An Efficient Partitioning Algorithm for Distributed Virtual Environment Systems

John C.S. Lui, *Member*, *IEEE*, and M.F. Chan, *Student Member*, *IEEE*

Abstract—Distributed virtual environment (DVE) systems model and simulate the activities of thousands of entities interacting in a virtual world over a wide area network. Possible applications for DVE systems are multiplayer video games, military and industrial trainings, and collaborative engineering. In general, a DVE system is composed of many servers and each server is responsible to manage multiple clients who want to participate in the virtual world. Each server receives updates from different clients (such as the current position and orientation of each client) and then delivers this information to other clients in the virtual world. The server also needs to perform other tasks, such as object collision detection and synchronization control. A large scale DVE system needs to support many clients and this imposes a heavy requirement on networking resources and computational resources. Therefore, how to meet the growing requirement of bandwidth and computational resources is one of the major challenges in designing a scalable and cost-effective DVE system. In this paper, we propose an efficient partitioning algorithm that addresses the scalability issue of designing a large scale DVE system. The main idea is to dynamically divide the virtual world into different partitions and then efficiently assign these partitions to different servers. This way, each server will process approximately the same amount of workload. Another objective of the partitioning algorithm is to reduce the server-to-server communication overhead. The theoretical foundation of our dynamic partitioning algorithm is based on the linear optimization principle. We also illustrate how one can parallelize the proposed partitioning algorithm so that it can efficiently partition a very large scale DVE system. Lastly, experiments are carried out to illustrate the effectiveness of the proposed partitioning algorithm under various settings of the virtual world.

Index Terms—Distributed virtual environment, scalability issue, partitioning algorithm, load balancing, communication reduction, linear optimization.

1 Introduction

Advances in multimedia systems, parallel/distributed database systems, and high speed networking technologies enable system designers to build a distributed system which allows many users to explore and interact under a three dimensional virtual environment. In general, a 3D virtual environment is a virtual world which consists of many high-resolution 3D graphics sceneries to represent a real-life world. For example, we can have a 3D virtual world to represent a lecture hall so that hundreds of students and scientists can listen to a seminar presented by Professor Daniel C. Tsui[1] or we can have a large 3D virtual world to represent the latest COMDEX show which has many customers reviewing the latest softwares and electronic gadgets. This type of shared, computer-resident virtual world is called a *distributed virtual environment* (DVE) [34]. Like other ground-breaking computer technologies, DVE will change the way we learn, work, and interact with other people in the society.

To illustrate how a DVE system can change our lifestyles and the way we handle our business operation, let us consider the following situation. Let's say an architect from New York, a civil and a structural engineer from Paris, a financial planner from Hong Kong, and an interior designer from Tokyo all need to have a business meeting to discuss the designing and financing issues of a new high-rise office complex. Under a DVE setting, these people can convene a meeting in a virtual world without leaving their respective homes and offices. Their meeting can be carried out in a DVE system. These participants can interact with each other 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 around* in the proposed high-rise office building, interact with each other and carry out the discussion. For example, in this virtual high-rise office complex, each participant in the meeting is represented by a 3D object, which is known as an *avatar*. Each participant can walk around in this virtual office building and, in the process, rearrange any 3D object in the environment (e.g., rearrange paintings and furniture or select different kinds of carpet). Any change to a 3D object in this virtual world will be visible to all participants. Participants in this meeting are able to interact with each other in real time, as well as to inquire and to receive any relevant information of the virtual world. For example, participants can query about the credit history of a manufacturer who is responsible to produce the office furniture.

There are many challenging issues in designing a scalable, cost-effective, and high performance DVE system. We list some of the important research issues (although not exhaustive) when designing such DVE systems.

1. A 1998 Noble Prize winner in Physics for the discovery of a new form of quantum fluid with fractionally charged excitations.

- *J.C.S. Lui is with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Shatin, N.T., Hong Kong. E-mail: cslui@cse.cuhk.edu.hk.*
- *M.F. Chan is with ??? E-mail: ???.*

Manuscript received 11 Jan. 2000; revised 21 Apr. 2001; accepted 13 Aug. 2001.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number 111201.

- *Backend Database.* Designing a spatial and relational database engine so that users who are virtually exploring a large 3D virtual environment can query and receive relevant information about the environment being explored. The backend database engine should be able to support relational, spatial, and possible fuzzy types of queries. This research issue has been addressed in the VINCENT project [20], which is an earlier version of our current DVE system.
- *Object Consistency.* Since DVE clients can manipulate any object in a 3D virtual world, therefore, it is important to keep all objects in the virtual world in a consistent state such that, once the object is being accessed, other users may not be able to access it anymore. In general, there are several approaches to solve this concurrent access problem. For example, by exclusive locking of the object or by defining various access operations (similar to defining the read/write operations in database) that can be performed on the object so as to allow concurrent access to the object. Although concurrency control has been well studied in the database research community [7], concurrent data access under a DVE environment is more complicated. Objects in a 3D environment are usually rich in semantics and, therefore, one can propose different classes of concurrency control algorithms so as to support a high degree of concurrent access.
- *View Consistency.* Since users can move around in the virtual world and any user can access any object in the environment, users who are exploring the same virtual world have to be notified of the activities so as to keep their local views consistent. For example, if a user moves a chair from one location to another location, another user in the virtual world should be able to visualize this change. The process of sending the control information so as to maintain the view consistency among all users requires a tremendous amount of communication bandwidth. Recent research work on multicasting techniques [3], [10], [11] can help to reduce the network resource demand. In [22], [23], the authors propose approaches to connect all participating clients using different communication subgraph construction algorithms. The authors also derive an optimal synchronization interval so that every participating client in the same virtual world can view all objects with a high degree of consistency.
- *Balancing Workload and Reducing Communication Cost.* Potentially, one can use a DVE system to model a virtual world which represents a very large real-life environment and, at the same time, to support many users who want to explore in this virtual environment. This implies that servers for this DVE system have to have a large computational power so as to render different 3D models, perform the positional updates, as well as transfer the control information between different clients. Each participating client and server also needs to process many updates sent by others so as to keep the states of every object in the virtual world consistent.

  As the number of clients in a DVE system increases, so doesthe amount of network traffic generated by the clients and the DVE servers. The communication networks (WANs and LANs) supporting this DVE system can be easily overwhelmed by the traffic load. Furthermore, the increase of computation overhead at each client/server for processing the incoming information makes it difficult to implement a scalable DVE system. To realize a large scale DVE system, it is clear that the system needs to use multiple servers to handle clients' requests. (Therefore, an interesting and important problem in designing a DVE system is how one can partition and assign the workload among different servers in the DVE system and, at the same time, maintain a manageable level of communication overhead.) This is the main focus and contribution of this paper.

Let us briefly describe some previous work on DVE systems. In [20], the authors illustrated how to design and implement a virtual walk-through system such that a user can query and retrieve information about the virtual world. One limitation of this system is that it only allows a single user to explore the virtual world. However, it allows many users to explore the same virtual world but under different sessions. Therefore, for this special type of DVE system, there is no communication and interaction between different users in the system. In [27], [28], the authors described how to build a storage system that can support applications like the video-on-demand and 3D walk-through of a virtual world. The result is particularly interesting in the sense that the storage server can provide a guarantee for timely data retrieval for different multimedia applications, which may have vastly different quality of service requirements. In [24], the authors demonstrated how to build a distributed virtual environment for military applications and showed how it can be used to support hundreds of users. In [8], the authors designed a prototype DVE system which operates based on the Internet IP Protocol [9]. In this work, the authors illustrated that it is impossible for a single system to handle all the required workload and, therefore, workload partitioning approach was mentioned. However, there is no detail description on how to maintain synchronization among different servers and how to carry out the partitioning operation so as to reduce the communication overhead. In [31], the authors presented a software toolkit known as DIVE to construct a DVE system. Using the DIVE toolkit, users can define their own objects and behavior, but the underlying DIVE system assumes a single server and that each DIVE world is maintained by a dedicated server only. In [22], [23], the authors derived the optimal synchronization interval so that every client in the same virtual world can view all objects with a high degree of consistency.

Let us also briefly describe some previous work on graph partitioning. One can view that the DVE problem we mentioned above is similar to the problem of mapping a finite element graph on a distributed memory multicomputer

system. This type of graph mapping problem is shown to be NP-complete [19]. There is a lot of research work [1], [2], [4], [13], [16], [17], [18] on how to perform this type of finite graph mapping. All these proposed approaches try to find some suboptimal solutions. There are number of important heuristics which include techniques like recursive coordinate bisection, inertial bisection, scattered decomposition, and index-based partitioners [5], [29], [32], [36] for parallel scientific computation. There are also a number of methods which use explicit graph information to achieve partitioning. These heuristics include spectral bisection, recursive spectral multisection, min-cut-based methods and genetic algorithms [6], [14], [15], [16]. Note that all these proposed algorithms provide some suboptimal solutions and that they are not applicable to the DVE systems which we mentioned above. The reason is that users of a DVE system can join and leave anytime. Also, the communication cost can also vary in time due to temporary traffic congestion. Therefore, the structure of the graph which is used to present a DVE system, is more dynamic and the problem is more difficult.

The paper is organized as follows: In Section 2, we describe some possible DVE system architectures and the characteristics of avatar objects in the virtual world. In Section 3, we formulate the workload partitioning problem. We propose a partitioning algorithm to solve the scalability problem. The partitioning algorithm is based on the *linear optimization technique* and is shown to be computationally efficient and can effectively partition the workload evenly among the servers and, at the same time, reduce the communication overhead. We also illustrate how we can parallelize the proposed partitioning algorithm so that it can partition a very large scale DVE system. Section 4 contains the result and discussion of different experiments with various sizes of virtual world and different avatar's location distributions to illustrate the effectiveness of our proposed partitioning algorithm. Lastly, the conclusion is given in Section 5.

2 Distributed Virtual Environment

In this section, we describe various elements in a distributed virtual environment system, namely, 1) the system architectures, 2) the methodologies of representing clients as avatars and their area-of-interest (AOI), and 3) the dynamic joining and leaving properties of avatars in a DVE system.

2.1 DVE Architectures

Let us first consider different possible architectures so as to realize a multimedia service like a DVE application. In general, there are two possible architectures for implementing a DVE system. The choice of which architecture to use depends on the size of the virtual world (or the 3D virtual environment) that we want to model, as well as the number of concurrent participating clients under this virtual world. These two types of architectures are 1) single server distributed virtual environment architecture (SSDVE) and 2) multiple servers distributed virtual environment architecture (MSDVE).

Under a SSDVE architecture, all clients are connected to a single and dedicated server. To guarantee that all clients have the same consistent view of the virtual world, any action or activity generated by a client has to be transmitted in real time to all clients or, at the very least, to those clients who need to know about this new activity. This form of communication is accomplished as follows: The initiating client sends a message to the DVE server, the DVE server first transforms the message to some database operations (e.g., locking an object in the virtual world, changing the position of a given object), etc., then the server broadcasts (or multicasts) this new information to other clients in the DVE system so that every client can update their local view of this virtual world. It is important to point out that the SSDVE approach has the scalability problem. For example, if the number of clients is large, then the demand on the processing power, system buffer, and communication bandwidth will also be large. Therefore, an SSDVE architecture is only suitable for a small scale DVE system, for example, a virtual world with a small number of objects and a small number of participating clients.

