On View Consistency in Multi-Server Distributed Virtual Environments

Haiyang Hu, Rynson W.H. Lau, *Senior Member, IEEE*, Hua Hu, and Benjamin Wah *Fellow, IEEE*,

Abstract—A distributed virtual environment (DVE) is a shared virtual environment (VE) that allows remote users to interact with each other through networks. DVEs are becoming very popular due to some prominent applications, such as online games and virtual worlds. To support a large number of users, a multi-server DVE architecture may be adopted, with each server managing a subset of users. However, there are two critical problems with this architecture: view inconsistency caused by delays and server overloading caused by uneven distribution of users. While the first problem affects users' perception of the VE and causes user disputes, the second problem affects the system response time. In this paper, we first show that the view inconsistency problem and the load balancing problem are conflicting objectives. We then propose an efficient joint optimization framework to address both problems. Our results show that the proposed method can improve the view inconsistency problem significantly, which is important to the interactivity of DVE applications.

Keywords—distributed virtual environments, view consistency, DVE load balancing, multi-server architecture.

1 Introduction

Distributed virtual environments (DVEs) have emerged as a new technology in distributed applications, due to advances in networking, computer graphics, and distributed systems technologies. A DVE system allows remote users to interact with each other in a shared virtual environment (VE) through networks, particularly the Internet. Each user of a DVE system may move around in the VE, communicate with other users, and inquire the states of objects in the VE. DVE systems are now widely used in various applications, such as online training, collaborative design, and multiplayer online games [30].

Due to the widely availability of the Internet, some DVE applications have grown to become very large with a huge number of users from all over the world. In order to support such a large user population, a multi-server architecture is typically adopted. As an example, for large games, a popular approach to handle large user population is to partition them to different servers. As the users join the game, they are immediately assigned to a server. When this server is full, a new server is started to serve additional users. This approach is adopted by most commercial games, such as Quake III Arena (www.idsoftware.com) and Diablo II (www.blizzard.com). Its main advantages are its simplicity and efficiency. However, it has two major limitations. First, as each server is running a separate instance of the game, users served by different servers may not be able to interact with each other. Second, once a user is assigned to a server, it cannot be changed. Hence, if a user suddenly invokes some computationally costly operations causing server overloading, it may not be trivially handled. Another approach is to partition the VE into static regions, with each region served by one server. Hence, users may be served by different servers depending on where they are in the VE. This approach is adopted by some commercial games, such as EverQuest (everquest.station.sony.com), Ultima Online (uo.com) and Asherons Call (ac.turbinegames.com). This approach is also simple and efficient. However, when a lot of users move into the same region, the server serving this region can still get overloaded. In addition, as this approach divides users according to their virtual locations, not physical geographical locations, users served by the same server may be coming from different parts of the world and hence suffer from very different amounts of network delay. This can significantly affect their interactions, caused by the view inconsistency problem, as explained below.

In a DVE system, whenever a user (referred to as **A**) changes its state, e.g., making a move, **A** needs to send an update message to other users who are near to **A** in the VE. We refer to these users as as relevant users to **A**. A typical approach to determine these relevant users is by defining a circular region around **A**, referred to as **A**'s area of interest (AOI) [30].

In designing a multi-server DVE system that is scalable to the number of users, we have two main challenges. First, it is important for the system to maximize "view consistency" of all users, i.e., if **A** changes its state, all relevant users should receive the corresponding update message in a timely manner so that their views

Manuscript submitted on January 24, 2013.

- *Haiyang Hu is with both the School of Computer Science and Technology, Hangzhou Dianzi University, China, and the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications and the State Key Laboratory for Novel Software Technology, Nanjing University, China. E-mail: huhaiyang@hdu.edu.cn*
- *Rynson Lau is with the Department of Computer Science, City University of Hong Kong, HK. E-mail: rynson.lau@cityu.edu.hk*
- *Hua Hu is with the School of Computer Science and Technology, Hangzhou Dianzi University, China. E-mail: huhua@hdu.edu.cn*
- *Benjamin Wah is with the Chinese University of Hong Kong, HK. E-mail: bwah@cuhk.edu.hk*

