering of state sequence at the same time step t. This may be caused by the network delay variation of delivering the "start-rendering" message to all the clients.
- Different computer may render the state sequence at different rate. This is due

to the fact that different computers may have different work load and therefore, some computers may miss the rendering operation at some time steps.

Let $c^l_{o_i}(t)$ be the state of object o_i at time step t that client l is rendering. We have the following definition:

Definition 2 If $c^l_{o_i}(t) = s^k_{o_i}$ and $c^m_{o_i}(t+\tau) = s^k_{o_i}$, then we say that the phase difference between client l and client m, denotes by $\Phi(l, m)$, is τ.

For example, assume that the set of all the possible states of object o_i is $S_{o_i} = \{s^1_{o_i}, s^2_{o_i}, \ldots, s^{10}_{o_i}\}$ and the rendering sequence of object o_i is $Seq = \{s^1_{o_i}, s^3_{o_i}, s^5_{o_i}, s^7_{o_i}, s^9_{o_i}\}$. If client C_A starts the rendering sequence at time step 0 and client C_B starts the same rendering sequence at time step 3, then the phase difference $\Phi(A, B) = 3$.

The *object synchronization procedure* in the DVE system is a process to guarantee that the absolute value of the phase difference between any clients of any object in the virtual world to be less than a pre-defined system threshold Φ. Some of the approaches to maintain object state consistency are:

- **Aggressive Object Synchronization Technique (AOS):** Under the AOS approach, each object o_i is assigned to a master process P_i. For every small period δt, process P_i will broadcast the state of the object o_i to all participating clients in the DVE system. Upon receiving the state information of object o_i, every client will render the state of o_i. Figure 5.1 illustrates the AOS synchronization technique. A major disadvantage of this approach is that there will be a high volume of synchronization message traffic and this will lead to scalability problem in the DVE system. Note that even though the process P_i will broadcast a synchronization message every δt time unit, the synchronization can still be off by d_{max} time unit due to the network delay of transmitting the synchronization message.

- **Dead Reckoning Synchronization Technique (DRS):** In the AOS technique, extremely high network bandwidth will be consumed. However, it is important to observe that in most cases, the state of an object at the current time step is very *similar* to the state of the object in the previous time step. Therefore, it is not necessary for the process P_i to send a synchronization message every δt. Consider the following example, if a ball is experiencing a free fall from a 1000 feet tall building and if the initial position of the ball is known to all

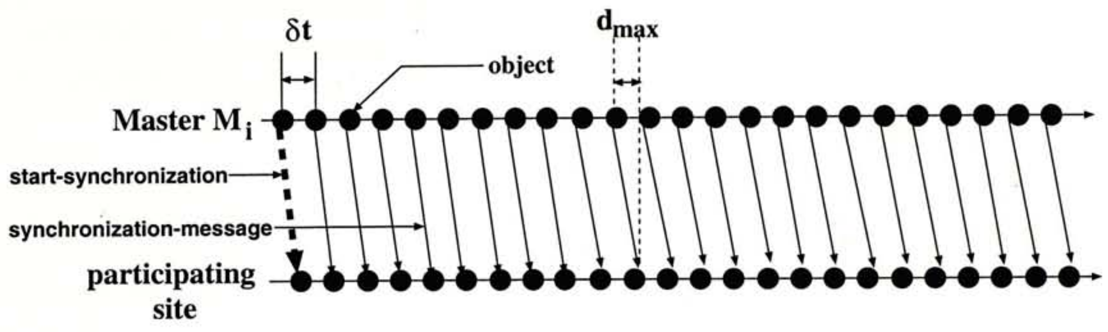

Figure 5.1: Aggressive Object Synchronization (AOS) Technique with d_{max} being the Maximum Network Delay

clients, then every participating sites should be able to calculate the *trajectory* of this ball and render the corresponding sequence of state changes. That is, every participating machines will render the object's coordinate for every time step and update the object's state in its local machine. This is the main idea about the *dead-reckoning* (DRS) technique [5] in which the initial state as well as the equation governing the trajectory of the object are available to all the participating machines and each machine will perform the local computation and will update its local display. The DRS technique is illustrated in Figure 5.2 where the `start-synchronization` message also encodes the object's initial position and directional velocity, for example.

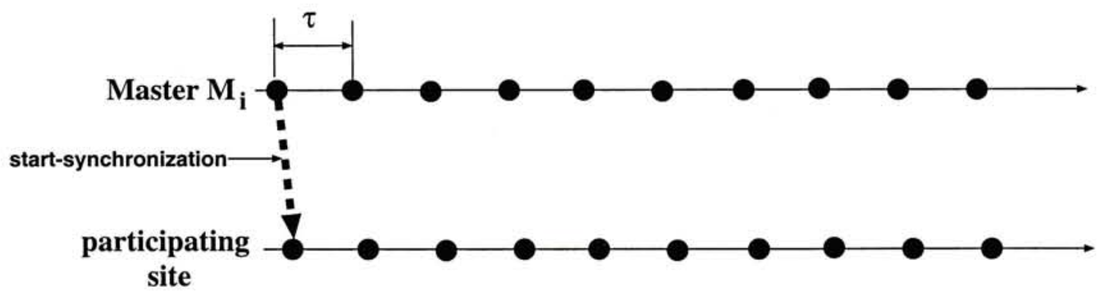

Figure 5.2: Dead Reckoning Synchronization Technique

- **Dead Reckoning with Periodic Synchronization Message (DRPS):** Note that even if the DRS technique is used, it is still necessary for the master process P_i of object o_i to send the synchronization message from time to time. This situation occurs if there is an external event that will affect the trajectory of the object o_i (e.g., a client catches the free-falling ball). Another reason why we need to send a synchronization message periodically is due to the fluctuation of work load at the participating computers, some computers may not be able to render the next state of the object on time and if it is not adjusted, this will create view inconsistency in the long run. Therefore, for the DRPS technique, it employs the dead-reckoning technique and for every synchronization interval, $n\tau$, a "synchronization message" is sent. Upon receiving the synchronization message, each computer will immediately update the state of the object (or in effect, the phase difference is reset to zero). The DRPS synchronization technique is depicted in Figure 5.3.

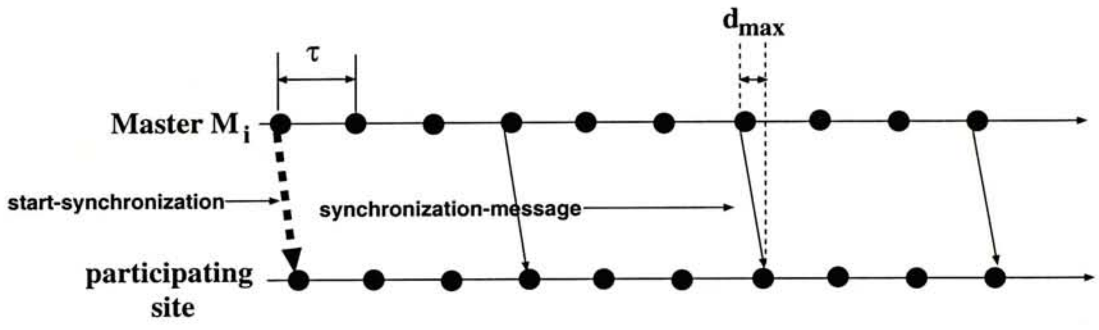

Figure 5.3: Dead Reckoning Synchronization Technique with Synchronization Message and d_{max} being the Maximum Network Delay

Note that in general, the longer the value of synchronization interval, the phase difference between any two clients will be higher, this imply that the *view consistency* between these two clients may be un-acceptable. On the other hand, if the synchronization message is sent too often (or the synchronization interval is too small), the DVE system needs to send too many synchronization messages to all the participating clients and it will consume too many network bandwidth.