To support many concurrent clients in a DVE system, one can adopt the MSDVE architecture. In an MSDVE architecture, multiple servers will be used and each server is responsible for handling a subset of the virtual world (e.g., some number of clients and some number of objects in the virtual environment), as well as the communication of its attached clients and the communication between servers. It is important to point out that, in order to keep the view consistency among the participating clients, some form of server-to-server communication is necessary. Therefore, a DVE system designer has to consider the issue of balancing the computational workload among different servers and reducing the communication cost between different servers. Balancing these two costs is not easy and, in general, it is architecturally dependent. This can be illustrated by the following examples. For a DVE system in which the servers are distributed around a wide area network, balancing the communication cost is more important. On the other hand, if a DVE system has many servers which are closely connected (or tightly coupled), then balancing the computational workload may be more important. Fig. 1 illustrates that we use three servers to divide up the virtual world. Clients are *associated* to a specific server whenever the client is in the administrative region of that server. Note that the administration region of a server can be *time-varying*. We will elaborate this point in a later section.

2.2 Representation of Client as Avatar and Its Area-of-Interest (AOI)

In a distributed virtual environment, we usually use an avatar, which is a 3D active object, to represent a participating client in a virtual world. In order to provide the interactive capability of a client, the avatar can move or traverse in a virtual world. A client can also use his/her avatar to communicate with other avatars (or other users in the virtual world) or use his/her avatar to access any 3D objects, such as books, chairs, glasses, etc., in the virtual environment. Since an avatar can move around and can interact with any static or dynamic 3D objects within the virtual world, for any action performed by an avatar, a

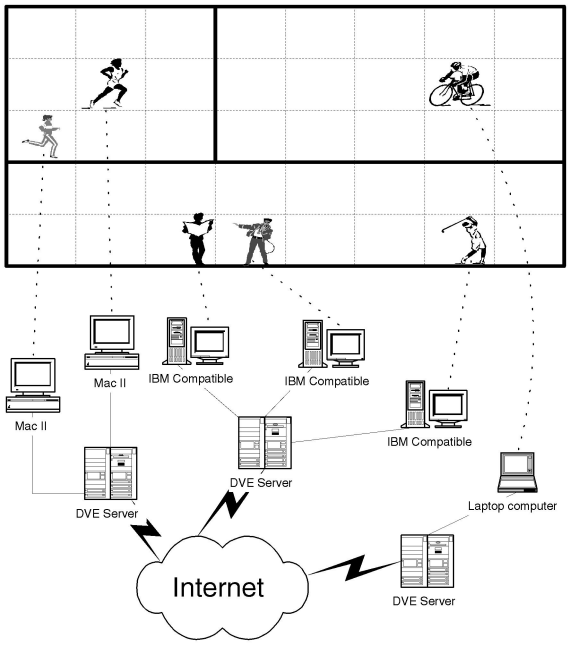


Fig. 1. MSDVE architecture for a DVE system.

DVE system needs to transfer the information for this new action to other avatars so as to keep the information of the virtual world consistent.

One simple way to maintain the consistency of the virtual world is to broadcast any action taken by any avatar to all other avatars in the system. However, this will incur a significant communication overhead. In general, each avatar only needs to know those activities that happened near his/her vicinity, for example, any activity that is within 10 meters of his/her position in the virtual world. Therefore, one way to significantly reduce the total communication overhead in a DVE system is to allow every avatar to define his/her own *area of interest* (AOI). In general, an AOI of an avatar is the region of the virtual world that, if there is any activity happened in this region, the avatar needs to know so as to update its own state and to make his/her view consistent. Fig. 2 illustrates the AOI concept of different avatars. In this paper, we use a circle to represent the AOI of each avatar. Let $AOI(A_i)$ denote the area-of-interest of avatar A_i. From the figure, we can see that $AOI(A_4) \subseteq AOI(A_1)$. Therefore, the DVE system has to inform A_1 of any activity generated by A_4. On the other hand, if there is any activity which happened within the intersection of $AOI(A_1)$ and $AOI(A_3)$, then the DVE system only needs to inform A_1 and A_3 of this activity. Also, since $AOI(A_5)$ or $AOI(A_6)$ does not intersect with the AOI of A_1 to A_4, the DVE system does not have to inform A_1,A_2, A_3, and $A_4$of any activity generated by A_5 or A_6.

2.3 Dynamic Membership

It is important to mention that the dynamic membership characteristic of an avatar in a DVE application. The dynamic membership characteristic of an avatar refers to the notion that a new client can join a virtual world at any given time or a participating client can leave a virtual world at any given time. Therefore, if a DVE application has the dynamic membership characteristic, this implies that the

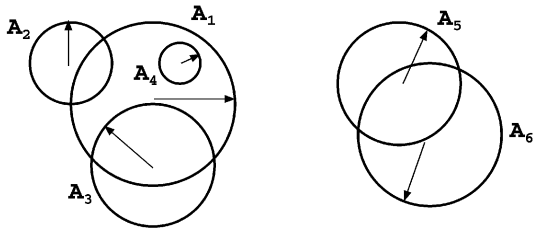


Fig. 2. Avatars and their area of interest.

number of participating clients who are exploring this virtual world is dynamically changing in time. For example, for the virtual world of business meeting we described in Section 1, the number of clients is *fixed* throughout the virtual world session (e.g., people are conducting a business meeting under a DVE system). On the other hand, in the virtual world of the COMDEX show we described in Section 1, the number of clients can vary in time since a user may want to log on to the DVE system and explore the COMDEX virtual world anytime or a user may decide to leave the COMDEX virtual world when he/she found the electronic gadget that he/she wanted to purchase.

Note that this dynamic membership characteristic of avatar in joining and leaving a virtual world increases the necessity of an efficient partitioning algorithm. Moreover, since an avatar can move from one location to another, it is possible that an avatar can move out of the region that is managed by a server, say S_i, and move into another region that is managed by another server S_j, where $i \neq j$. It is easy to observe that, if we do not adjust the avatar-to-server assignment, eventually, the workload among servers will vary significantly and the amount of traffic between servers may reach an unacceptable level. Therefore, it is important to find an efficient algorithm that can partition the workload in the virtual world *evenly* so that every server will carry the same amount of workload and, at the same time, the partitioning algorithm can also reduce the server-to-server communication overhead.

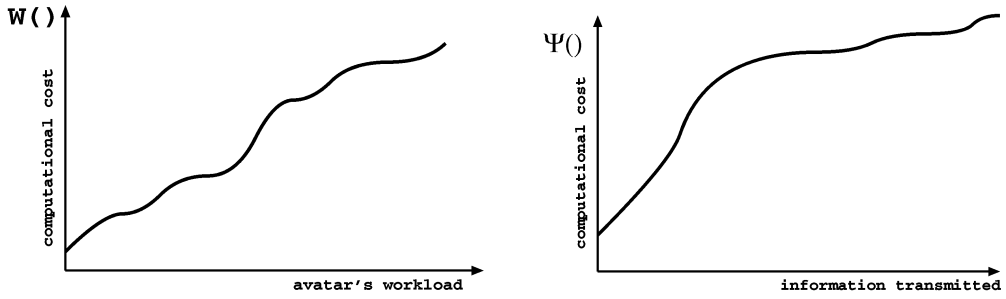
3 Partitioning Algorithm

In this section, we present the partitioning problem of a DVE system. We first formulate the partitioning problem and illustrate that it is an NP-complete problem in general. We then present an iterative partitioning algorithm. At the end of this section, we illustrate how we can parallelize the proposed partitioning algorithm so as to handle a very large scale DVE system. The effectiveness of the partitioning algorithm will be illustrated in Section 4.

3.1 Problem Formulation

Before we formulate the problem, let us define the following notation:

- P = Number of partitions or servers in a DVE system.
- S_i = The ith server in the DVE system where $i = 1, 2, \ldots, P$.
- n = Number of avatars in a DVE system.
- a_i = The ith avatar in a DVE system where $i = 1, 2, \ldots, n$.



(a) (b)

Fig. 3. (a) $w()$: Mapping function from the avatar's workload to the server's computational cost. (b) $\Psi()$: Mapping function from the avatar's information exchange to the server's computational cost.

- $w(\cdot)$ = A nondecreasing function that maps the workload of an avatar to the computational cost on a server. For example, $w(a_i)$ represents the computational cost on a server if the server is responsible for handling all workload events generated by the avatar a_i.
- $\mathcal{I}(a_i, a_j)$ = Amount of information exchange (in unit of bit) from avatar a_i to avatar a_j. If there is no communication between a_i and a_j, then $\mathcal{I}(a_i, a_j) = 0$.
- $\Phi_{S_i,S_j}(\cdot)$ = A nondecreasing function that maps the amount of information exchange (in unit of bits) to the communication cost server from S_i to S_j in the DVE system.
- $\Psi(a_i)$ = A nondecreasing function that maps the amount of information transmitted (in unit of bits) by an avatar a_i to the computational cost on a server.
- W_1 = A nonnegative real number representing the relative importance of the computational workload cost on a server.
- W_2 = A nonnegative real number representing the relative importance of the server-to-server communication cost on a server. Note that $W_1 + W_2 = 1.0$.
- $C_{\mathcal{P}}^W$ = Computation workload cost for a given partition policy $\mathcal{P}$.
- $C_{\mathcal{P}}^L$ = Communication cost for a given partition policy $\mathcal{P}$.
- $C_{\mathcal{P}}$ = Total cost for a given partition policy $\mathcal{P}$.
- $|\mathcal{P}|$ = The total number of possible partition policies.

We use a graph notation to represent a DVE system. Given a graph $G = (V, E)$, V represents the set of avatars in a DVE system, E represents the set of edges such that an edge $e_{ij} \in E$ represents that the avatar a_i needs to communicate with avatar a_j (e.g., the AOIs of a_i and a_j intersect with each other). Let $\mathcal{P}$ be a partition policy that divides V into P (number of servers) disjoint subsets $V_1, V_2, \ldots, V_P$ such that $V_i \cap V_j = \emptyset$ for $i \neq j$ and $\cup_{i=1}^{P} V_i = V$. In other words, all avatars in the subset V_i will be assigned to server S_i in the DVE system. Let the function $w()$ be an nondecreasing function that maps the workload of an avatar to the computational cost on a server. For example, Fig. 3a represents an example of a mapping function from the avatar's workload (e.g, workload of moving, rendering, etc.) to the server's computational cost. Let the function $\Psi()$ be a nondecreasing function that maps the amount of information transmitted (in unit of bits) by an avatar to the computational cost on a server. Fig. 3b illustrates an example of such a mapping function.

Given a partition $\mathcal{P}$, we define the computation workload cost $C_{\mathcal{P}}^W$ of this partition strategy as

$$C_{\mathcal{P}}^W = \sum_{j=1}^{P} \left(\left| \sum_{a_i \in V_j} w(a_i) + \Psi(a_i) - w^* \right| \right), \tag{1}$$

where $w^* = \sum_{i=1}^{n} [w(a_i) + \Psi(a_i)]/P$ is the computational workload per server under the *perfectly balanced* workload partition strategy. Therefore, $C_{\mathcal{P}}^W$ measures the *deviation* from the ideal load balancing partitioning strategy.