of the VE agree with **A**. When this is not the case, some users may be acting according to incorrect views, which may cause disputes among them. If all relevant users are served by the same server, any update messages sent from the client (i.e., the client machine) of **A** involve only client-server delay. However, if the relevant users are distributed in multiple servers, any update messages sent from **A** involve both client-server and server-server delays. To minimize server-server delays, we may place all servers together in the same site. This will reduce server-server delays to nearly neglectable values. However, this will increase the client-server delays significantly as the users may be coming from anywhere in the world. To minimize client-server delays, we may distribute the servers across the globe. However, the server-server delays will no longer be neglectable then. Second, it is important to avoid server overloading, which would affect user interactions. A multi-server DVE should have the ability to efficiently redistribute workloads among servers so as to minimize the number of overloaded servers.

Unfortunately, these two challenges have conflicting objectives. As we assign more users, i.e., loads, to the same server in order to minimize view inconsistency, this server will take up a higher load than other servers and hence become overloaded easier. On the other hand, as we partition the users equally among all servers to minimize the chance of server overloading, more update messages will need to be sent among the servers, which involve server-server delays. Hence, more users will suffer from the view inconsistency problem.

To address these two problems, we need a load balancing method that would minimize both view inconsistency and the chance of server overloading. There are a number of multi-server load balancing methods proposed for DVEs. Most of them focus on balancing the workload and communication costs, and neglect the view inconsistency problem. In this paper, we present a novel joint optimization framework to address the above two problems. Our main contributions are:

- We classify different types of user movement in order to quantify the view inconsistency problem, by estimating the rate at which view inconsistency events are perceived by users. (See Section 4.)
- We present a joint optimization framework to find the proper trade-offs between view consistency and server workload. By dynamically dividing the VE into partitions, the objective of the framework is to reduce view inconsistency brought by the server-server delay while keeping the workload of each partition under a given threshold. (See Section 5.1.)
- We present a Kuhn-Munkres based algorithm to assign the partitions to different servers so as to reduce view inconsistency brought by the client-server delay, and a greedy algorithm to locally refine the partition-to-server assignment in order to reduce view inconsistency brought by both client-server and server-server delays. (See Section 5.2.)

To the best of our knowledge, this is the first work on joint optimization of view consistency and load balancing in multi-server DVEs. Note that although cloud services are becoming popular [1], they also suffer from the two problems as multi-server DVEs when multiple servers are involved in providing the service. Hence, the joint optimization framework that we present here is also applicable in a cloud environment.

The rest of this paper is organized as follows. Section 2 reviews related work on load balancing in multi-server DVEs. Section 3 introduces our DVE model. Section 4 analyzes different types of user movement and quantize the view inconsistency problem caused. Section 5 presents optimization algorithms to divide the VE into partitions and then assign the partitions to different servers so as to minimize client-server and server-server delays. Section 6 discusses a number of experiments to evaluate the effectiveness of the proposed method.

2 Related Work

There has been a lot of research on DVE load balancing. A straightforward approach, referred to as *user partitioning*, is by dividing users directly among the servers. In [34], a direct client assignment method is proposed to minimize the total amount of network latency among clients and servers. Each server manages a copy of the whole VE. When assigning a new client to a server, it estimates the total amount of latency between the new client and each of the clients already assigned. The objective is to assign the new client to a server with a minimum amount of latency to all the assigned clients, which includes both client-server and server-server latencies. The main limitation of this method is that since neighboring users in the VE may potentially be assigned to different servers, many more update messages need to be sent across servers and a higher view inconsistency problem is expected.

Another approach, referred to as *spatial partitioning*, is by dividing the VE into partitions, each served by a different server. In general, spatial partitioning methods can be roughly classified into two types, global load balancing methods and local load balancing methods. Global load balancing methods make use of the load information of all servers to compute load balancing solutions, while local load balancing methods only make use of the load information of nearby servers to compute load balancing solutions. In the following two subsection, we review these two types of load balancing methods. We then review works that address the synchronization or view inconsistency problem of DVEs.