5.2 System Model

We have two DVE clients in two different computing nodes namely C_a and C_b, in a communication network with maximum delay d_{max}. The computation of C_a and C_b can be described by their *phase*. Specifically, C_a and C_b start from phase 0 and then they compute phase i based on phase $i-1$ (for $i \geq 1$) together with the user inputs, if any. Due to the fluctuation of the work load on the computing nodes, we have p_a as the probability for the computing node C_a to fail to calculate the next phase. We have p_b for the computing node C_b similary. The values of p_a and p_b can be obtained by statistics inside the kernel.

We define the phase difference of C_a and C_b at time t as $\Phi(C_a, C_b) = i-j$, assuming that C_a is in phase i and C_b is in phase j at time t. It is clear that C_a and C_b may not be in the same phase after a period of time, synchronization is required to maintain the consistency of the phases. We choose C_a as the *master process* and it is responsible for sending the synchronization message to C_b to maintain the consistency.

5.2.1 Problem Definition

Our goal is to ensure that the phase difference of the system is not exceeding the threshold Φ which is a system parameter defined based on the nature of the underlying application as described in Section 5.2.3.

We assume that $\Phi(C_a, C_b)$ will be reset to zero if C_a sends its current phase to C_b, for simplicity. [1] Our problem is then to find out how often should C_a send synchronization messages to C_b with a set of consistency and system parameters.

5.2.2 The Markov Chain Model

We use a Markov Chain to model the system with its states as $\Phi(C_a, C_b)$. The computing nodes have to update their phase every Δt time units, for example, we want $\Delta t = \frac{1}{30}$s for a smooth animation. In general, the computing nodes can make $L = \lceil \frac{d_{max}}{\Delta t} \rceil$ phase changes between successive synchronization messages if d_{max} is the maximum network delay.

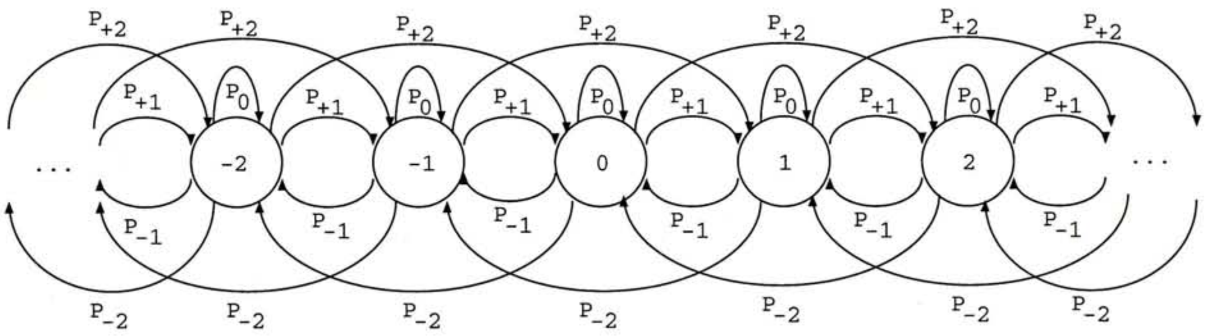

Figure 5.4: The Markov Chain with $L = 2$

The Markov Chain with $L = 2$ is shown in Figure 5.4. In each state i, it can make a transition back to state i with probability p_0, or make transitions to state $i \pm 1$ and state $i \pm 2$ with probability $p_{\pm 1}$ and $p_{\pm 2}$ respectively, after two phase changes in each of the computing node.

In general, we define p_0 to be the probability of the system to stay in the same

[1]Simulation in Section 5.4 reveals that this assumption is a good approximation to the case with the consideration of the network delay.

state after L phase changes in each of the computing node, p_{+k} and p_{-k} to be the probability of the system to advance or to go back k states after L phase changes in the each of the computing node, and obviously, we have $|k| \leq L$ for a system with L steps transition.

We first give the probability generating function for the calculation of p_0 with L steps transition. The intuitive idea is to sum up the probability of all the possible cases within a L phase changes which result in a zero net transition. To have zero net transition, we should have l phase changes combinations which result in an increase in the phase difference, and another l phase changes combinations which result in an equal decrease in the phase difference. Obviously, we should have $0 \leq l \leq \lfloor \frac{L}{2} \rfloor$ so that we can ensure that the number of forward moves and backward moves can cancel out the effect of each others, and the rest of the phase changes combinations of the computing nodes left by these $2l$ phase changes should not generate any further change in the phase difference. The probability generating function for L steps transition for p_0 is therefore:

$$p_0 = \sum_{l=0}^{\lfloor \frac{L}{2} \rfloor} \left[\frac{L!}{l!l!(L-2l)!} (p_F^l)(p_B^l)(p_S^{L-2l}) \right] \tag{5.1}$$

where $p_F = (1-p_a)p_b$, is the probability of $\Phi(C_a, C_b)$ being increased by one, that is, C_a can compute the next phase and C_b cannot, in one Δt; $p_B = p_a(1-p_b)$ is the probability of $\Phi(C_a, C_b)$ being decreased by one, that is C_b can compute the next phase on time and C_a cannot in one Δt; $p_S = p_a p_b + (1-p_a)(1-p_b)$ is the probability of $\Phi(C_a, C_b)$ being unchanged, that is, when both of the nodes either success or fail, in one Δt, for the computation of the next phase.

Similarly, the probability generating function for going forward k steps after L phase changes in each of the computing node can be generalized as follow, with the upper limit on the number of forward steps reduced to ensure there will be sufficient backward steps left to cancel out its effect such that the net forward transition is exactly k:

$$p_{+k} = \sum_{l=0}^{\lfloor \frac{L-k}{2} \rfloor} \left[\frac{L!}{(l+k)!l!(L-2l-k)!} (p_F^{l+k})(p_B^l)(p_S^{L-2l-k}) \right] \tag{5.2}$$

And the probability generating function for going backward k steps after L phase changes in each of the computing node, can be generalized similarly as follow:

$$p_{-k} = \sum_{l=0}^{\lfloor \frac{L-k}{2} \rfloor} \left[\frac{L!}{(l+k)!l!(L-2l-k)!} (p_B^{l+k})(p_F^l)(p_S^{L-2l-k}) \right] \tag{5.3}$$

The transition probability matrix $\mathbf{Q}$ for the Markov Chain with L steps transitions can therefore, be specified by:

$$q_{ij} = \begin{cases} p_0 & \text{for } i = j \\ p_{+k} & \text{for } i + k = j, L \geq k > 0 \\ p_{-k} & \text{for } i - k = j, L \geq k > 0 \\ 0 & \text{otherwise} \end{cases} \tag{5.4}$$

5.2.3 Deciding the Threshold Φ

We show how to decide the value of the threshold Φ by giving a brief description on an application example. The main idea is that we should choose a threshold such that the *causal relationship* of the objects state changes are preserved upon synchronization actions.

We use a table tennis game as an example. In this game, the only critical causal relationship among the actions are the smovement of the ball and the action of the player on the ball applied through the bat. Suppose even when the ball travels at its maximum speed v_{max}, it requires t seconds to travel from one end of the table to another end of the table of length d (or to be within the reach of another player from one player), where t can be estimated by the simple formula: $t = \frac{d}{v_{max}}$.

In order to provide smooth animation, we have to render the scene every $\frac{1}{30}$s, and it is also the time required for a phase change. To maintain the the position of the ball to be consistent with respect to the causal relationship, we have to set the threshold $\Phi = t \div \frac{1}{30} = \frac{30d}{v_{max}}$ or even smaller, since in the worst case, the ball would reach another player suddenly (because all the intermediate phases are skipped) and thus one might not be able to react to the ball. The above formula provides a guideline to estimate the largest possible threshold required for a particular application by considering the characteristics like the maximum speed of the objects, the inter-avatar distances, etc.