For the communication cost between servers under a partition strategy $\mathcal{P}$, we have to consider the AOI of each avatar. Specifically, if an avatar a_i is within the AOI of another avatar a_j, then, for any action taken by the avatar a_i, the DVE system needs to send this new information to the avatar a_j. Let $\mathcal{P}$ be given a partition strategy which divides V into $\{V_1, V_2, \ldots, V_P\}$ and that we assign partition V_i to server S_i. Let $\mathcal{I}(a_i, a_j)$ denote the amount of information exchange (in unit of bit) from avatar a_i to avatar a_j. And, let $\Phi_{S_i,S_j}()$ be a nondecreasing function that maps the amount of information exchange (in unit of bits) to the communication cost server from S_i to S_j in the DVE system. The communication cost between partition V_l and V_m (for $l \neq m$), denoted as C_{lm}, can be expressed as

$$\begin{aligned} C_{lm} = & \sum_{v_i \in V_l} \sum_{v_j \in V_m} \{\Phi_{S_l,S_m}(\mathcal{I}(v_i, v_j))\} \\ & + \sum_{v_j \in V_m} \sum_{v_i \in V_l} \{\Phi_{S_m,S_l}(\mathcal{I}(v_j, v_i))\}. \end{aligned} \tag{2}$$

The first term of the above equation expresses the communication cost for transmitting information updates from partition V_l to V_m, while the second term expresses the communication cost for transmitting information updates from partition V_m to V_l. Let $C_{\mathcal{P}}^L$ be the communication cost for a given partition strategy $\mathcal{P}$, we have:

$$C_{\mathcal{P}}^L = \sum_{l=1}^{P} \sum_{m>l}^{P} C_{lm}. \tag{3}$$

Therefore, $C_{\mathcal{P}}^L$ represents the total server-to-server communication cost for a given partition $\mathcal{P}$. In this paper, we assume the communication cost between avatars which are assigned to the same server as part of the server

computational workload cost. This assumption can be easily relaxed and be included in the total cost $C_{\mathcal{P}}$. The overall cost for the partition strategy $\mathcal{P}$, denoted by $C_{\mathcal{P}}$, can be expressed as

$$C_{\mathcal{P}} = W_1 C_{\mathcal{P}}^W + W_2 C_{\mathcal{P}}^L \qquad \text{with } W_1 + W_2 = 1, \tag{4}$$

where W_1 and W_2 represent the relative importance of the computational workload cost and the communication cost, respectively. For example, to implement a DVE system wherein servers are distributed across the Internet, we may want to assign more weight to W_2 so as to reduce the communication cost. On the other hand, if we used a clustered-based architecture wherein servers are connected within a LAN, then we may want to give more weighting to W_1. Lastly, the DVE partitioning problem is to find an optimal partition $\mathcal{P}^*$ such that

$$C_{\mathcal{P}}^* = \min_{\mathcal{P}}\{C_{\mathcal{P}}\}. \tag{5}$$

Before we discuss the proposed partitioning algorithm, we need to show the following important result [21].

Theorem 1. *The workload partitioning problem given in (5) is NP-complete.*

Proof. Let us consider the simplified version of the workload partition problem where $W_2 = 0$ (which corresponds to the case that the network has an infinite communication bandwidth and, therefore, the server-to-server communication cost is negligible). Given a set of nodes in V, we partition them into P disjoint subsets $V_1, \ldots, V_P$ such that $\cup_{i=1}^{P} V_i = V$ and the partitioning cost is

$$C_{\mathcal{P}} = \sum_{i=1}^{P} \left| \sum_{a \in V_i} w(a) + \Psi(a) - w^* \right|.$$

The main idea is to transform the partitioning problem to the *subset sum problem* [12], which is known to be NP-complete.

The subset sum problem can be described as follows: Given a positive integer B and a finite set $A = \{a_1, a_2, \ldots, a_N\}$ where $s(a_i) \in Z^+$ denotes the size of the element a_i. The subset sum problem is to determine whether there is a subset $A' \subseteq A$ such that the sum of the sizes of the elements in A' is exactly equal to B.

The transformation works as follows: For each avatar in the virtual world, we create an element $a_i \in A$. Let $s(a_i)$ equal the computational workload of the avatar a_i, that is, $s(a_i) = w(a_i) + \Psi(a_i)$. The communication cost between any avatar is set to zero. For the workload partition problem, we set the number of partitions as $P = k$. The value of B in the subset sum problem is set to $\frac{1}{k}\sum_{i=1}^{N} s(a_i)$. If an input instance of the subset sum problem should return a yes, then it implies that we can evenly divide the computational workload among the k servers. If the answer is no, this implies that the workload partitioning problem will have a load imbalance cost which is greater than zero. The transformation of a workload partitioning problem to a subset sum problem can be achieved in polynomial time. However, since the solution to the subset sum problem is NP-complete, we can conclude that the workload partitioning problem is also NP-complete. □

3.2 Exhaustive Partition (EP) Algorithm

One way to partition the avatars among different servers is by the exhaustive approach, that is, given n avatars in the DVE system and P servers, then each avatar can have at most P choices. Let $|\mathcal{P}|$ denote the total number of partition policies; thus, we have

$$|\mathcal{P}| = (P)(P)\cdots(P) = P^n. \tag{6}$$

Note that, although the exhaustive algorithm can find the optimal partition policy (e.g., a partition which has the minimum cost $C_{\mathcal{P}}^*$), this algorithm can only be applied to a small scale DVE system. For example, for $n = 16, P = 2$, the system needs to evaluate 65,536 different partition policies. For a moderate sized DVE system with $n = 16, P = 4$, the exhaustive algorithm requires approximately 4.3×10^9 evaluations to find the optimal partition.

Lemma 1. *The complexity of Exhaustive Partition Algorithm is* $O(P^{n+2}n^2)$.

Proof. Let the number of avatars and the number of servers be n and P, respectively. The complexity for calculating the cost between two servers is $O(n^2)$. For each partition configuration, the complexity for calculating the cost among all P servers is $O(P^2n^2)$. We need P^n evaluations to get an optimal solution because there are P^n partition policies. Thus, the overall complexity of the EP algorithm is $O(P^{n+2}n^2)$. □

3.3 Partitioning Algorithm

Due to the NP-completeness nature of the problem in (5), we propose the following efficient heuristic partitioning algorithm. The general idea of the proposed partitioning algorithm has the following steps

Partitioning Algorithm:

1. **begin**
2. Use the *recursive bisection partitioning(RBP)* algorithm to find the initial partition $\mathcal{P}_{RBP}$;
3. current_cost = $C_{\mathcal{P}_{RBP}}$;
4. difference = ∞; i=1;
5. **while** (difference $>$ d^*){
6. Use the *layering partitioning (LP)* algorithm to find a new partition $\mathcal{P}(i)$;
7. Given $\mathcal{P}(i)$, use the *communication refinement partitioning (CRP)* algorithm
8. to find a new partition $\mathcal{P}'(i)$;
9. difference = $|C_{\mathcal{P}'(i)} - \text{current\_cost}|$;
10. current_cost = $C_{\mathcal{P}'(i)}$;
11. i++;
12. }
13. final partition is $\mathcal{P}'(i)$;
14. **end**

The partitioning algorithm has three key components, namely, 1) the *recursive bisection partitioning algorithm* (RBP), 2) the *layering partitioning algorithm* (LP), and 3) the *communication refinement partitioning algorithm* (CRP). The RBP algorithm uses a divide-and-conquer approach to find the initial partition $\mathcal{P}_1$ that reduces the workload deviation and interserver communication cost. The LP algorithm and the CRP algorithm are derived from an elegant graph partitioning technique [30], which is based on the linear

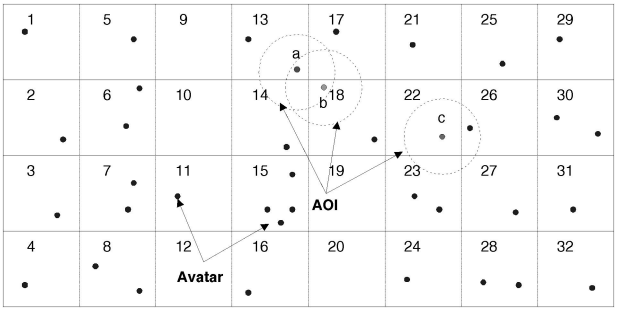


Fig. 4. A virtual world represented by 32 disjoint cells.

optimization principles [33], [35] to minimize the the workload deviation and the interserver communication, respectively. The algorithm will iterate the layering partitioning algorithm and the communication refinement partitioning algorithm until the difference of the total partitioning cost is less than some predefined threshold d^*. In general, we can vary the value of this predefined threshold so as to balance between the computational cost of running this algorithm versus the desired level of optimal partitioning. In Section 4, we show that the proposed partitioning algorithm can efficiently find a partition strategy that can reduce the total cost. In what follows, we describe each of these three components in detail.

3.3.1 Recursive Bisection Partition (RBP) Algorithm

The main idea about the recursive bisection partitioning algorithm is to divide the avatars in the virtual world into groups and then, based on a *divide-and-conquered* technique, to find an initial partitioning strategy $\mathcal{P}_1$. In the recursive bisection algorithm, we first assume that the AOI of avatar is in the form of a circle with an average diameter of $\mathcal{D}$. We then divide up the virtual world into N disjoint squared cells such that the area of a cell is equal to $\mathcal{D}^2$. The rationale of dividing the virtual world into cells is that, with high probability, most of the communication between avatars is between neighboring cells. Fig. 4 illustrates that the virtual world is divided into 32 disjoint cells.

Given the state of the DVE system, we can construct a graph $G_{RBP} = (V_{RBP}, E_{RBP})$ based on the following steps:

1. For each cell c_i, $1 \leq i \leq N$, create a node v_i in V_{RBP}.
2. Let $\mathcal{S}_{c_i}$ be the set of avatars that reside in cell c_i. Compute the computational workload for cell c_i, which is equal to $\sum_{a_j \in \mathcal{S}_{c_i}} w(a_j)$.
3. For any two adjacent cells, c_i and c_j, create an edge E_{ij} between the node v_i and v_j such that the cost of E_{ij}, which is denoted by $C(E_{ij})$, is nonzero. The cost of the edge E_{ij} is computed based on the following formula:

$$C(E_{ij}) = \sum_{v_i \in \mathcal{S}_{c_i}} \sum_{v_j \in \mathcal{S}_{c_j}} \Phi_{\mathcal{S}_{c_i}, \mathcal{S}_{c_j}}(\mathcal{I}(v_i, v_j)) + \sum_{v_j \in \mathcal{S}_{c_j}} \sum_{v_i \in \mathcal{S}_{c_i}} \Phi_{\mathcal{S}_{c_j}, \mathcal{S}_{c_i}}(\mathcal{I}(v_j, v_i)).$$

The recursive bisection partitioning algorithm is based on the concept of the divide-and-conquered technique. Without the loss of generality, let us first present the RBP algorithm for N cells system and the number of servers (P) is equal to two. Let V_k^n be the partition for the kth server (where $k = 1, 2, \ldots, P$) with $n \leq N$ cells. Initially, we set

$$V_1^N = V_{RBP} = \{v_1, v_2, \ldots, v_N\} \quad ; \quad V_2^0 = \emptyset. \tag{7}$$

Let $\mathcal{P}_{RBP}(i)$ be the ith partition configuration and let $C_{\mathcal{P}_{RBP}(i)}$, the cost based on (4), be the cost of partition configuration $\mathcal{P}_{RBP}(i)$. Based on the initial partition, we have $\mathcal{P}_{RBP}(0) = (V_1^N, V_2^0)$ and the corresponding $C_{\mathcal{P}_{RBP}(0)}$. We can then find the next partition policy $\mathcal{P}_{RBP}(1)$ by moving one cell from V_1^N to V_2^0 and compute the cost $C_{\mathcal{P}_{RBP}(1)}$. In finding the new partition $\mathcal{P}_{RBP}(1)$, we choose to move a cell from the first server to the second server such that the total cost $C_{\mathcal{P}_{RBP}(1)}$ is minimized. This can be achieved by considering each cell in V_1^N and this process takes linear time with respect to the total number of cells (N) in the system. Formally, we have

$$\mathcal{P}_{RBP}(i) = (V_1^{N-i}, V_2^i) \qquad i = 0, 1, \ldots, N, \tag{8}$$

where $\mathcal{P}_{RBP}(i+1)$ can be derived by the following recursive construction method:

$$\begin{aligned} \mathcal{P}_{RBP}(i+1) =& (V_1^{(N-(i+1))}, V_2^{i+1}) \\ =& (V_1^{(N-i)} - \{v_j\}, V_2^i \cup \{v_j\}) \\ & \text{for } v_j \in V_1^{(N-i)} \\ & \text{and } C_{\mathcal{P}_{RBP}(i+1)} \text{ is minimized.} \end{aligned} \tag{9}$$