2.1 Global Load Balancing

In [23], a global method is proposed that models the users (or nodes) of a VE as a connected graph, with each edge indicating the communication cost between two connected users. The graph is then partitioned among the servers to achieve load balancing. The load of each region is computed based on the number of users in it and the inter-server communication costs involved. As the optimization process of this method involves all the nodes, it is very slow. In [25], two partitioning algorithms based on heuristic search and genetic algorithm are proposed. They use the aggregated CPU bandwidth of the servers to avoid system saturation and try to cluster nearby users to the same server to reduce the latency. In [32], the entire VE is divided into regular cells. Greedy, simulated annealing and integer linear programming algorithms are used to determine the best way to assign the cells to different servers based on some global load information. However, these algorithms have high computational costs, especially for the integer linear programming algorithm, which produces globally optimal solutions. Although [2] attempts to speed up [32] by trying to obtain suboptimal solutions instead of globally optimal solutions, it is still too slow for large-scale DVEs.

In general, global methods apply optimization techniques to achieve well balanced load distribution while minimizing the communication costs among the servers. However, optimization techniques typically have high computational costs and hence are slow.

2.2 Local Load Balancing

In [27] [28], an efficient local method is proposed. When a server is overloaded, it identifies the neighboring servers with the lightest loads for load redistribution. To minimize the number of users located at partition boundaries, it also tries to minimize the perimeter length of each partition. Although this method is very efficient, it may not be able to disperse the load quickly if neighboring servers also have high loads. In [17], a revised method is proposed to address this limitation. Instead of just finding neighboring servers, this method identifies a set of connected servers to the overloaded server and performs optimized repartitioning of the corresponding partitions to achieve better load balancing. As a result, this method has a higher computational cost than [27]. In [6], a local method that considers QoS is proposed. Each server monitors its own QoS violations, measured in terms of user response time. It determines if a perceived QoS violation is due to heavy workload or high inter-server communication, and then triggers either load shedding or load aggregation.

In general, local methods are efficient as they simply redistribute the extra load to neighboring servers. However, as they produce mainly short-term solutions, they are usually less effective than the global methods [8]. On the other hand, although both view consistency and interactivity are important to users, all the above methods (global and local) mainly focus on maintaining interactivity through load balancing and neglect view consistency. This paper attempts to address both problems with an efficient joint optimization framework.

2.3 Synchronization and View Consistency

Synchronization schemes are widely used in distributed systems to ensure state consistency and events being processed in correct temporal order. There are two kinds of synchronization approaches [10], *conservative* and *optimistic*. While conservative synchronization requires all events to be processed in causal order and allows no violations to occur [3] [24], optimistic synchronization allows violations to occur, but mechanisms are provided to detect and correct the violations [13] [18] [19] [33].

View consistency may be considered as a specific type of synchronization. It mainly concerns if users' views, i.e., the contents shown in their AOIs, are consistent with each other. In [35], a method is proposed to estimate the amount of time-space inconsistencies between any two users by considering factors such as clock asynchrony and communication delay. The objective of this work is mainly to characterize the time-space inconsistency problem rather than to find a way to minimize it. Based on [35], [31] assumes that users' positions are predictable, i.e., can be modeled as a function of time, and proposes an update mechanism to help each server decide which entities to update in each frame, so as to reduce inconsistency under the constraint of network capacity. In [22], an update scheme is proposed to construct a graph to represent the communication delay between any two users, and then derive an optimal interval for sending update messages based on Markov chains and the properties of the fundamental matrix. In [20], a method is proposed to maintain the time-space consistency of users and their replicas by adapting each server's update schedule.

In summary, existing works on view consistency mainly focus on finding an appropriate error threshold for dead-reckoning schemes to send out update messages. These dead-reckoning schemes typically apply polynomial predictors to extrapolate object positions in order to compensate for the network latency, which help reduce the inconsistency problem. However, there are two fundamental limitations to this approach. First, they are effective only when the object motion is known in advance, i.e., predictable, which is often not the case for user movements in DVEs [4] [5]. Second, it is difficult to predict any sudden change of motion [18] [19], e.g., when a user will suddenly start/stop moving. In contrast, the proposed method tries to reduce the overall network latency, and hence the total view inconsistency, through a joint-optimization scheme.