5.3 Optimal Synchronizing Interval

5.3.1 An "on-average" Guarantee

To answer the question of how often (e.g. how many time steps) a "synchronization-message" should be sent such that the expected value of the absolute phase difference between clients C_A and C_B is less than or equal to Φ. We use the theory of the fundamental matrix and analyze the underlying Markov chain $\mathcal{M}$. Let state s_i, $i \geq 0$ represents the value of the phase difference i. We aggregate[3] all those states $s_i, i > \Phi$, as one state and we make it a trap state [2] and we have a new Markov chain $\mathcal{M}'$ with an associated transition probability matrix $\mathbf{P}'$ as:

$$\mathbf{P} = \left(\begin{array}{ccccc|c}
p_0 & p_{+1} & p_{+2} & \cdots & p_{+2\Phi} & 1-\sum_{i=0}^{2\Phi} P_{0i} \\
p_{-1} & p_0 & p_{+1} & \cdots & p_{+(2\Phi-1)} & 1-\sum_{i=0}^{2\Phi} P_{1i} \\
p_{-2} & p_{-1} & p_0 & \cdots & p_{+(2\Phi-2)} & 1-\sum_{i=0}^{2\Phi} P_{2i} \\
\vdots & \vdots & \vdots & & \vdots & \vdots \\
p_{-\Phi} & \vdots & \vdots & & p_{+\Phi} & 1-\sum_{i=0}^{2\Phi} P_{\Phi i} \\
\vdots & \vdots & \vdots & & \vdots & \vdots \\
p_{-(2\Phi-2)} & p_{-(2\Phi-3)} & p_{-(2\Phi-4)} & \cdots & p_{+2} & 1-\sum_{i=0}^{2\Phi} P_{2\Phi-2i} \\
p_{-(2\Phi-1)} & p_{-(2\Phi-2)} & p_{-(2\Phi-3)} & \cdots & p_{+1} & 1-\sum_{i=0}^{2\Phi} P_{2\Phi-1i} \\
p_{-2\Phi} & p_{-(2\Phi-1)} & p_{-(2\Phi-2)} & \cdots & p_0 & 1-\sum_{i=0}^{2\Phi} P_{2\Phi i} \\
\hline
0 & 0 & 0 & \ldots & 0 & 1
\end{array}\right)$$

The new Markov chain has $2\Phi+2$ states with the last state being the trap state. We can partition P as:

$$\mathbf{P}' = \left(\begin{array}{c|c} \mathbf{Q} & \mathbf{C} \\ \hline \mathbf{0} & 1 \end{array}\right)$$

where $\mathbf{Q}$ is an sub-stochastic matrix describing the transition probability between the $2\Phi+1$ states, $\mathbf{C}$ is a column vector representing the transition probabilities between the $2\Phi+1$ transient states to the trap state and $\mathbf{0}$ is a row vector of $2\Phi+1$ zeros, that means once the system has reached the trap state, it has no way to leave. We define

[2] A trap state is a state where in once the system reaches that state, the system stays in that state forever.

the matrix $\mathbf{M}$ (also known as the fundamental matrix) as:

$$\mathbf{M} = (\mathbf{I}-\mathbf{Q})^{-1} = \mathbf{I}+\mathbf{Q}+\mathbf{Q}^2+\mathbf{Q}^3+\cdots = \sum_{k=0}^{\infty}\mathbf{Q}^k \tag{5.5}$$

To compute the average number of time steps (or transition) so that the absolute value of the phase difference in $\mathcal{M}$ is greater than Φ, we let X_{ij} $(0 \le i,j \le \Phi)$ be the random variable denoting the number of time steps that the Markov chain $\mathcal{M}'$ visits state j before entering the absorbing state, given that it started in state i. Let $E[X_{ij}]$ be the expected value of X_{ij}. We have the following theorem

Theorem 3 For $0 \le i,j \le 2\Phi+2$, we have $E[X_{ij}] = m_{ij}$ where m_{ij} is the (i,j) element of the fundamental matrix $\mathbf{M}$.

Proof: Please refer to Bhat [3]. ∎

Remark: Note that the above theorem indicates that if the system starts at state i (the absolute value of the phase difference is equal to i), then we know on average it takes m_{ij} time steps for the system to enter the trap state (a state representing the absolute value of the phase difference is greater than Φ).

Corollary 1 Assume that the phase difference between client C_A and C_B is zero initially, then on average, after t^* time steps, the expected value of the absolute phase difference between client C_A and C_B is greater than Φ. We can express t^* as:

$$t^* = \sum_{k=0}^{2\Phi} m_{0k} \tag{5.6}$$

Proof: We can show this by directly applying Theorem 3. ∎

Corollary 2 To ensure that the expected value of the absolute phase difference between client C_A and C_B is less than or equal to Φ for all time steps, the master process P_{o_i} of object o_i needs to send the "synchronization-message" every $t^* = \sum_{k=0}^{2\Phi} m_{0k}$ time steps.

Proof: This can be directly observed by applying Corollary 1. ■

Remark: Note that once the "synchronization-message" is received by all clients, the phase difference is reset to zero. Therefore, in order to maintain the phase difference between C_A and C_B to stay within Φ, the master process P_{o_i} needs to periodically broadcast the synchronization message and the period is $t^* = \sum_{k=0}^{2\Phi} m_{0k}$.

5.3.2 A Stochastic Guarantee

We are going to derive the optimal synchronizing interval τ_p^*, with a given value of threshold Φ such that the system will stay in states not exceeding Φ with probability p. It is a generalization of the problem described in [15].

However, direct derivation of the value of τ_p^* is difficult and so we try to find it indirectly. We find that it is easier to find the value of p for a system with a given τ and Φ. Therefore, we can solve the problem by searching. More formally, we derive a function $p = f_M(\tau, \Phi)$ for system M, and it returns the corresponding p, which is the probability of the system to be in state not exceeding Φ, if synchronization messages were sent every τ time unit; then we try to find the optimal τ_p^* which satisfy our consistency requirements with the searching techniques in Section 5.3.4.

5.3.3 Finding p with τ and Φ

Finding the exact value of p with given τ and Φ is difficult as well because the state space of the underlying Markov Chain is infinite as shown in Figure 5.5 for a system with $L = 1$.

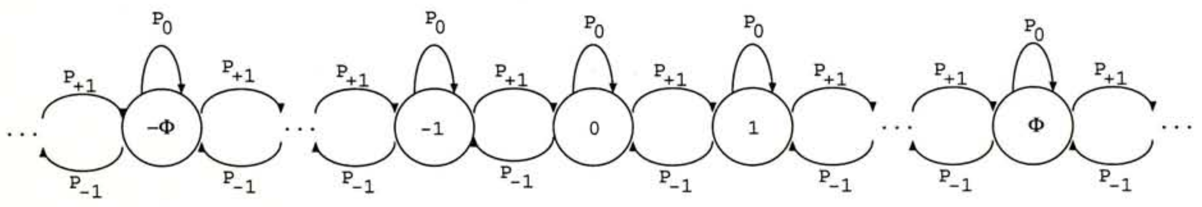

Figure 5.5: The Markov Chain with Infinite State Space

Therefore, we suggest the following method to find an approximation of the value of p with given τ and Φ. With the observation that synchronization action would be applied to the system, the Markov chain will be in most of the time, staying in states i with a small $|i|$. To obtain an approximation of the infinite state system, we chop off the Markov Chain at states $\pm(\Phi + 1)$, and then we aggregate the chopped states by two new states namely $\pm B$, as in Figure 5.6.