That is, we consider every node in $V_1^{(N-i)}$ such that in choosing node v_j, we minimized the cost of partition policy $\mathcal{P}_{RBP}(i+1)$. Note that $C_{\mathcal{P}_{RBP}(0)}$ and $C_{\mathcal{P}_{RBP}(N)}$ represent the two extremes of the highest load imbalanced cost (i.e., all cells are assigned to one server and there is no server-to-server communication). Therefore, the RBP algorithm is to choose a configuration that

$$\mathcal{P}_{RBP}(i^*) = \{\mathcal{P}_{RBP}(i) \mid C_{\mathcal{P}_{RBP}(i)} = \min_{0 \leq j \leq N} \{C_{\mathcal{P}_{RBP}(j)}\}\}. \tag{10}$$

The above bisection algorithm applies for $P = 2$. For a larger number of P, we can first use the bisection partitioning algorithm presented above, then choose a partition that has the largest cost, and then apply the bisection partitioning algorithm again. At the end of the RBP algorithm, we obtain a partition strategy $\mathcal{P}_{RBP}$ that partitions the graph G_{RBP} into P disjoint regions (or $V_{RBP} = \{V_1 \cup \cdots \cup V_P\}$) such that all the nodes in V_i will be assigned to the ith server.

Lemma 2. *The complexity of Recursive Bisection Partition Algorithm is $O(N^3(P-1))$.*

Proof. Let the number of cells and the number of servers be N and P, respectively. The number of evaluation of the partitioning configurations is $N(P-1)$. For each evaluation, we need to calculate the cost between two servers, the complexity is $O(N^2)$. Therefore, the overall complexity of the RBP Algorithm is $O(N^3(P-1))$. Note that the complexity of the RBP algorithm is much smaller than the exhaustive approach because the number of cells (N) is much smaller than number of avatars (n) in the DVE system. □

3.3.2 Layering Partitioning (LP) Algorithm

Although the RBP algorithm can produce a partition strategy $\mathcal{P}_{RBP}$, there are several shortcomings in the approach. For example, the computational complexity is high (which we will illustrate in Section 4) and, at the same time, the overall cost $C_{\mathcal{P}_{RBP}}$ for $\mathcal{P}_{RBP}$ can still be reduced further. The main idea about the layering partitioning algorithm is to label each avatar using a server number. The label (or server number) serves as a *possibility* of moving the corresponding avatar to a new partition which has that server number. The decision of whether to move the avatar can be formulated as a *linear programming optimization* problem which we illustrate in this section.

Since we have obtained a partition $\mathcal{P}_{RBP}$ from the recursive bisection partitioning algorithm, we can *relax* the assumption that the DVE world is divided up into cells. We first have to construct a graph $G_{LP} = (V_{LP}, E_{LP})$ such that each node in the graph represents an avatar. An edge $e_{ij} \in E$ represents that avatar a_i is within the AOI of avatar a_j and the cost of this edge e_{ij} is $\mathcal{I}(a_i, a_j)$. The construction of the graph $G_{LP} = (V_{LP}, E_{LP})$ is specified by the following algorithm

Graph Construction Algorithm:

1. **begin**
2. **for** each avatar a_i, create a node v_i in G_{LP};
3. **for** each $v_i \in G_{LP}$, **do** { /* initiate */
4. initialize variables connected$[v_i] =$ false;
5. initialize variables server_number$[v_i] = k$ where $v_i \in V_k$ and $1 \leq k \leq P$;
6. /* note that the server index k for V_k can be obtained from the output of the RBP algorithm */
7. }
8. **for** $v_i \in V_{LP}$ **do** {
9. /* create edges and mark those nodes along the partition boarder as connected */
10. **for** $v_j \in V_{LP}$ where $i \neq j$, **do** {
11. **if** v_j is within the AOI of v_i **then** {
12. create an edge e_{ji} in E_{LP}; /* e_{ji} is an edge between v_j and v_i where $v_i, v_j \in V_{LP}$ */
13. set the weight of $e_{ji} = \mathcal{I}(v_j, v_i)$;
14. **if** (server_num$[v_i] \neq$ server_num$[v_j]$) **then** {
15. connected$[v_i] =$ connected$[v_j] =$ true; }
16. }
17. }
18. }
19. **for** all $v_i \in G_{LP}$ **do** { /* connect the remaining nodes */
20. **if** (connected$[v_i] =$ false) **then** {
21. **if** ((there exists a node v_j which is a neighbor of v_i) /* v_j is a neighbor of v_i if e_{ij} exists */
22. **and** (connected$[v_j] =$ true)) **then**
23. connected$[v_i] =$ true;
24. **if** (connected$[v_i] =$ false) **do** {
25. find a nearest node v_k such that connected$[v_k] =$ true **and** server_num$[v_i] =$ server_num$[v_k]$;
26. create an edge $e_{ik} \in E_{LP}$;
27. set weight of $e_{ik} = \mathcal{I}(v_i, v_k)$ or $\epsilon > 0$;
28. connected$[v_i] =$ true;
29. }
30. **if** (connected$[v_i] =$ false) **do** {
31. find a nearest node $v_k \in G_{LP}$ such that connected$[v_k] =$ true;
32. create an edge $e_{ik} \in E_{LP}$;
33. set weight of $e_{ik} = \mathcal{I}(v_i, v_k)$ or $\epsilon > 0$;
34. connected$[v_i] =$ true;
35. }
36. }
37. }
38. **end**

The purpose of the constructing graph G_{LP} is to produce a *connected* graph so that we can perform the layering step. The motivation of the layering procedure is to identify which node can be assigned to a different server so as to reduce the overall cost. The layering procedure can be described as follows:

First, a node v_i is considered as a *boarder node* when there exists a node v_j such that

1. there exists an edge $e_{ij} \in E_{LP}$ and
2. server_num$[v_i] \neq$ server_num$[v_j]$. In other words, node v_i is along a partition boarder.

Let $\mathcal{S}_{bn}$ be the set of all boarder nodes in the graph G_{LP}. For each node $v_i \in \mathcal{S}_{bn}$, we find a partition V_{j^*} such that:

$$\sum_{v_k \in V_{j^*}} \mathcal{I}(e_{ik}) = \max_{1 \leq l \leq P} \left\{ \sum_{v_k \in V_l} \mathcal{I}(e_{ik}) \right\}.$$

Once we find that partition V_{j^*}, we set the layer number of the node v_i, denoted by layer_num$[v_i]$, to j^*. At this point, we let $\mathcal{S}_l$ to denote the set of nodes that has an assigned layer number. Note that $\mathcal{S}_l \subset V_{LP}$. The remaining step is to consider all those nodes in V_{LP} which have no layer number yet. To accomplish this, let us consider a node v_i which has no layer number. For this node v_i, we find a label j^* such that the sum of the weight from node v_i to nodes with label j^* is

$$\max \left(\sum_{v_k \in \mathcal{S}_l} \mathcal{I}(e_{ik}) \right) \qquad \text{where layer\_num}[v_k] = j^*.$$

Then, we set layer_num$[v_i]$, the layer number of node v_i, as j^*. Now, for all those nodes that have the newly assigned layer number, we add them to the set $\mathcal{S}_l$. We repeat the layer number assignment process for all nodes in V_{LP}.

To illustrate the layer number assignment concept, Fig. 5 depicts a graph G_{LP} with the corresponding edge weight. All boarder nodes in the graph G_{LP} are highlighted. Note that graph G_{LP} is divided into three partitions, namely, V_i, V_j, and V_k. Fig. 6a shows the assignment of layer number for the boarder nodes and Fig. 6b illustrates the assignment of layer number for the remaining nodes.

After we finished layering all nodes in the graph G_{LP}, we can consider moving some of the nodes with layer number i

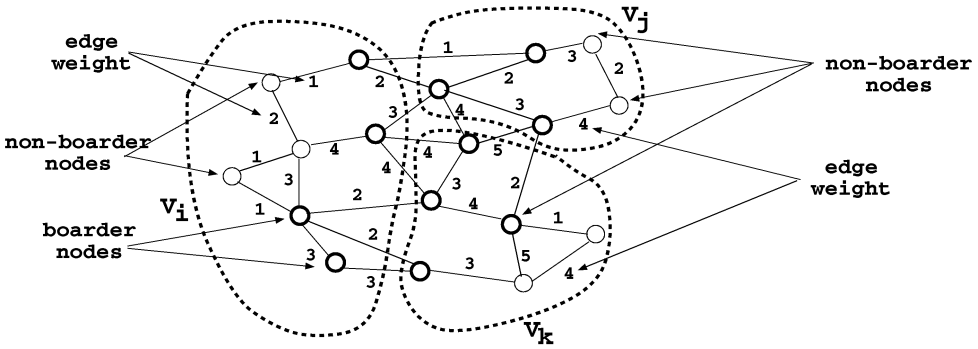


Fig. 5. A graph G_{LP} with boarder nodes (e.g., nodes in bold circle), edge weight, and three partitions V_i, V_j and V_k.

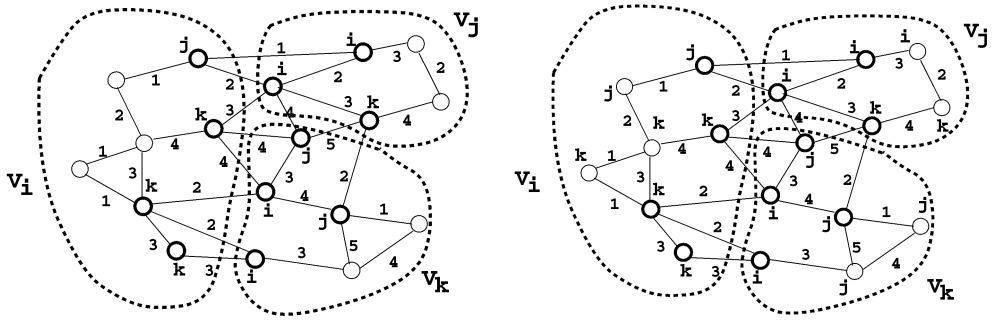


Fig. 6. Assigning labels to all nodes in G_{LP}. (a) Assign labels to boarder nodes. (b) Assign labels to other nodes.

to server i, where $1 \leq i \leq P$, so as to reduce the computation workload deviation (e.g., reduce the workload cost according to (1)). The number of nodes that can be moved can be formulated as a linear optimization problem. Let α_{ij} represent the number of nodes in partition V_i that can be moved to partition V_j (e.g., these are the nodes that are in partition V_i and with a layer number equal to j). For example, in Fig. 6b, $\alpha_{ij} = 2$. Let $|V_i|$ represent the total number of nodes in partition V_i (e.g., in Fig. 6b), $|V_i| = 7$). Let x_{ij} be the decision variable of the number of nodes that we eventually move from partition V_i to V_j so as to reduce the computation workload cost of the DVE system. We would like to minimize the total number of movement, or minimize $\sum_{1 \leq i \neq j \leq P} x_{ij}$ so as to achieve workload balancing property.