A concurrent work [21] to ours also tries to address the inconsistency problem. It formulates the partitioning problem as a mix integer programming problem and proposes an Iterative Partitioning Algorithm (IPA) based on alternating optimization to minimize the total time-space inconsistency of the DVE. Similar to the dead-reckoning schemes, this method assumes that object motion is predictable. In contrast, our work focuses on developing a load balancing method to reduce view inconsistency due to user movements and network latency. We do not assume any user motion behaviors, as user movements are often unpredictable.

3 Our DVE Model

In our DVE model, we divide a VE regularly into squared virtual cells $\{c_i\}$, to avoid the partitioning process producing completely arbitrary partition boundaries. These virtual cells are partitioned among the servers $\{s_j\}$. The servers may be geographically distributed and each one manages its assigned virtual cells and the users inside. Each user has a position, $p = (x, y)$, in the VE and an AOI (area of interest). To simplify our discussion, we assume the size of the AOI for all users to be the same, and approximate the AOI as a square of dimension $2r \times 2r$, with the user being at the center.

If a user a_k is located inside a virtual cell c_i and managed by server s_m, s_m is referred to as a_k's *local server* while a_k is referred to as s_m's *local user*. Whenever a_k changes its state, it sends an update message to notify its local server, which will then forward the message to a_k's relevant users. If a_k is located near to c_i's boundary, its AOI may contain some users located in the adjacent virtual cell, c_j. In the case where c_j is also managed by s_m, s_m simply forwards the message from a_k to these users. However, if c_j is managed by another server s_n, i.e., this virtual cell boundary is also a partition boundary, s_m will forward the update message to s_n and then s_n to the relevant local users. If a_k suddenly moves outside of c_i into c_j, which is managed by s_n, then s_m will send a *transfer packet* to s_n to transfer a_k to s_n.

Under this scenario, view inconsistency (VI) among users can be caused by two types of network delay:

1) Between clients and a local server - We refer to the VI incurred as *local view inconsistency* (or local VI).
2) Between any two servers that an update message is sent across - We refer to the VI incurred as *remote view inconsistency* (or remote VI).

4 Modeling of Remote and Local VI's

Based on the DVE model in Section 3, we classify different types of user movement in a single time step, quantify the remote and local VIs, and then estimate the total amount of VI problem in DVEs in this section.

We consider the situation where two adjacent virtual cells, c_i^m and c_j^n, are managed by two servers, s_m and s_n, respectively. We define a boundary region $R1$ of width r in c_i^m (or simply c_i) that is adjacent to c_j^n (or simply c_j) and a boundary region $R2$ in c_j adjacent to $R1$, as shown in Fig. 1. As the size of the AOI is $2r \times 2r$, if a user a_k is located at p within $R1$, part of its AOI will fall inside c_j. As such, a_k may be visible to some users located near to the boundary of c_j, e.g., a_2 and a_3 in Fig. 1. Here, the AOI of a_k at position p, i.e., AOI(p), is composed of two parts, the part that falls inside c_i (or $AOI1$(p)) and the part that falls inside c_j (or $AOI2$(p)).

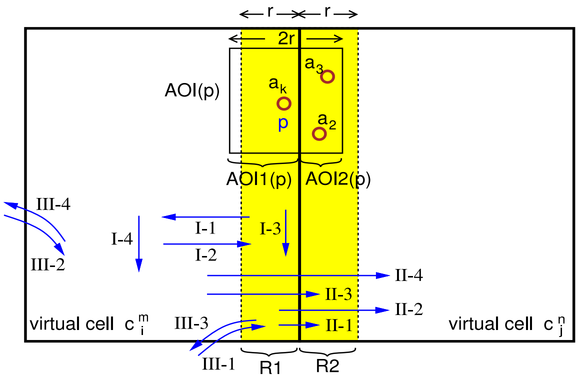


Fig. 1. Different types of user movements.