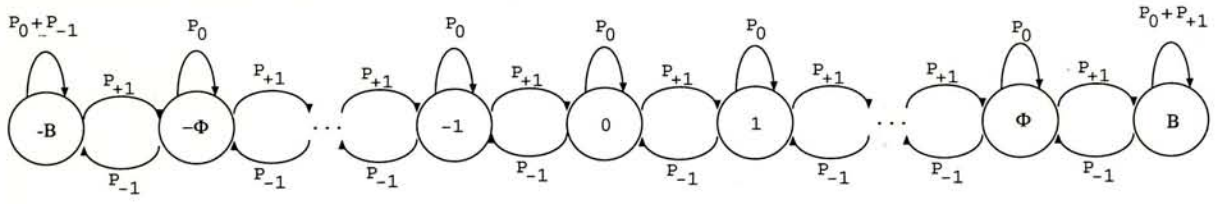

Figure 5.6: The Chopped Markov Chain

In the general case with L steps transitions, we try to redirect the traffic going from any states to the chopped states, to the two new states $\pm B$ and then we also redirect the traffic going from states $\pm B$ to the chopped states back to states $\pm B$. We call the corresponding transition probability matrix of the above chopped Markov Chain as $\mathbf{Q}$. The next task is to incorporate the synchronization action into $\mathbf{Q}$. To do this, we modify $\mathbf{Q}$ according to τ. The intuitive idea of this transformation is based on the following two facts:

1. We have $(1 - \frac{1}{\tau})$ chance to make transitions as usual, for example, if we have probability p_{+k} to go to state k from state 0 originally, we now have probability $(1 - \frac{1}{\tau}) \times p_{+k}$.
2. In addition, we have $\frac{1}{\tau}$ more chance to go to the state 0 from each state due to synchronization, for example, if we have probability p_{+k} to transit to state 0 from state k originally, we now have probability $\frac{1}{\tau} + (1 - \frac{1}{\tau})p_{+k}$.

Formally, we have the following new transition probability matrix **Q'** with synchronization:

$$\mathbf{q'_{ij}} = \begin{cases} \frac{1}{\tau} + (1 - \frac{1}{\tau})\mathbf{q}_{ij} & \text{if } j = 0 \\ (1 - \frac{1}{\tau})\mathbf{q}_{ij} & \text{otherwise} \end{cases} \tag{5.7}$$

Now, we can calculate the residence probability of the states of the chopped Markov Chain with synchronization action, by solving the following system of equations:

$$\begin{cases} \mathbf{Q'} \times \Pi = \Pi \\ \sum_i \Pi_i = 1 \end{cases} \tag{5.8}$$

After finding Π, p_τ can be obtained by the following formula:

$$
\begin{aligned}
p_\tau &= \sum_{i=-\Phi}^{\Phi} \Pi_i && (5.9)\\
&= 1 - (\Pi_{-B} + \Pi_B) && (5.10)
\end{aligned}
$$

which is the approximation of the probability for the system to be in states not exceeding the threshold with the synchronizing interval τ. And it is intuitive that this approximation would be better if we try to chopped off the infinite state Markov chain at states c with a larger c rather than $\Phi + 1$.

With the above derivations, we can calculate the value of p_τ from a given τ and Φ with a Markov Chain $\mathcal{M}$. In another words, we have the function $p_\tau = f_M(\tau, \Phi)$.

5.3.4 Searching for τ_p^*

We have the function $p_\tau = f_M(\tau, \Phi)$ derived in previous section. Now, we propose two searching techniques for finding τ_p^* with a given consistency criteria, that is, finding the optimal synchronizing interval to guarantee that the system would be in state not exceeding the threshold Φ with probability p. The first technique is the *simple iteration* technique, we give the algorithm formally as below:

```
function si_find_interval(double p) :int;
begin
    $\tau = 1$;
    if $f_M(\tau, \Phi) < p$ then return -1; (* impossible $p$ *)
    $\tau = \tau + 1$;
    (* Find the maximum $\tau$ to satisfy $p$ *)
    while $f_M(\tau, \Phi) > p$ do $\tau = \tau + 1$;
    (* Return the largest one without violating $p$ *)
    return $\tau - 1$;
end
```

This technique is efficient if the optimal synchronizing interval τ_p^* is small, and to deal with system with larger value of τ_p^*, we propose a second technique called the *step searching* technique, we give the algorithm formally as below:

```
function step_find_interval(double p, int step) :int;
begin
    $\tau = 1$;
    if $f_M(\tau, \Phi) < p$ then return -1; (* impossible $p$ *)
    if step $< 1$ then step $= 1$; (* step should greater than zero * )
    $\tau = \tau +$ step;
    (* Find a $\tau$ to violate $p$ *)
    while $f_M(\tau, \Phi) > p$ do $\tau = \tau +$ step;
    (* Search backward to get the largest one without violating $p$ *)
    while $f_M(\tau, \Phi) < p$ do $\tau = \tau - 1$;
    (* Return the one which first match the consistency criteria *)
    return $\tau + 1$;
end
```

The parameter "step" is used to adjust the jumping size of the forward search, and a larger value should be used for larger values of the threshold.

5.4 Experiments

This section is divided into two parts. In the first part, simulation results are presented to show the influences of changing various system parameters on the effectiveness of the synchronization actions. Then, a simulation has been conducted to justify our assumption that $\Phi(C_a, C_b)$ can be reset to zero when C_a sends its current phase to C_b. In the second part, we show the synchronizing interval derived by our theoretical model and the corresponding results obtained from simulations are shown in parallel for comparison.

5.4.1 Simulation Results

We begin by giving a brief description on our simulation program. The simulation program is written in C with the following input parameters:

1. p_a, p_b: the probabilities of missing a computation of the next phase for the two computing nodes

2. d_{max}: the maximum network delay

3. τ: the synchronizing interval

4. Φ: the threshold (in the number of phase difference)

Each of the simulation runs for one million iterations, and it is equivalent to a DVE session which last for about ten hours. The fluctuation of the work load on the computing nodes were modeled by using the random number generation function in the C library.

On an Example System

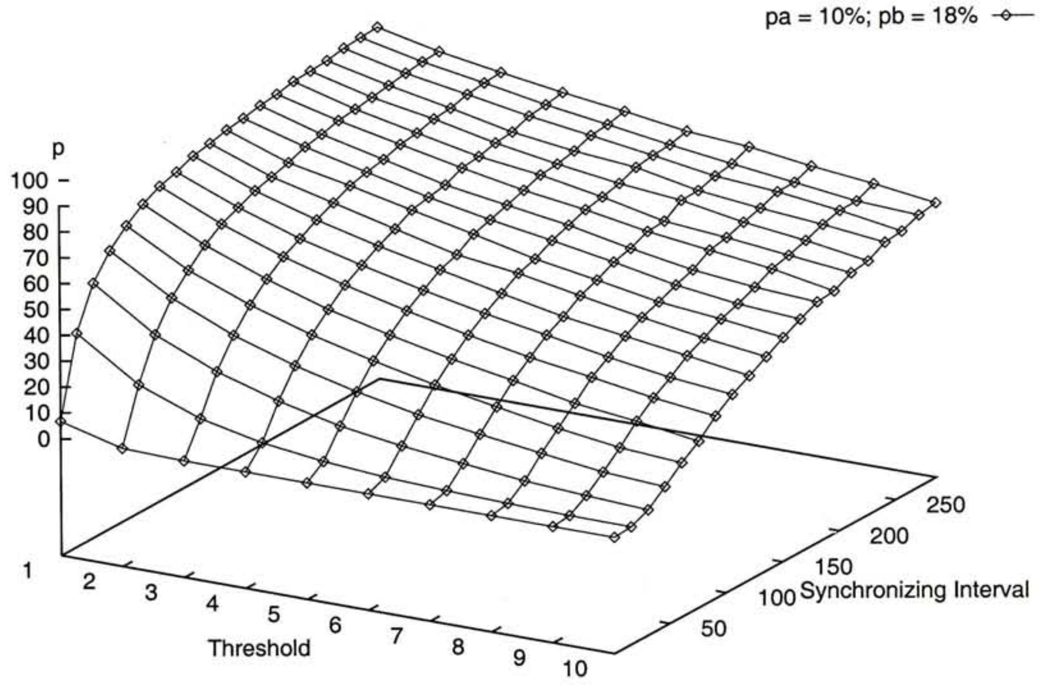

Figure 5.7: Simulations of an Example System

Simulations were conducted with the values of threshold τ ranged from 1 to 10, the values of p_a and p_b are set to 10% and 18% respectively, and the synchronizing interval ranged from 1 to 300 time units are used. The value of the maximum network delay (d_{max}) of 5 is used.