The formulation of the linear optimization is

$$\text{Minimize} \sum_{1 \leq i \neq j \leq P} x_{ij}, \tag{11}$$

which is subject to the following constraints:

$$0 \leq x_{ij} \leq \alpha_{ij} \leq |V_i| \tag{12}$$

$$\sum_{1 \leq j \neq i \leq P} (x_{ij} - x_{ji}) = |V_i| - \frac{1}{P} \sum_{j=1}^{N} \big(w(a_j) + \Psi(a_j)\big) \quad \text{for } 1 \leq i \leq P. \tag{13}$$

The constraint in (12) ensures that the number of nodes that we move from partition V_i to partition V_j is less than or equal to the feasible number of candidate nodes. The constraint in (13) ensures that the difference of the total number of nodes that we move into server i and the total number of nodes that move out of server i is equal to the workload deviation of server i under the ideal load balanced situation. In other words, we try to make sure that the workload in server i is as close to the ideal workload balanced situation as possible.

Lemma 3. *The complexity of the Layering Partitioning Algorithm is $O(P^6)$.*

Proof. Let the number of partitions be P. We will have $P(P-1)$ variables and $P(P-1) + P = P^2$ constraints. Although the number of iterations required for a linear programming is problem dependent, a good estimate is about $2(P^2 + P(P-1))$. Thus, the overall complexity of the LP algorithm is $O(P^6)$. □

To illustrate this linear optimization solution technique, let us consider the graph in Fig. 6b. The formulation is given as follows:

$$\begin{aligned}
&\textbf{Minimize} && x_{ij} + x_{ik} + x_{ji} + x_{jk} + x_{ki} + x_{kj} \\
&\textbf{subject to :} && x_{ij} \leq 2; x_{ik} \leq 5; x_{ji} \leq 3; x_{jk} \leq 2; x_{ki} \leq 2; \\
&&& x_{kj} \leq 4 \\
&&& x_{ij} + x_{ik} - x_{ji} - x_{ki} = 7 - 6 = 1 \\
&&& x_{ji} + x_{jk} - x_{ij} - x_{kj} = 5 - 6 = -1 \\
&&& x_{ki} + x_{kj} - x_{ik} - x_{jk} = 6 - 6 = 0.
\end{aligned}$$

The solution to the above optimization problem is $x_{ij} = 1$, $x_{ik} = x_{ji} = x_{jk} = x_{ki} = x_{kj} = 0$. Therefore, we choose the node that has a layer number equal to j in partition V_i and move it to partition V_j. In selecting which node to move, we start from the boarder nodes, then to the nodes in

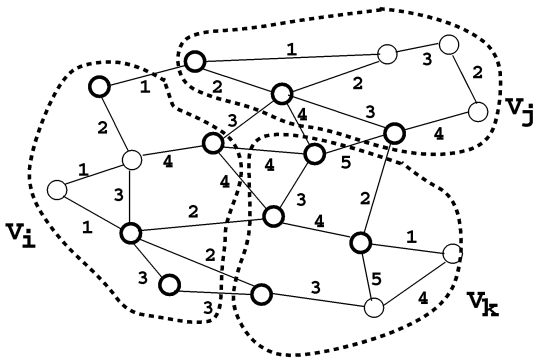


Fig. 7. New partition after the LP algorithm.

the inner layers. Note that during the node movement, only those nodes which will not increase the communication cost will be moved. After we moved a node v_l from partition V_i to V_j, we reassign the server number of node v_l as $\text{server\_num}(v_l) = j$. The process stops when the number of nodes moved is equal to the solution. The new partitioning policy $\mathcal{P}_{LP}$ for the graph G_{LP} in Fig. 6b is given in Fig. 7.

Let the workload weighting, W_1, and the communication cost weighting, W_2, equal 0.5. Assume the workload for maintaining an avatar is 10. Then, before we apply the LP algorithm, the cost of the partition generated by the RBP algorithm is:

$$\begin{aligned} C_{\mathcal{P}}^{W} &= \sum_{j=1}^{P} \left(\left| \sum_{a_i \in V_j} w(a_i) + \Psi(a_i) - w^* \right| \right) \\ &= 10 \times (|7-6| + |5-6| + |6-6|) = 20 \\ C_{\mathcal{P}}^{L} &= C_{ij} + C_{ji} + C_{ik} + C_{ki} + C_{jk} + C_{kj} \\ &= (2+3) + (1+3) + (4+2+3) + (4+4+3) \\ &\quad + (4+5) + (5+2) \\ &= 45 \\ C_{\mathcal{P}} &= W_1 C_{\mathcal{P}}^{W} + W_2 C_{\mathcal{P}}^{L} \\ &= 0.5 \times 20 + 0.5 \times 45 = 32.5. \end{aligned}$$

After we applied the LP algorithm, the cost of the partition generated by the LP algorithm is

$$\begin{aligned} C_{\mathcal{P}}^{W} &= \sum_{j=1}^{P} \left(\left| \sum_{a_i \in V_j} w(a_i) + \Psi(w_i) - w^* \right| \right) \\ &= 10 \times (|6-6| + |6-6| + |6-6|) = 0 \\ C_{\mathcal{P}}^{L} &= C_{ij} + C_{ji} + C_{ik} + C_{ki} + C_{jk} + C_{kj} \\ &= (1+3) + (1+3) + (4+2+3) + (4+4+3) \\ &\quad + (4+5) + (5+2) \\ &= 44 \\ C_{\mathcal{P}} &= W_1 C_{\mathcal{P}}^{W} + W_2 C_{\mathcal{P}}^{L} \\ &= 0.5 \times 0 + 0.5 \times 44 = 22 \end{aligned}$$

We can see that the cost is reduced by more than 30 percent. In Section 4, we will illustrate that, by using the layer partitioning algorithm, there is a significant reduction in the overall partition cost.

3.3.3 Communication Refinement Partitioning (CRP) Algorithm

The layering partitioning (LP) algorithm is used to reduce the overall computation workload cost of the system. In the communication refinement partitioning (CRP) algorithm, the objective is to reassign some nodes (or avatars) to another partition so as to reduce the server-to-server communication cost.

Given the partitioned graph $G_{LP} = \{V_{LP}, E_{LP}\}$ from the LP algorithm, let us consider all the boarder nodes. Again, a node v_i is considered as a boarder node when there exists a node v_j such that 1) there is an edge from $e_{ij} \in E_{LP}$ and 2) $\text{server\_num}(v_i) \neq \text{server\_num}(v_j)$. Let $\mathcal{S}_{bn}$ be the set of all boarder nodes. For each $v_i \in \mathcal{S}_{bn}$, we compute Γ_j, the communication cost to partition V_j, as

$$\Gamma_j = \sum_{v_k \in V_j} \mathcal{I}(e_{ik}) \qquad \text{for } 1 \leq j \leq P. \tag{14}$$

Let $\Gamma_{j^*} = \max_{1 \leq j \leq P}\{\Gamma_j\}$. Then, Γ_{j^*} is the maximum communication due to node v_i and this communication is to partition V_{j^*}. Then, the node $v_i \in \mathcal{S}_{bn}$ will be assigned a label j^*, that is, $\text{communication\_number}(v_i) = j^*$. The motivation for finding the $\text{communication\_number}$ is that, if we move node v_i to partition V_{j^*}, then it is possible for us to reduce the communication cost. We repeat this process for all nodes in $\mathcal{S}_{bn}$. At the end of this communication number assignment, we have determined the possible partition assignment of all those boarder nodes so as to reduce the system communication cost.

In order to determine the final partition assignment, we can formulate the problem as a linear optimization problem. Let β_{ij} denote the number of nodes in partition V_i that has the $\text{communication\_number}$ assignment equal to j. We define y_{ij} as the decision variable of the number of nodes that we eventually move from partition V_i to V_j so as to reduce the communication cost of the system. We can formulate the problem as

$$\text{Maximize} \sum_{1 \leq i \neq j \leq P} y_{ij}, \tag{15}$$

subject to

$$0 \leq \ y_{ij} \leq \beta_{ij} \qquad \text{for } 1 \leq i \neq j < P \tag{16}$$

$$\sum_{1 \leq i < j} (y_{ij} - y_{ji}) = 0 \qquad (0 \leq j < P). \tag{17}$$

The constraint in (16) ensures that the number of nodes that we move from partition V_i to partition V_j is less than or equal to the feasible number of candidate nodes. The constraint in (17) ensures that the number of nodes that move into server i and the number of nodes that move out of server i is the same so as to maintain the workload balancing property.

Lemma 4. *The complexity of Communication Refinement Partitioning Algorithm is $O(P^6)$.*

Proof. This algorithm also uses the linear programming technique. Let the number of partitions be P. We will have $P(P-1)$ variables and $P(P-1) + P = P^2$ constraints.

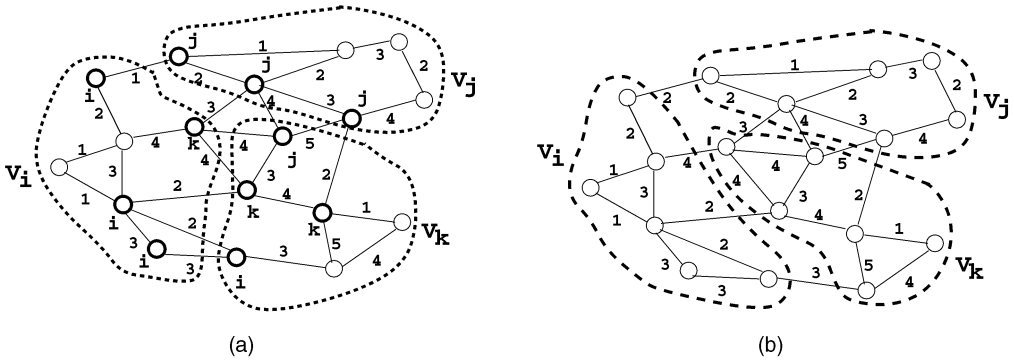


Fig. 8. Example of the CRP refinement algorithm. (a) Partition before the CRP refinement. (b) Partition after the CRP refinement.

The number of iterations required for linear programming is problem dependent and a good estimate is $2(P^2 + P(P-1))$. The time required for one iteration of the linear programming is $O(P(P-1) * P^2)$. Thus, the overall complexity for the linear programming is $O(P^6)$. □

Let us illustrate the communication refinement algorithm given that we have the partitioned graph G_{LP} in Fig. 7. The optimization formulation is

$$\begin{aligned}
&\textbf{Maximize} && x_{ij} + x_{ik} + x_{ji} + x_{jk} + x_{ki} + x_{kj} \\
&\textbf{subject to :} && x_{ij} \leq 0; x_{ik} \leq 1; x_{ji} \leq 0; x_{jk} \leq 0; x_{ki} \leq 1; \\
& && x_{kj} \leq 1 \\
& && x_{ij} + x_{ik} - x_{ji} - x_{ki} = 0 \\
& && x_{ji} + x_{jk} - x_{ij} - x_{kj} = 0 \\
& && x_{ki} + x_{kj} - x_{ik} - x_{jk} = 0.
\end{aligned}$$

The solution to the above optimization problem is

$$x_{ik} = 1, x_{ki} = 1 \text{ and } x_{ij} = x_{ji} = x_{jk} = x_{kj} = 0.$$

Therefore, we can exchange one node between partitions V_i and V_k to reduce communication cost. The new partitioning is given in Fig. 8.