In general, if a user moves from/to a position in the border region, i.e., $R1$ of c_i, it will cause local VI to the relevant users in c_i and remote VI to the relevant users in c_j. If it moves from/to the non-border region, i.e., $c_i \setminus R1$, it will only cause local VI to the relevant users in c_i. If a_k is currently located in c_i, we can classify a_k's movement between two consecutive time frames into three main types (referring to Fig. 1) as follows:

Type I: a_k moves within the same virtual cell, c_i.

- **Case I-1**: a_k moves from $R1$ of c_i to $c_i \setminus R1$.
- **Case I-2**: a_k moves from $c_i \setminus R1$ to $R1$ of c_i.
- **Case I-3**: a_k moves within $R1$ of c_i.
- **Case I-4**: a_k moves within $c_i \setminus R1$.

Type II: a_k moves into the adjacent virtual cell, c_j.

- **Case II-1**: a_k moves from $R1$ of c_i to $R2$ of c_j.
- **Case II-2**: a_k moves from $R1$ of c_i to $c_j\setminus R2$.
- **Case II-3**: a_k moves from $c_i \setminus R1$ to $R2$ of c_j.
- **Case II-4**: a_k moves from $c_i \setminus R1$ to $c_j\setminus R2$.

Type III: a_k teleports into or out of c_i.

- **Case III-1**: a_k joining the DVE or jumping from a distant virtual cell into $R1$ of c_i.
- **Case III-2**: a_k joining the DVE or jumping from a distant virtual cell into $c_i \setminus R1$.
- **Case III-3**: a_k leaving the DVE or jumping out to a distant virtual cell from $R1$ of c_i.
- **Case III-4**: a_k leaving the DVE or jumping out to a distant virtual cell from $c_i \setminus R1$.

In the following subsections, we first quantify the remote VI between two adjacent cells for Type-I, Type-II and Type-III movements in Sections 4.1, 4.2 and 4.3, respectively. We then quantify the local VI inside a single cell in Section 4.4. We will use a stochastic approach to model the VI events occurring between each pair of adjacent cells, due to different types of user movement, which involve statistical distributions. Finally, we estimate the total amount of VI in the DVE in Section 4.5.

4.1 Type I: Moving Within a Cell

In this subsection, we consider the remote VI caused by users moving within cell c_i (managed by server s_m) to observing users in cell c_j (managed by server s_n).

Consider the remote VI caused by a user a_k moving within c_i to an observing user a_o located in c_j. The message transmission latencies include from a_k to s_m, s_m to s_n and then s_n to a_o. Let the average delay between users in c_i and s_m be L_i^m, the average delay between users in c_j and s_n be L_j^n, and the average delay between s_m and s_n be $T_{m,n}$. Hence, the total message delay is:

$$\psi(c_i^m, c_j^n) = L_i^m + T_{m,n} + L_j^n$$

During this period, a_o still see a_k in its original location, which is incorrect. we refer to this as a *view-inconsistent event* (or VI event). Let β_j be the rate at which users in the DVE query for the states of other users inside their own AOIs. (β_j is typically set to the frame rate.) The expected number of VI events observed by a_o as a result of a_k's movement can be approximated by $\beta_j \psi(c_i^m, c_j^n)$.

To quantify the amount of remote VI between c_i and c_j, we define $\rho_i(\mathrm{p})$ as the probability of a user in c_i being located at p. The probability, $J_j(\mathrm{p})$, of a user in c_j being inside $AOI2(\mathrm{p})$, when a_k is located at p, is then:

$$J_j(\mathrm{p}) = \textstyle\sum_{(x,y)\in AOI2(\mathrm{p})} \rho_j((x,y)) \tag{1}$$