We choose the values of p_a and p_b to be rather high but not too high because we would like to illustrate the effect of the synchronization actions. Too low the values of p_a and p_b making synchronization action insignificant, too high the values make the system non-realistic since machines with too high a load is not suitable for DVE application at all.

To provide a good quality animation inside the DVE, we need 30 frames per second, and therefore one time unit in the system is approximately $\frac{1}{30}$s. With a 10Mbps network, we can transmit a maximum of 41.6k bytes per time unit and therefore each synchronizing message can have a size up to 41.6k bytes at most, in this network setting with the delay of 5.

In Figure 5.7, the probability p of the system with the phase difference exceeding the threshold is shown with different values of the threshold Φ and the synchronizing interval τ. Notice that as the value of synchronizing interval τ increases, the system becomes less synchronized, regardless of the value of the threshold Φ used. In addition, the larger the value of the threshold Φ used, the more effective the synchronization action is.

Effect of the Skewness on the System Loadings

Difference in p_a and p_b	p_a	p_b	d_{max}	threshold Φ
D = 0%	10%	10%	5	5
D = 0%	90%	90%	5	5
D = 20%	10%	30%	5	5
D = 20%	30%	50%	5	5
D = 40%	10%	50%	5	5
D = 40%	50%	90%	5	5
D = 80%	10%	90%	5	5

Table 5.1: Parameters used in the Simulation in Figure 5.8

The probability of the system to be in states exceeding the threshold against the synchronizing intervals ranged from 1 to 300 are shown in Figure 5.8, and the parameters used for the simulations are shown in Table 5.1.

We can see that simulations with the same value of D behave similarly and the larger the skewness of the system loadings, the smaller the synchronizing interval should be used to maintain a good consistency. This indicates that the synchronization action is less effective in a system with large difference in p_a and p_b.

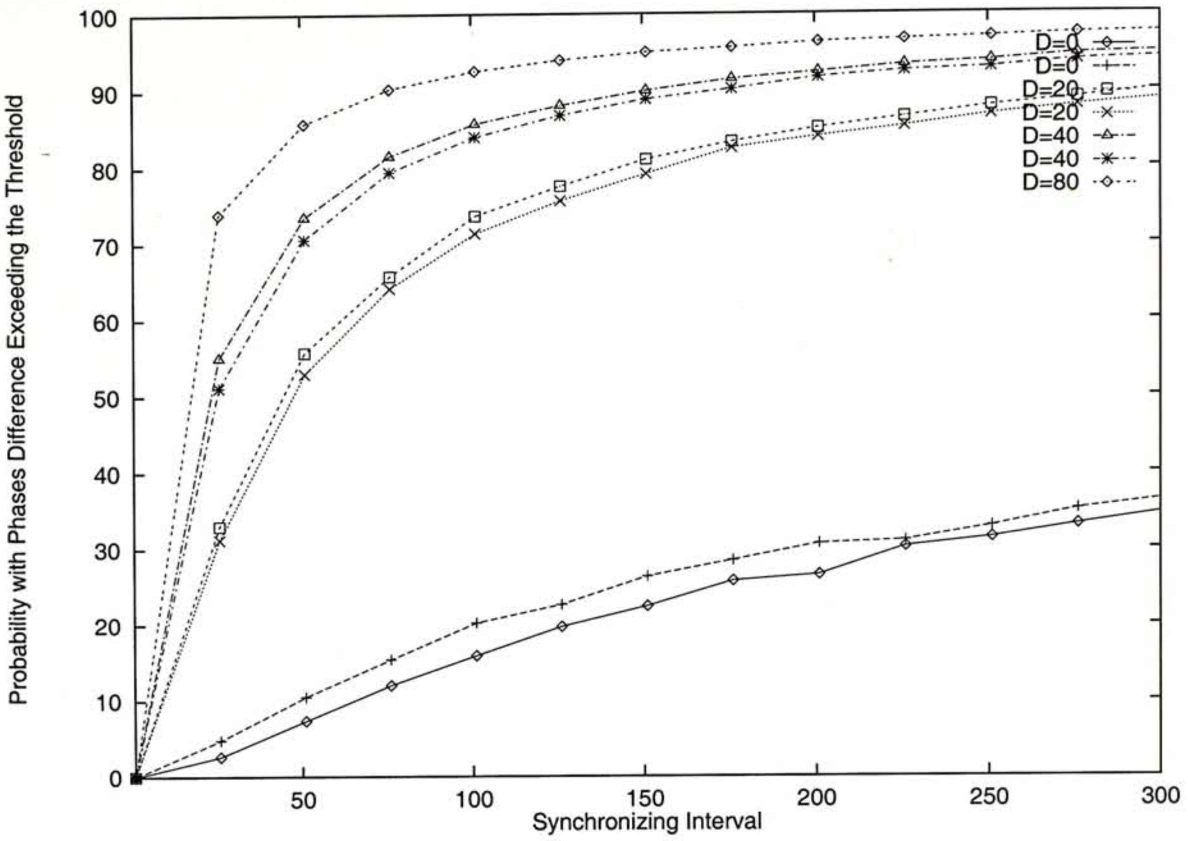

Figure 5.8: Simulation Results Showing the Effect on the Skewness of the System Loadings

With or without Network Delay

This simulation has been conducted to justify our assumption that $\Phi(C_a, C_b)$ can simply be reset to zero if C_a sends its current phase to C_b. The simulation result is obtained by using $d_{max} = 20$ and it is plotted in Figure 5.9. In the figure, we can see that the average phase differences in one million iterations, either with or without considering network delay, is very similar to each others and thus our assumption is valid and good approximation to the real case with network delay.

5.4.2 Theoretical Results

We have implemented a program in Mathematica and another program in C to calculate the optimal synchronizing intervals based on the theory and the two different methods presented in Section 5.3.