After we apply the CRP algorithm, the cost of the partition is

$$\begin{aligned}
C_{\mathcal{P}}^W &= \sum_{j=1}^{P} \left(\left| \sum_{a_i \in V_j} w(a_i) + \Psi(w_i) - w^* \right| \right) \\
&= 10 \times (|6-6| + |6-6| + |6-6|) = 0 \\
C_{\mathcal{P}}^L &= C_{ij} + C_{ji} + C_{ik} + C_{ki} + C_{jk} + C_{kj} \\
&= 1 + 1 + (4+2+3) + (4+2+3) + (4+5) \\
&\quad + (3+5+2) \\
&= 39 \\
C_{\mathcal{P}} &= W_1 C_{\mathcal{P}}^W + W_2 C_{\mathcal{P}}^L \\
&= 0.5 \times 0 + 0.5 \times 39 = 19.5.
\end{aligned}$$

3.4 Parallel Partitioning Algorithm

In order to support a large virtual world that has many graphical entities and, at the same time, support many concurrent clients, we need to consider a large scale DVE system. For this type of large scale DVE system, it usually requires many servers so as to satisfy the enormous computation demand. As the number of clients and servers increases, so does the computational and communication requirements to realize a high performance DVE system. In order to satisfy the performance requirement, we need to have a fast and efficient partition algorithm. In this section, we introduce a parallel approach to perform the workload and communication partition. For the ease of presentation, assume that the number of servers we have is $P = 2^n$, where $n \in \{1, 2, \ldots\}$. In general, the parallel approach has the following steps

Parallel Partition Approach:

1. **begin**
2. Divide the virtual world into n equal size regions, we called them $R_1, \ldots, R_n$;
3. Let S_i (where $i = 1, 2, \ldots, P$) be the i^{th} server in the DVE system;
4. Assign the virtual world in region R_i to the set of servers in $\{S_{(i-1)n+1}, S_{(i-1)n+2}, \ldots, S_{(i)n}\}$.
5. **for** each virtual world in region R_i, **do parallel** {
6. Use the *recursive bisection partitioning (RBP)* algorithm to find the initial partition $\mathcal{P}_{RBP}$;
7. current_cost = $C_{\mathcal{P}_{RBP}}$;
8. difference = ∞; i=1;
9. **while** (difference $> d^*$) {
10. Use the *layering partitioning algorithm (LP)* to find a new partition $\mathcal{P}(i)$;
11. Given $\mathcal{P}(i)$, use the *communication refinement partitioning (CRP) algorithm*
12. to find a new partition $\mathcal{P}'(i)$;
13. difference = $|C_{\mathcal{P}'(i)} - \text{current\_cost}|$;
14. current_cost = $C_{\mathcal{P}'(i)}$;
15. i++;
16. }
17. }
18. Combine these smaller virtual worlds into one large virtual world;
19. Use the LP algorithm to find a new partition $\mathcal{P}(i)$;
20. Use the CRP algorithm to find a new partition $\mathcal{P}'(i)$ based on $\mathcal{P}(i)$;
21. final partition is $\mathcal{P}'(i)$;
22. **end**

Consider the partitioning of a virtual world with 16 servers. This can be completed in two stages: In the first stage, we can divide the virtual world into four smaller virtual worlds and assign these small virtual worlds into different servers. In the second stage, we apply the partitioning algorithm we discussed in Section 3.3 to partition each of the small virtual worlds. The partitioning processes can be carried out in parallel for different servers. From the experiments in Section 4, we see that the parallel approach can have a comparable partitioning cost, as compared to the sequential partitioning approach. The advantage of the parallel approach is that the time required for applying RBP, LP, and CRP algorithms to each small virtual world will be much less than the time required for applying these algorithms to the original large virtual world. We also add one additional stage to the parallel partitioning algorithm. In this additional stage, we combine all the small virtual worlds and apply the LP and CRP algorithms to the combined virtual world. As we will illustrate in the next section, we can obtain a partition which has a smaller cost than the sequential partitioning algorithm.

4 Experiments

In this section, we present experiments for the algorithm that we discussed in the previous section and apply the algorithm to both a small and a large scale virtual world. For the small virtual world experiment, since the problem state space is manageable, we can compare the performance of our proposed algorithm with the exhaustive partitioning algorithm, which guarantees to produce the optimal partition policy $\mathcal{P}^*$. We also carry out experiments to investigate the dynamic characteristics of avatars (e.g, avatars that can move around, as well as joining and leaving a virtual world session) and the effectiveness of proposed parallel partitioning algorithm. In general, we use three different methods to generate the position of each avatar in the virtual world. These methods 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 is the horizontal dimension of the virtual world and V_y is the vertical dimension of the virtual world.
- **Skewed Distribution.** Given the size of the DVE world as (V_x, V_y), we divide the number of avatars in the DVE system 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 $\left(0, \frac{iV_x}{4}\right)$ and y is uniformly distributed between $\left(0, \frac{iV_y}{4}\right)$. Under this scheme, most of the avatars will be positioned within the square area defined by the two coordinates $[0, 0]$ and $\left[\frac{V_x}{4}, \frac{V_y}{4}\right]$.
- **Clustered Distribution.** Given the size of the DVE world as (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 are 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

$$x = \begin{cases} 0 & \text{if } x_i + dx \times \Omega < 0 \\ V_x & \text{if } x_i + dx \times \Omega > V_x \\ x_i + dx \times \Omega & \text{otherwise.} \end{cases} \quad (18)$$

and

$$y = \begin{cases} 0 & \text{if } y_i + dy \times \Omega < 0 \\ V_y & \text{if } y_i + dy \times \Omega > V_y \\ y_i + dy \times \Omega & \text{otherwise.} \end{cases} \quad (19)$$

In our experiments, 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 virtual world whose size is of 4×4 units and $\Omega = 3.0$ for the virtual world whose size is of 25×25 units.

4.1 Experiment 1: Small Virtual World

In this experiment, we use a small virtual world with a dimension of 4×4 units. The total number of avatars in this virtual world is equal to 13 and the number of servers P is equal to three. We set both the workload weighting, W_1, and the communication cost weighting, W_2, to 0.5. The diameter of the AOI of each avatar is equal to 1.

Table 1, Table 2, and Table 3 illustrate the experimental results under the uniform, skewed, and clustered location distributions, respectively. To illustrate, consider the uniform distribution scenario in Table 1a. It indicates that using the exhaustive partitioning algorithm, the optimal (or minimum) system cost is 3.47 and it takes 1202.12 seconds to obtain the optimal partitioning configuration. Using our proposed partitioning algorithm, we first use the RBP algorithm and it generates a partition with a system cost of 3.47 and it takes less than 0.01 seconds to generate the partitioning configuration. After we obtain this initial partitioning configuration, we also apply the LP and CRP algorithms. Note that, for this experiment, we do *not* observe any cost reduction after we apply the LP and CRP algorithms.[2] For the uniform distribution, the computation time for these two algorithms is again less 0.01 seconds. This means that we only spend 0.0025 percent of the exhaustive algorithm's computation time and we are able to obtain a partition cost that is very close to the optimal partition cost. Also, for this virtual world, wherein avatars are uniformly distributed, if we have a more stringent interative requirement d^* (which dictates the number of iteration for the partition algorithm we presented in Section 3.3), we do not reduce the partitioning cost any further. This can be illustrated in Table 1b. This indicates that, with a modest computational time of 0.03 seconds, we can find the optimal partition policy.

2. However, as we will illustrate in other experiments (please refer to Table 2 and Table 3), further cost reduction can be achieved by executing LP and CRP algorithms after we obtain the initial partition using the RBP algorithm.

TABLE 1
Small Virtual World wherein Avatars Are Scattered under Uniform Distribution

Algorithm	Exhaustive	RBP	LP	CRP
System Cost $C_{\mathcal{P}}$	3.47	3.47	3.47	3.47
Computation Time (secs)	1202.12	0.01	<0.01	<0.01

(a)

Algorithm	$d^* = 0.1$	$d^* = 0.01$	$d^* = 0.001$	$d^* = 0.0001$
Exhuastive	3.47 (1202.12)	3.47 (1202.12)	3.47 (1202.12)	3.47 (1202.12)
Proposed Alg.	3.50 (<0.03)	3.47 (<0.03)	3.47 (<0.04)	3.47 (<0.04)

(b)

(a) System cost and computational time for the exhaustive and the proposed partitioning algorithm with $d^* = 0.01$.
(b) System cost and computational time under different stopping threshold d^*.

TABLE 2
Small Virtual World wherein Avatars Are Scattered under Skewed Distribution

Algorithm	Exhaustive	RBP	LP	CRP
System Cost $C_{\mathcal{P}}$	5.45	9.66	7.45	7.22
Computation Time (secs)	1290.88	0.01	0.01	<0.01

(a)

Algorithm	$d^* = 0.1$	$d^* = 0.01$	$d^* = 0.001$	$d^* = 0.0001$
Exhuastive	5.45 (1290.88)	5.45 (1290.88)	5.45 (1290.88)	5.45 (1290.88)
Proposed Alg.	8.56 (<0.03)	7.22 (<0.03)	6.32 (<0.04)	5.55 (<0.05)

(b)

(a) System cost and computational time for the exhaustive and the proposed partitioning algortithm with $d^* = 0.01$.
(b) System cost and computational time under different stopping threshold d^*.

TABLE 3
Small Virtual World wherein Avatars Are Scattered under Clustered Distribution

Algorithm	Exhaustive	RBP	LP	CRP
System Cost $C_{\mathcal{P}}$	4.39	9.42	7.12	7.01
Computation Time (secs)	1199.68	0.01	0.01	<0.01

(a)

Algorithm	$d^* = 0.1$	$d^* = 0.01$	$d^* = 0.001$	$d^* = 0.0001$
Exhuastive	4.39 (1199.68)	4.39 (1199.68)	4.39 (1199.68)	4.39 (1199.68)
Proposed Alg.	8.30 (<0.03)	7.01 (<0.03)	5.89 (<0.04)	4.97 (<0.06)

(b)

(a) System cost and computational time for the exhaustive and the proposed partitioning algorithm with $d^* = 0.01$.
(b) System cost and computational time under different stopping threshold d^*.

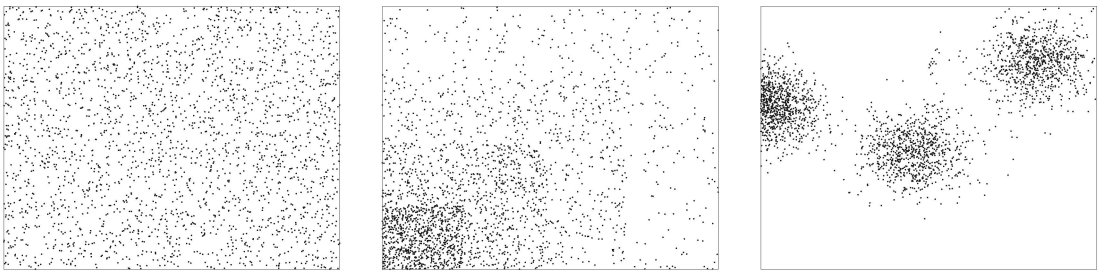


Fig. 9. Virtual world with 25×25 cells wherein avatars are scattered under (a) uniform, (b) skewed, and (c) clustered location distribution.

When the positions of the avatar are distributed under the skewed or clustered distribution, the exhaustive algorithm can find the optimal partition policy with costs of 5.45 and 4.39 and the computational times are $1,290.88$ and $1,199.68$ seconds. When we use our proposed partition algorithm with $d^* = 0.01$, we find the partition policies with costs of 7.22 and 7.01 and the computational times are 0.01 and 0.05 seconds, respectively. These results are illustrated in Table 2a and Tabel 3a, respectively. Table 2b and Table 3b illustrate that, if we increase the iteration threshold to $d^* = 0.0001$, we can obtain a partition cost that is very close to the optimal partition cost. These show the effectiveness of our partition algorithm, that is, we only need to spend less than 0.005 percent of the computational time of the exhaustive algorithm and we are able to find a partition policy that is very close to the optimal partition.