The probability of a user in c_j being inside $AOI2(\mathrm{p}) \cup AOI2(\mathrm{p}')$ is then:

$$I_j(\mathrm{p},\mathrm{p}') = \textstyle\sum_{(x,y)\in AOI2(\mathrm{p})\cup AOI2(\mathrm{p}')} \rho_j((x,y)) \tag{2}$$

where $I_j(\mathrm{p},\mathrm{p}')$ models the four Type I cases. For case I-1, the area of $AOI2(\mathrm{p}')$ in Eq. (2) will be zero and the probability value of $I_j(\mathrm{p},\mathrm{p}')$ becomes $J_j(\mathrm{p})$. For case I-2, the area of $AOI2(\mathrm{p})$ in Eq. (2) will be zero and the probability value of $I_j(\mathrm{p},\mathrm{p}')$ becomes $J_j(\mathrm{p}')$. For case I-3, the areas of both $AOI2(\mathrm{p})$ and $AOI2(\mathrm{p}')$ will be non-zero. For case I-4, the area of $AOI2(\mathrm{p}) \cup AOI2(\mathrm{p}')$ in Eq. (2) will be zero and the probability value of $I_j(\mathrm{p},\mathrm{p}')$ becomes zero, indicating that no users in c_j will perceive remote VI events caused by a_k.

Let $m_{j\to j}$ be the portion of users in c_j that move only within c_j at anytime and $N_j(t^-)$ be the number of users in c_j just before time t. The number of users remaining in c_j at time t is then $R_j(t) = N_j(t^-)m_{j\to j}$. If we further let $\rho_{\mathrm{p}\to\mathrm{p}'}$ be the probability of a user moving from p to p′, the expected number of remote VI events received by users in c_j due to one user moving within c_i is:

$$Q_1(c_i^m, c_j^n) = \psi(c_i^m, c_j^n) R_j(t) \sum_{\mathrm{Cond1}} \rho_i(\mathrm{p})\rho_{\mathrm{p}\to\mathrm{p}'} I_j(\mathrm{p},\mathrm{p}')\beta_j \tag{3}$$

where
Cond1 $= \mathrm{p} \in R1 \wedge \mathrm{p}' \in c_i \setminus R1$, for case I-1
Cond1 $= \mathrm{p} \in c_i \setminus R1 \wedge \mathrm{p}' \in R1$, for case I-2
Cond1 $= \mathrm{p} \in R1 \wedge \mathrm{p}' \in R1$, for case I-3
Cond1 $= \mathrm{p} \in c_i \setminus R1 \wedge \mathrm{p}' \in c_i \setminus R1$, for case I-4.

As the expected number of users remaining in c_i at time t is $R_i(t) = N_i(t^-)m_{i\to i}$, the expected rate at which remote VI events occur in c_j caused by users in c_i moving within c_i is then:

$$vi_1(c_i^m, c_j^n) = \lim_{\tau\to\infty} \frac{1}{\tau} \int_0^\tau R_i(t) Q_1(c_i^m, c_j^n)\delta t \tag{4}$$

which can also be represented by

$$vi_1(c_i^m, c_j^n) = \psi(c_i^m, c_j^n)\eta_1(c_i, c_j) \tag{5}$$

where

$$\eta_1(c_i, c_j) = \lim_{\tau\to\infty} \tfrac{1}{\tau} \textstyle\int_0^\tau R_i(t)R_j(t) \displaystyle\sum_{\mathrm{Cond1}} \rho_i(\mathrm{p})\rho_{\mathrm{p}\to\mathrm{p}'} I_j(\mathrm{p},\mathrm{p}')\beta_j\delta t$$

4.2 Type II: Moving to an Adjacent Cell

In this subsection, we consider the remote VI caused by users moving from cell c_i (managed by server s_m) to an adjacent cell c_j (managed by server s_n), to observing users in c_j.

If we let $m_{i\to j}$ be the portion of users in c_i that move to c_j at anytime, the expected number of users moving from c_i to c_j at time t is $N_{i\to j}(t) = N_i(t^-)m_{i\to j}$. The expected rate at which remote VI events occur in c_j caused by users in c_i moving to c_j is then:

$$vi_2(c_i^m, c_j^n) = \psi(c_i^m, c_j^n)\eta_2(c_i, c_j) \tag{6}$$

where

$$\begin{aligned}\eta_2(c_i, c_j) \quad = \quad & \lim_{\tau\to\infty} \tfrac{1}{\tau} \textstyle\int_0^\tau N_{i\to j}(t)R_j(t) \\ & \displaystyle\sum_{\mathrm{Cond2}} \rho_i(\mathrm{p})\rho_{\mathrm{p}\to\mathrm{p}'} I_j(\mathrm{p},\mathrm{p}')\beta_j\delta t\end{aligned}$$