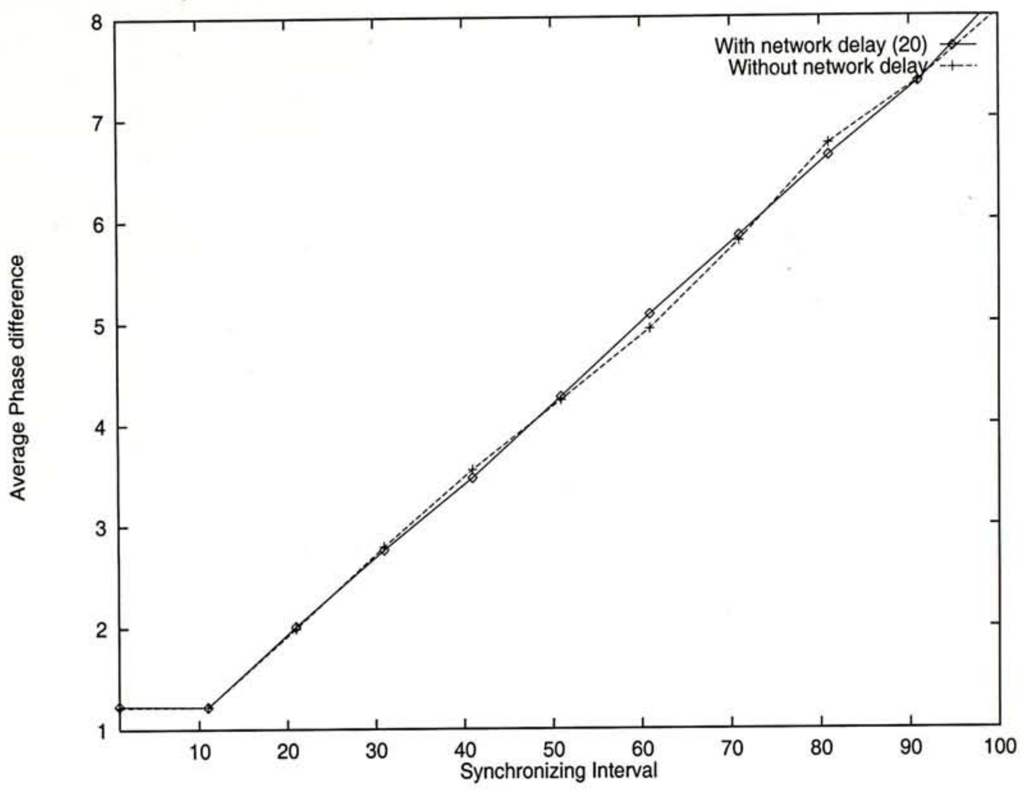

Figure 5.9: The Difference with or without Network Delay

Optimal Synchronizing Interval for an "on-average" Guarantee

In the first experiment, we have calculated the optimal synchronizing interval for different values of the threshold by using the theory of fundamental matrix to give an "on-average" guarantee. We have used a system with $p_a = 0.1$ and $p_b = 0.18$, the maximum network delay d_{max} is set to 5. The results are plotted in Figure 5.10.

Notice that we can see a linear relationship for the optimal synchronizing interval and the value of the threshold. It indicates that the synchronization mechanism is efficient and can be scaled up well.

Effect on the Difference in p_a and p_b

In the second experiment, we investigate the relationship of the difference in p_a and p_b, the optimal synchronizing interval and the threshold. In this experiment, p_a is fixed at 0.1 and p_b varies from 0.2 to 0.8, values of the threshold ranged from 5 to 10 are used. The results are plotted in Figure 5.11.

In general, the larger the difference in p_0 and p_1, the smaller the synchronizing interval is required for a given value of threshold. It is because the skewness of p_0 and

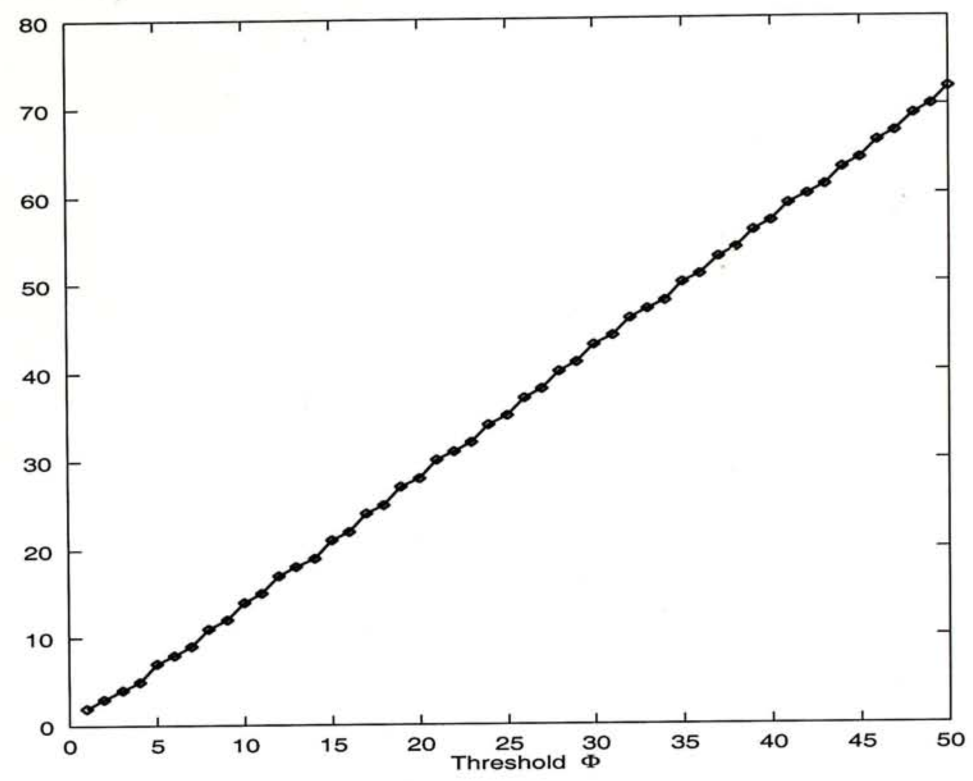

Figure 5.10: Optimal Synchronizing Interval Derived for an "on-average" Guarantee

p_1 specify the difference of the behavior of the two computing nodes, which reflects the difference in the rate of phase calculations.

Effect on the Absolute Magnitude of p_a and p_b

In the third experiment, we investigate the effect on the absolute magnitude of p_a and p_b on the optimal synchronizing interval and the values of the threshold. Figure 5.12 is obtained by values of p_a ranged from 0.1 to 0.8, where the corresponding p_b used can be obtained by adding 0.1 to p_a. Different values of threshold (from 5 to 10) are plotted.

Notice that when p_a and p_b approaches to 0.5, the smallest synchronizing interval is required, and this is consistent with what we observed as in the experiment about the skewness in p_a and p_b in Section 5.4.1.

Optimal Synchronizing Interval for the Stochastic Guarantee

We have calculated the optimal synchronizing intervals for a system with the values of p_a and p_b of 10% and 18% respectively. The value of the d_{max} is set to 5, and we