4.2 Experiment 2: Large Virtual World

In this experiment, we use a large virtual world with a dimension of 25×25 units with the total number of avatars equal to 2,500 and the number of servers, $P = 8$. The radius of the AOI of each avatar is equal to 0.5. Fig. 9 illustrates this virtual world under three different avatar's location distributions. Note that each point in these figures represents a 3-dimensional avatar in the virtual world.

Table 4, Table 5, and Table 6 illustrate the experimental results under the uniform, skewed, and clustered location distributions, respectively. Since the size of the virtual world is large, it is impossible to use the exhaustive algorithm (since it has 8^{2500} possible partition schemes). It is important to observe that, by applying the proposed algorithm, we can generate a good partition with much lower execution time. For all location distributions, *we iterate the partitioning algorithm three times*. For example,

TABLE 4
Experimental Results wherein Avatars Are Scattered under Uniform Distribution

Iteration Count	1	1	1	2	2	3	3
Algorithm	RBP	LP	CRP	LP	CRP	LP	CRP
System Cost $C_{\mathcal{P}}$	1160.03	443.58	441.06	399.61	399.61	358.07	358.07
Computation Time (secs)	162	0.41	0.38	0.4	0.26	0.35	0.24

TABLE 5
Experimental Results wherein Avatars Are Scattered under Skewed Distribution

Iteration Count	1	1	1	2	2	3	3
Algorithm	RBP	LP	CRP	LP	CRP	LP	CRP
System Cost $C_{\mathcal{P}}$	4405.73	2188.83	2188.83	2143.36	2143.36	2136.24	2136.24
Computation Time (secs)	226.1	0.39	0.39	0.36	0.33	0.35	0.31

TABLE 6
Experimental Results wherein Avatars Are Scattered under Clustered Distribution

Iteration Count	1	1	1	2	2	3	3
Algorithm	RBP	LP	CRP	LP	CRP	LP	CRP
System Cost $C_{\mathcal{P}}$	9714.83	6974.77	6531.38	6120.48	6114.32	5668.01	5668.01
Computation Time (secs)	166.5	0.33	0.39	0.36	0.31	0.36	0.27

TABLE 7
Experimental Results wherein Avatars Are Scattered under Uniform Distribution

Iteration Count	after we move avatars	1	1	2	2	3	3
Algorithm		LP	CRP	LP	CRP	LP	CRP
System Cost $C_{\mathcal{P}}$	4839.62	485.01	485.01	462.78	462.78	462.78	462.78
Computation Time (secs)	-	0.41	0.44	0.39	0.47	0.45	0.48

TABLE 8
Experimental Results wherein Avatars Are Scattered under Skewed Distribution

Iteration Count	after we move avatars	1	1	2	2	3	3
Algorithm		LP	CRP	LP	CRP	LP	CRP
System Cost $C_{\mathcal{P}}$	4905.73	2188.83	2188.83	2143.36	2143.36	2136.24	2136.24
Computation Time (secs)	-	0.43	0.42	0.40	0.35	0.38	0.34

TABLE 9
Experimental Results wherein Avatars Are Scattered under Clustered Distribution

Iteration Count	after we move avatars	1	1	2	2	3	3
Algorithm		LP	CRP	LP	CRP	LP	CRP
System Cost $C_{\mathcal{P}}$	34317.51	9160.67	8817.35	6643.21	6643.21	6643.21	6643.21
Computation Time (secs)	-	0.36	0.39	0.35	0.25	0.36	0.31

consider the scenario in Table 4. We use the RBP, LP, and CRP algorithms in the first iteration (as indicate by the iteration count) and, in the second and third iteration, we only use the LP and CRP algorithms. After three iterations, the system cost converges to a fixed value. For example, in Table 4, the system cost reduces from $1,160.03$ to 358.07 and it takes less than 165 seconds to obtain this system cost. For the scenario in Table 5, it takes a total of 228.23 seconds to obtain a system cost of $2,136.24$. For the scenario in Table 6, it takes a total of 118.52 seconds to obtain a system cost of 5,668.01. This shows that, for a large-scale DVE system, we can use our proposed partition algorithm to efficiently find a partition configuration that has a reasonable system cost.

4.3 Experiment 3: Dynamic Moving Avatars

In this experiment, we use the same setting as in Experiment 2. We assume that every 10 seconds, each avatar will move to a new position with a probability of 0.3 and stay in current position with a probability of 0.7. If an avatar should move, it will move to a random location which is within its AOI. Since the initial partition is already known (e.g., this is the partition we obtained from Experiment 2), all we need to perform is to iterate the LP and the CRP algorithms. Therefore, we start with the partition configuration that we obtained from Experiment 2 and, for every 10 seconds, we move each avatar to a new location with probability of 0.3. The second column in Table 7, Table 8, and Table 9 indicate that, right after the avatars' movement, there is a significant increase of system costs (as compared with the system cost in Experiment 2). We also periodically (every 10 seconds) apply our proposed iterative algorithm and, each time we invoke the partition algorithm, we iterate it three times. Table 7, Table 8, and Table 9 illustrate the experimental results under the uniform, skewed, and clustered location distributions, respectively. It is important to observe that, by applying the LP and CRP partitioning algorithms iteratively (in this case, we iterate three times only), we can get a good partition policy within short time intervals (less than 1.5 seconds).

4.4 Experiment 4: Dynamic Moving, Joining, and Leaving of Avatars

In this experiment, we use the same setting as the one in Experiment 3. That is, every 10 seconds, avatars will move to another position with a probability of 0.3 or stay in current position with a probability of 0.7. Moreover, 50 avatars will leave the virtual world and, at the same time, 150 avatars will join this virtual world. Again, the system cost converges to a small value after several seconds. Table 10, Table 11, and Table 12 illustrate the experimental results under the uniform, skewed, and clustered location distributions, respectively. It is important to observe that within a short period of time

TABLE 10
Experimental Results of Dynamic Join and Leave wherein Avatars Are Scattered under Uniform Distribution

Iteration Count	after moving, leaving or joining	1	1	2	2	3	3
Algorithm	Dynamic	LP	CRP	LP	CRP	LP	CRP
System Cost $C_{\mathcal{P}}$	1469.15	1215.09	1055.30	1055.30	1019.90	1019.90	1013.36
Computation Time (secs)	-	0.60	0.40	0.70	0.44	0.59	0.46

TABLE 11
Experimental Results of Dynamic Join and Leave wherein Avatars Are Scattered under Skewed Distribution

Iteration Count	after moving, leaving or joining	1	1	2	2	3	3
Algorithm	Dynamic	LP	CRP	LP	CRP	LP	CRP
System Cost $C_{\mathcal{P}}$	9066.94	6061.10	5849.96	5828.21	5461.38	5461.38	5456.71
Computation Time (secs)	-	0.63	0.44	0.62	0.52	0.64	0.50

TABLE 12
Experimental Results of Dynamic Join and Leave wherein Avatars Are Scattered under Clustered Distribution

Iteration Count	after moving, leaving or joining	1	1	2	2	3	3
Algorithm	Dynamic	LP	CRP	LP	CRP	LP	CRP
System Cost $C_{\mathcal{P}}$	39662.02	17799.05	10889.81	10825.43	10359.41	9825.01	8873.03
Computation Time (secs)	-	0.43	0.39	0.43	0.32	0.44	0.34

(less than three seconds), we can get a good partition scheme.

4.5 Experiment 5: Parallel Partitioning Algorithm

In this experiment, we use a very large virtual world, with a dimension of 30×30 units, the total number of avatars is equal to 25,000, and the number of servers, P, is equal to 16. The radius of the AOI of each avatar is equal to 0.5. Table 13, Table 14, and Table 15 illustrate the experimental results under the uniform, skewed, and clustered location distribution using the same setup as experiment 2. Since the size of the virtual world is 10 times larger than the large virtual world in experiment 2, we need to add more servers to maintain the virtual world. As both the number of clients and servers are large, the time needed in partitioning the virtual world also increases tremendously. To improve the performance, we use a parallel partitioning algorithm which is described in Section 3.4. It is important to observe that by using the parallel partitioning algorithm, we can generate a good partition with much lower execution time. First, we divide the very large virtual world and assign each small region of the virtual world to a server. For all divided virtual worlds, we iterate the partitioning algorithm twice. During the first iteration, we execute the RBP, LP, and CRP algorithms. For the second iteration, we only execute the LP and CRP algorithms. After two iterations, we observe that the parallel partitioning algorithm can have a smaller system cost as compared to the sequential partitioning algorithm used in experiment 2. Another important point to note is that the parallel partitioning algorithm only requires a very small fraction of processing time, as compared with the sequential partition algorithm. Table 16, Table 17, and Table 18 illustrate the

TABLE 13
Uniform Distribution

Iteration Count	1	1	1
Algorithm	RBP	LP	CRP
Cost	73013.99	40453.82	38656.35
Time(s)	641.22	44.91	38.02

30×30 *cells, 16 partitions, 25,000 avatars*

TABLE 14
Skewed Distribution

Iteration Count	1	1	1
Algorithm	RBP	LP	CRP
Cost	195224.53	130151.50	126849.17
Time(s)	811.00	34.61	31.10

30×30 *cells, 16 partitions, 25,000 avatars*

TABLE 15
Clustered Distribution

Iteration Count	1	1	1
Algorithm	RBP	LP	CRP
Cost	265481.66	163869.56	157648.94
Time(s)	749.25	51.13	35.82

30×30 *cells, 16 partitions, 25,000 avatars*

TABLE 16
Combined Uniform World

Iteration Count	1	2	2
Algorithm	Parallel(RPB+LP+CRP)	Parallel LP	Parallel CRP
Cost	56931.76	37818.00	37694.41
Time(s)	7.00	44.61	27.59

30×30 *cells, 16 partitions, 25,000 avatars*

TABLE 17
Combined Skewed World

Iteration Count	1	2	2
Algorithm	Parallel(RPB+LP+CRP)	Parallel LP	Parallel CRP
Cost	128759.35	81014.57	80542.74
Time(s)	8.00	25.22	32.28

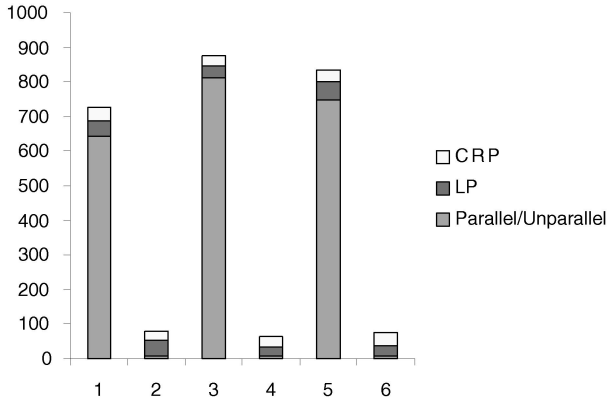
30×30 *cells, 16 partitions, 25,000 avatars*

TABLE 18
Combined Clustered World

Iteration Count	1	2	2
Algorithm	Parallel(RPB+LP+CRP)	Parallel LP	Parallel CRP
Cost	172824.14	148949.02	145532.83
Time(s)	9.00	28.50	37.61

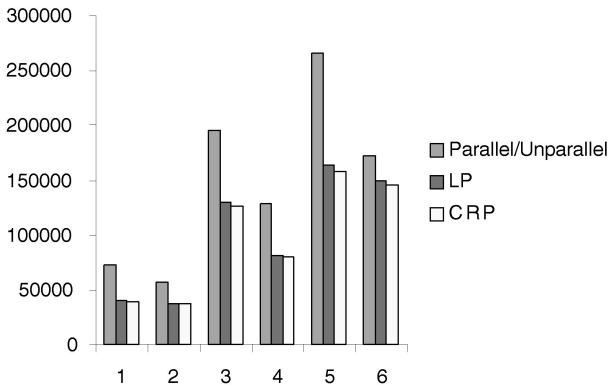
30×30 *cells, 16 partitions, 25,000 avatars*

experimental results. Fig. 10 illustrates the processing time under different approaches. As we can observed, the parallel partitioning algorithm achieves a significant computational reduction as compare to the sequential partitioning algorithm. Fig. 11 illustrates the partitioning cost under different approaches. As we can observe, on each iteration, both the sequential and the parallel partitioning algorithm can reduce the overall system cost and that their system



Distribution	Uniform		Skewed		Clustered	
Index on x-axis	1	2	3	4	5	6
Algorithm	Sequential	Parallel	Sequential	Parallel	Sequential	Parallel
Time	641.22	7.0	811.00	8.0	749.25	9.0
Time(LP)	44.91	44.61	34.61	25.22	51.13	28.50
Time(CRP)	38.02	27.59	31.10	32.28	35.82	37.61

Fig. 10. Processing time under different approaches.



Distribution	Uniform		Skewed		Clustered	
Index on X-axis	1	2	3	4	5	6
Algorithm	Sequential	Parallel	Sequential	Parallel	Sequential	Parallel
Initial System Cost	73013.99	56931.76	195224.53	128759.35	265481.66	172824.14
Cost by applying LP	40453.82	37818.00	130151.50	81014.57	163869.56	148949.02
Cost by applying CRP	38656.35	37694.41	126849.17	80542.74	157648.94	145532.83

Fig. 11. System cost under different approaches.

costs are comparable. However, because the parallel partitioning algorithm is more computational efficient, we can apply this partitioning strategy for a large scale DVE system.

5 Conclusion

With the advances in multimedia systems, parallel/distributed database systems and high speed networking technologies, DVE systems are becoming increasingly common in the scientific, industrial, and entertainment industries. There is a growing need to increase the realism and fidelity of DVE systems. Modeling and implementing special effect, such as real-time atmospheric scenes, including wind, smoke, clouds, haze, rain, and snow, will produce a flood of traffic that may exceed computational and communication capacity of a DVE system. In this paper, we discuss the scalability problem in DVE and present related techniques to solve this problem. To build a scalable DVE system, we use a multiple servers DVE architecture. Under the MSDVE architecture, there is a necessity to balance the computational workload and, at the same time, reduce the communication cost of a DVE system. We formulate the partitioning problem and show that it is NP-complete in general. We then propose a computation efficient partitioning algorithm so that we can quickly obtain an efficient partitioning policy $\mathcal{P}$. Our experiments show that our propose algorithm can achieve a significant reduction in both communication and computation cost. We also illustrate that we can adopt the partitioning algorithm to a virtual world wherein 1) avatars can dynamically move from one position to another position in the virtual world and 2) avatars can dynamically join or leave the virtual world. We present the parallel version of the partitioning algorithm so as to obtain a partition policy for a very large scale virtual world that allows many clients and dynamic objects. The techniques we discuss and the partitioning algorithm we propose can address the scalability issue of designing a large scale DVE system.

Acknowledgments

The authors would like to thank the anonymous referees for their helpful and insightful suggestions. This work is supported in part by the Research Grants Council Earmarked Grant and the Chinese Univeristy Hong Kong Direct Research Grant.

References

[1] S.T. Barnard and H.D. Simon, "Fast Multilevel Implementation of Recursive Spectral Bisection for Partitioning Unstructured Problems," *Concurrency: Practice and Experience,* vol. 6, no. 2, pp. 101-117, 1994.

[2] S.T. Barnard and H.D. Simon, "A Parallel Implementation of Multilevel Recursive Spectral Bisection for Application to Adaptive Unstructured Meshes," *Proc. Seventh SIAM Conf. Parallel Processing for Scientific Computing,* pp. 627-632, Feb. 1995.

[3] T. Ballardie, P. Francis, and J. Crowcroft, "Core Based Tree: An Architecture for Scalable Multicast Routing," *Proc. ACM SIGCOMM '93,* pp. 85-95, Sept. 1993.

[4] F. Ercal, J. Ramanujam, and P. Sadayappan, "Task Allocation onto a Hypercube by Recursive Min-Cut Bipartitioning," *J. Parallel and Distributed Computing,* vol. 10, pp. 35-44, 1990.

[5] G. Fox, *Graphical Approach to Load Balancing and Sparse Matrix Vector Multiplication on the Hypercube.* M. Schultz, ed., Berlin: Springer-Verlag, 1988.

[6] G. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salmon, and D. Walker, *Solving Problems on Concurrent Processors,* Vol I. Prentice Hall, 1988.

[7] P.A. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency Control and Recovery in Database Systems.* Addison Wesley, 1987.

[8] W. Broll, "Distributed Virtual Reality for Everyone: A Framework for Networked VR on the Internet," *Proc. IEEE Virtual Reality Ann. Int'l Symp.,* Mar. 1997.
[9] D.E. Comer, *Internetworking with TCP/IP: Volume I, Principles, Protocols, and Architecture.* Prentice Hall, 1995.
[10] S.E. Deering and D.R. Cheriton, "Multicast Routing in Datagram Internetworks and Extended LANs," *ACM Trans. Computer Systems,* vol. 8, pp. 85-110, May 1990.
[11] V. Firoiu and D. Towsley, "Call Admission and Resource Reservation for Multicast Sessions," *Proc. 15th Ann. Joint Conf. IEEE Computer and Comm. Soc. (INFOCOM '96),* 1996.
[12] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, 1978.
[13] J.R. Gilbert, G.L. Miller, and S.H. Teng, "Geometric Mesh Partitioning: Implementation and Experiments," *Proc. Ninth Int'l Parallel Processing Symp.,* pp. 418-427, Apr. 1995.
[14] B. Hendrickson and R. Leland, "An Improved Spectral Load Balancing Method," *Proc. Sixth SIAM Conf.,* pp. 953-961, 1993.
[15] B. Hendrickson and R. Leland, "Multidimensional Spectral Load Balancing," technical report, Sandia National Laboratory, Albuquerque, 1993.
[16] B. Hendrickson and R. Leland, "An Improved Spectral Graph Partitioning Algorithm for Mapping Parallel Computations," *SIAM J. Scientific Computing,* vol. 16, no. 2, pp. 452-469, 1995.
[17] B. Hendrickson and R. Leland, "An Multilevel Algorithm for Partitioning Graphs," *Proc. Supercomputing,* Dec. 1995
[18] B.W. Kernigham and S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs," *Bell Systems Technology J.,* vol. 49, no. 2, pp. 292-370, 1970.
[19] C.-J. Liao and Y.-C. Chung, "Tree-Based Parallel Load-Balancing Methods for Solution Adaptive Finite Element Graphs on Distributed Memory Multicomputers," *IEEE Trans. Parallel and Distributed Systems,* vol. 10, no. 4, Apr. 1999.
[20] J.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," *Proc. Third Int'l Workshop Multimedia Information Systems (MIS '97),* 1997.
[21] J.C.S. Lui, M.F. Chan, K.Y. So, and T.S. Tam, "Balancing Workload and Communication Cost for a Distributed Virtual Environment," *Proc. Fourth Int'l Workshop Multimedia Information Systems (MIS '98),* 1998.
[22] J.C.S. Lui, O.K.Y. So, and P.T.S. Tam, "Deriving Communication Sub-graph and Optimal Synchronizing Interval for a Distributed Virtual Environment System," *Proc. IEEE Int'l Conf. Multimedia Systems,* June 1999.
[23] J.C.S. Lui, "Constructing Communication Subgraphs and Deriving an Optimal Synchronization Interval for Distributed Virtual Environment Systems," Accepted for publication, *IEEE Trans. Data and Knowledge Eng.,* 2001.
[24] M.R. Macedonia, D.P. Brutzman, M.J. Zyda, D.R. Pratt, P.T. Barham, J. Falby, and J. Locke, "NPSNET: A Multi-Player 3D Virtual Environment over the Internet," *Proc. ACM Symp. Interactive 3D Graphics,* Apr. 1995.
[25] M.R. Macedonia and M.J. Zyda, "A Taxonomy for Networked Virtual Environments," *Proc. IEEE Int'l Conf. Multimedia Systems,* 1996.
[26] M.R. Macedonia, M.J. Zyda, D.R. Pratt, and P. Barham, "Exploiting Reality with Multicast Groups: A Network Architecture for Large Scale Virtual Environments," *Proc. 11th DIS Workshop Standards for the Interoperability of Distributed Simulation,* pp. 503-510, Sept. 1994.
[27] R. Muntz, J.R. Santos, and S. Berson, "RIO: A Real-Time Multimedia Object Server," *ACM Performance Evaluation Rev.,* vol. 25, no. 2, pp. 29-35, Sept. 1997.
[28] R.R. Muntz, J. Renato Santos, and S. Berson, "A Parallel Disk Storage System for Realtime Multimedia Applications," *J. Intelligent Systems,* vol. 13, no. 12, 1998.
[29] B. Nour-Omid, A. Raefsky, and G. Lyzenga, "Solving Finite Element Equations on Current Computers," *Parallel Computations and their Impact on Mechanics,* pp. 299-227, 1986.
[30] C.-W. Ou and S. Ranka, "Parallel Incremental Graph Partitioning," *IEEE Trans. Parallel and Distributed Systems,* vol. 8, no. 8, Aug. 1997.
[31] O. Stahl and M. Anderson, "DIVE: A Toolkit for Distributed VR Applications," Swedish Inst. of Computer Science, SICS. http://www.sics.se/dive/description.html. year???
[32] H. Simon, "Partitioning of Unstructured Mesh Problems for Parallel Processing," *Proc. Conf. Parallel Methods on Large Scale Structural Analysis and Physics Applications,* 1991.
[33] M.M. Syslo, N. Deo, and J.S. Kowalik, *Discrete Optimization Algorithms.* Prentice-Hall, 1983.
[34] R.C. Waters and J.W. Barrus, "The Rise of Shared Virtual Environments," *IEEE Spectrum,* pp. 20-25, Mar. 1997.
[35] W.L. Winston, *Introduction to Mathematical Programming—Applications and Algorithms.* Duxbury Press, 1995.
[36] R. Williams, "Performance of Dynamic Load-balancing Algorithm for Unstructured Mesh Calculations," *Concurrency,* vol. 3, pp. 457-481, 1991.

John C.S. Lui received the PhD degree in computer science from the University of California, Los Angeles. While he was a graduate student, he participated in a parallel database project in the IBM Thomas J. Watson Research Center. After his graduation, he joined a team at the IBM Almaden Research Laboratory/San Jose Laboratory and participated in research and development of a parallel I/O architecture and file system project. He later joined the Department of Computer Science and Engineering of the Chinese University of Hong Kong. His current research interests are in communication networks, distributed multimedia systems, OS design issues, parallel I/O, storage architectures, and performance evaluation theory. He is a member of Tau Beta Pi, the ACM, the IEEE, and the IEEE Computer Society. His personal interests include general reading and films.

M.F. Chan received the Bachelor's and the Master's degrees in computer science from the Chinese University of Hong Kong. While he was a graduate student, he participated in a Distributed Virtual Environment project led by professor John C.S. Lui. After his graduation, he worked as an IT Coordinator to promote IT education in Hong Kong. He later joined Poly-Asia computer company as a senior consultant. His research interests have centered around distributed VR systems, networks communication, and security. He is a student member of the IEEE.