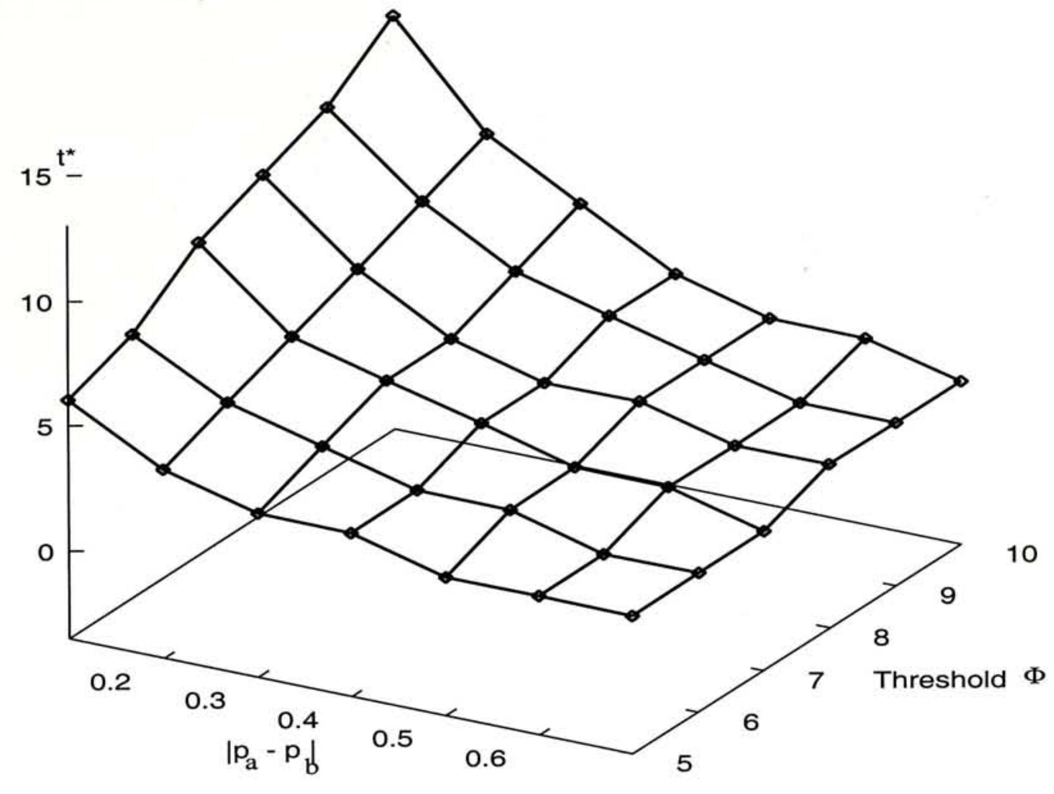

Figure 5.11: Effect on the Value of $|p_a - p_b|$

want the system to be with the phase difference under the threshold (ranged from 1 to 10) in 90% of time. The results are shown in Table 5.2.

Notice that the value of τ derived theoretically is always sufficient to maintain the required consistency requirement specified in the derivation of the optimal synchronizing intervals.

The bound derived is not very tight, it is because the effect of the network delay is not incorporated in the theoretical model, we have made an assumption that the phase difference of the two computing nodes will be reset to zero when the synchronization message arrives. In a system with a small value of threshold, this can create a noticeable effect. It is also worth to know that the synchronizing interval should be adjusted in a stepwise manner, and therefore, we can only use a synchronizing interval which is a little bit over-shoot to the original specification.

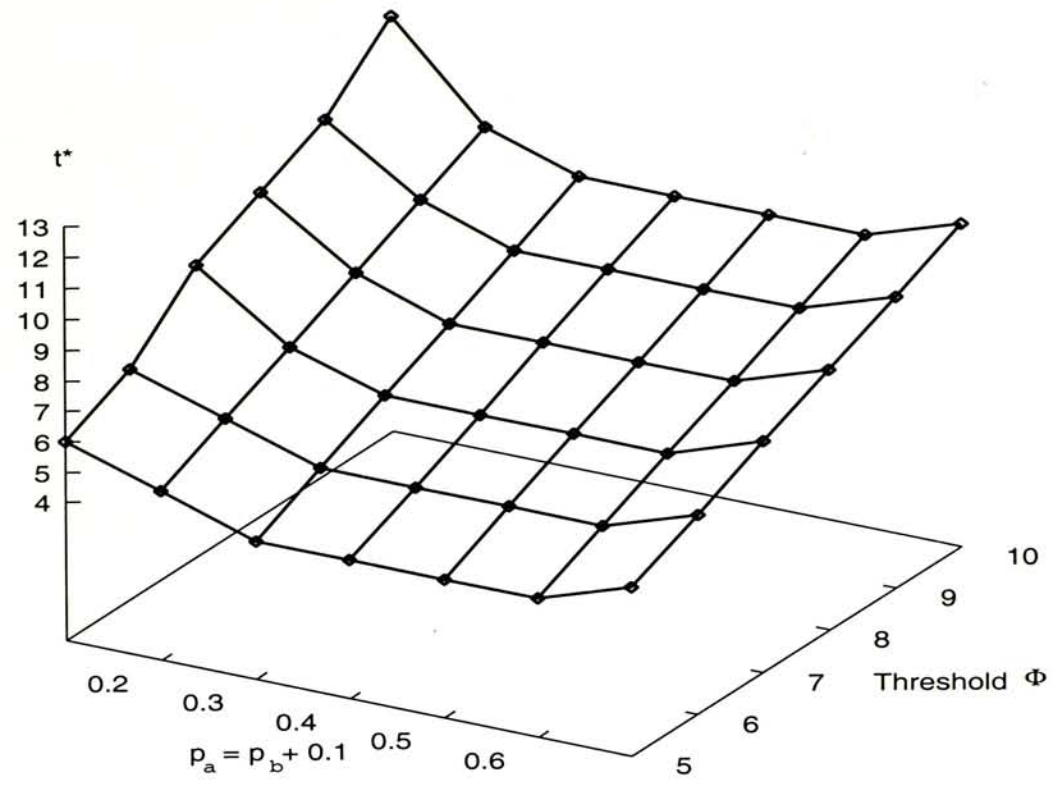

Figure 5.12: Effect on the Absolute Magnitude of p_a and p_b

Φ	$\tau^*_{90\%}$	p form simulation	p in theory
1	1	6.73%	0.00%
2	1	6.73%	0.00%
3	2	1.75%	6.58%
4	3	0.23%	7.53%
5	4	0.11%	7.67%
6	5	0.41%	9.96%
7	7	0.21%	9.62%
8	8	0.12%	9.15%
9	9	0.25%	8.77%
10	11	0.14%	9.85%

Table 5.2: Comparisons of Simulations and Theoretical Results

Chapter 6

Related Work

In this chapter, we describe some of the related work about DVE system research. This is by no means a complete survey since the research of DVE is one of the hottest topics in field of distributed computing, distributed multimedia, graphics and high speed networking. Here we present some of the work which are closely related to the problems we have mentioned in this text. Finally, we describe two example DVE systems at the end of this chapter.

6.1 Load Balancing on DVE

To the best of our knowledge, there is no other published literature on the load balancing aspect of a DVE system. Load balancing is the key to allow an efficient utilization of the available computing resources in a distributed system [6].

We have formulated the load balancing problem in a DVE system as a partitioning problem in [14] by using the concept of AOI described in [17]. Further improvement over our proposed solutions can be found in [12].

6.2 Object State Synchronization Techniques

One of the very important features of a DVE system is that we should provide a consistent view for different users in the same virtual world. Synchronizations are required, however, we do not need to keep all the objects state synchronized. For example, when a user is looking forward, we will not need to synchronize the state of the objects behind the user. We may also use the concept of the AOI for data filtering [19]. The basic principle is to filter all the irrelevant message exchange.

Another technique called the *dead-reckoning* is proposed in [5]. It can be used to reduce the amount of network traffic required for the exchange of the state information of the objects. With the observation that the trajectory of some objects can be calculated independently in different DVE clients, we can simply send the trajectory (e.g. in the form of a mathematical formula) with the initial state of the objects, instead of sending their new states through the network after every update in the master process.

6.3 Group Communication and Multicasting

Since the DVE clients of the same virtual world session share many common objects in their views, the state information of these common objects have to be sent to all these DVE clients. Multicasting is the natural choice in the propagation of these state information.

Multicasting in a Distributed Interactive Simulation (DIS) has been studied in [25]. It can be readily applied to the DVE systems because of the high similarity of the two application areas.

Other work about multicasting but not specific to the DVE systems include the studies of the *dynamic light weight group* in [11], the maintenance of the *total ordering* of the multicast messages [10], etc. The concept of the dynamic light weight group can be used in a DVE system to deal with the dynamic membership of the DVE clients. The maintenance of the *total ordering* on the receiver side of the multicast can be used to ensure the *casual* relations on the effect of the state changes in the virtual world. Other work on real-time communication like the RTP [22] and RSVP [27, 4] can also be employed to make efficient use of the available network resources with guaranteed performance.

6.4 DVE System Development Toolkits

Distributed Interactive Virtual Environment (DIVE) is a toolkit for building distributed virtual reality applications in a heterogeneous networking environment. It is developed by the Swedish Institute of Computer Science. The DIVE consists of a set of processes, running on different nodes within a network. These processes can access to a number of databases which they can update concurrently. Each database contains the information of the virtual objects in the virtual world. Different virtual world session is associated with a process group so that multicasting protocols are used for the communication within the group. Besides the DIVE, other examples include: the NPSNET [16], the SIMNET [7], the SPLINE [1], etc.

The research and development of a multimedia storage server in [8, 20] can be used to allow the efficient access of the data in a DVE system.

6.5 Example DVE Systems

The VINCENT system developed by the CUHK in [13] is implemented to show how spatial queries can be incorporated in a highly graphical three-dimensional virtual environment. It allows the users to explore and to make any query about the information of the virtual world. It also demostrates how existing software or library packages like the Alias Wavefront [23] and the IRIS Performer [24] can be used for building a DVE system.

Another example system called the Diamond Park [1] is developed using the SPLINE of the MERL. It allows the users to explore and to interact with each others inside the virtual world. Inside the park, live conversations among the users are possible and it is designed with extensive audio effects. This is one of the unique features that the Diamond Park supports audio rendering. One of the interesting features of the Diamond Park is that, the users can ride a bicycle inside, or even to compete with each others, via a specific hardware.

Chapter 7

Conclusion

7.1 A Vision to the Future

I worked together with a colleague *in* the Europe this morning *on* the Mars. After the routine checking of all the equipments in the iron ore mine, we came back to the Earth for a lunch. I guess the robots there can be trusted, after these few months of close monitoring.

After the lunch, I still have tens minutes left and so I *logged* into my reading room to read my email and to browse through my favorite discussion groups.

Time's up! I logged out from my reading room and then I logged into the conference room for a meeting. I guess the worst thing for this kind of meeting is that, we could not *drink* anything inside the room, the Head-mounted Display (HMD) is not so convenient for such purpose. We can share only *non-consumable* objects inside. But the good thing is that, you can really fall into sleep without any obvious sign.

The conference finally finished, but we all have no idea that it's already late night. Working a whole day sitting in front of the computer is never a pleasant experience.

7.2 Conclusion

With the distributed virtual environment, people can communicate and interact with each others instantly regardless of the geographical distances between them. Potential application includes, but is not limited to, the virtual classroom, the Internet shopping, the virtual conference room, the multi-user games and the tele-presence.

In this thesis, we have formulated the load balancing problem on the server side in a DVE system as a partitioning problem by using the concept of AOI. The partitioning problem is proven to be NP-complete and heuristic algorithms are proposed to generate some sub-optimal partitioning schemes. By solving the partitioning problem, the work load of the DVE system can be shared among the DVE servers and the inter-communication incurred by the partitioning scheme is minimized. This allows the realization of a very large scale DVE in a cost effective manner.

On the client side of a DVE system, we have described the object state synchronization problem. Communication sub-graph construction algorithms have been presented to support the efficient delivery of the synchronization messages by multicasting, based on a number of design choices. We have also presented the derivation of the optimal synchronizing intervals with different level of guarantees based on a given set of consistency requirements.

Bibliography

[1] D.B. Anderson, J.W. Barrus, J.H. Howard, C. Rich, S. Chia, and R.C. Waters. Building multiuser interactive multimedia environments at merl. *IEEE Multimedia Volume 2 4*, pages pp 77–82, March 1997.

[2] A.J. Ballardie, P.F. Francis, and J. Crowcroft. Core based trees. *ACM SIGCOMM*, 1993.

[3] U. N. Bhat. Elements of applied stochastic processes. *John Wiley & Sons, New York*, 1972.

[4] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource reservation protocol (rsvp) – version 1 functional specification. *RFC 2205*, September 1997.

[5] T. Chiueh. Distributed systems support for networked games. *SPIE First International Symposium on Technologies and Systems for Voice, Video and Data Communications*, October 1995.

[6] G. Coulouris, J. Dollimore, and T. Kindberg. *Distributed Systems: Concepts and Design.* Addison-Wesley, 1994.

[7] K. Koskimies *et al.* Simnet - a software tool for system simulation. *The Second Symposium on Programming Languages and Software Tools*, August 1991.

[8] F. Fabbrocino, J.R. Santos, and R. Muntz. An implicitly scalable, fully interactive multimedia storage server. *International Workshop on Distributed Interactive Simulation and Real Time Applications (DIS-RT'98)*, 1998.

[9] M.R. Garey and D.S. Johnson. *Computer and Intractability: A Guide to the Theory of NP-Complessness.* W.H. Freeman and Company, 1979.

[10] R. Guerraoui and A. Schiper. Total order multicast to multiple. *ICDCS*, 1997.

[11] K. Guo and L. Rodrigues. Dynamic light weight groups. *ICDCS*, 1997.

[12] John C.S. Lui and M.F. Chan. Efficient partitioning algorithm for the distributed virtual environment system. *The Sixth International Conference on Distributed Multimedia Systems*, 1999.

[13] John C.S. Lui, M.F. Chan, T.F. Chan, W.S. Cheung, and W.W. Kwong. Virtual exploration and information retrieval system: Design and implementation. *Third International Workshop on Multimedia Information Systems*, 1997.

[14] John C.S. Lui, M.F. Chan, Oldfield K.Y. So, and T.S. Tam. Balancing workload and communication cost for a distributed virtual environment. *Fourth International Workshop on Multimedia Information Systems (MIS'98)*, September 24-26, 1998.

[15] John C.S. Lui, Oldfield K.Y. So, and T.S. Tam. Deriving communication subgraph and optimal synchronizing interval for distributed virtual environment system. *The IEEE International Conference on Multimedia Computing and Systems (ICMCS'99)*, June 1999.

[16] M.R. Macedonia, M.J. Zyda, D.R. Pratt, D.P. Brutzman, P.T. Barham, J. Falby, and J. Locke. Npsnet: A network software architecture for large scale virtual environments. *Presence Vol. 3, No. 4*, pages pp 265–280, Fall 1994.

[17] M.R. Macedonia, M.J. Zyda, D.R. Pratt, and P.T. Barham D.P. Brutzman. Exploiting reality with multicast groups: A network architecture for large-scale virtual environments. *IEEE Computer Graphics and Applications*, September 1995.

[18] F. Morgan. The hexagonal honeycomb conjecture. *Trans. Amer. Math. Soc.*, pages 1753–1763, 1999.

[19] K.L. Morse. Interest management in large-scale distributed simulations. *Technical Report, University of California, Irvine, Department of Information and Computer Science*, pages IC–TR–96–27, July 1996.

[20] R. Muntz, J.R. Santos, and S. Berson. A parallel disk storage system for realtime multimedia application. *Special Issue on Multimedia Computing Systems of the International Journal of Intelligent Systems*, 1998.

[21] J. Plesník. The complexity of designing a network with minimum diameter. *Networks 11*, pages 77–85, 1981.

[22] H. Schulzrinne, GMD Fokus, S. Casner, R. Frederick, and V. Jacobson. Rtp: A transport protocol for real-time applications. *RFC 1889*, January 1996.

[23] Alias Wavefront Software. *Learning Alias Level One.* A Division of Silicon Graphics Canada Limited, 1995.

[24] Performer Software. *IRIS Performer Programmer Guides.* Silicon Graphics Computer Softwares, 1995.

[25] S. Srinivasan. Multicasting in dis: A unified solution. *ELECSIM*, 1995.

[26] R.C. Waters and J.W. Barrus. The rise of shared virtual environments. *IEEE Spectrum*, pages pp 20–25, March, 1997.

[27] L. Zhang, S.E. Deering, D. Estrin, S. Shenker, and D. Zappala. Rsvp: A new resource reservation protocol. *IEEE Network*, pages 7(5):8–18, September 1993.