and
Cond2 $= \mathrm{p} \in R1 \wedge \mathrm{p}' \in R2$, for case II-1
Cond2 $= \mathrm{p} \in R1 \wedge \mathrm{p}' \in c_j \setminus R2$, for case II-2
Cond2 $= \mathrm{p} \in c_i \setminus R1 \wedge \mathrm{p}' \in R2$, for case II-3
Cond2 $= \mathrm{p} \in c_i \setminus R1 \wedge \mathrm{p}' \in c_j \setminus R2$, for case II-4.

4.3 Type III: Teleportation

In this subsection, we consider the remote VI caused by users teleporting in/out of or joining/leaving the DVE at cell c_i (managed by server s_m), to observing users in cell c_j (managed by server s_n).

Teleportation is very popular in 3D games. In general, there are two types of teleportation. One is jumping between c_i and a distant cell and the other is moving through a predefined channel that connects two distant cells together. The jumping process can be divided into two parts: teleporting out of one cell and then into another cell. While the first part is similar to a user leaving the DVE, the second part is similar to a user joining the DVE. With channeling, we may consider the two cells connected by a channel as neighboring cells and handle them in the same way as moving to an adjacent cell (Section 4.2). As such, we only need to model leaving/joining of the DVE here.

For case III-1, let $N_{\overline{VE}\to i}(t) = N_i(t^-)m_{\overline{VE}\to i}$ be the expected number of users joining the DVE at c_i or jumping from a distant cell to c_i at t. The rate at which remote VI events occur in c_j caused by these users is:

$$vi_3(c_i^m, c_j^n) = \psi(c_i^m, c_j^n)\eta_3(c_i, c_j) \tag{7}$$

where

$$\eta_3(c_i, c_j) = \lim_{\tau\to\infty} \tfrac{1}{\tau} \textstyle\int_0^\tau N_{\overline{VE}\to i}(t) R_j(t) \sum_{\mathrm{p}\in R1} \rho_i(\mathrm{p}) J_j(\mathrm{p}) \beta_j \delta t$$

For case III-3, let $N_{i\to\overline{VE}}(t) = N_i(t^-)m_{i\to\overline{VE}}$ be the expected number of users leaving the DVE at c_i or jumping from c_i to a remote cell at t, the rate at which remote VI events occur in c_j caused by these users is:

$$vi_4(c_i^m, c_j^n) = \psi(c_i^m, c_j^n)\eta_4(c_i, c_j) \tag{8}$$

where

$$\eta_4(c_i, c_j) = \lim_{\tau\to\infty} \tfrac{1}{\tau} \textstyle\int_0^\tau N_{i\to\overline{VE}}(t) R_j(t) \sum_{\mathrm{p}\in R1} \rho_i(\mathrm{p}) J_j(\mathrm{p}) \beta_j \delta t$$

Since cases III-2 and III-4 concern with users joining or leaving c_i at the non-border region, they only incur local VI within c_i and are discussed in Section 4.4.

4.4 Local View Inconsistency

In this subsection, we consider the local VI caused by all three types of user movements, to observing users in virtual cell c_i.

As the message delay between two users in c_i is $2L_i^m$ and the number of users in c_i at time t is $R_i(t)$, the expected number of local VI events observed by users in c_i due to one user moving inside c_i (Type I) is:

$$Q^1_{LOC}(c_i^m) = R_i(t)2L_i^m \sum_{\mathrm{p}\in c_i \wedge \mathrm{p}'\in c_i} \rho_i(\mathrm{p})\rho_{\mathrm{p}\to\mathrm{p}'} I'_i(\mathrm{p}, \mathrm{p}')\beta_i$$

where $I'_i(\mathrm{p}, \mathrm{p}') = \sum_{(x,y)\in AOI1(\mathrm{p})\cup AOI1(\mathrm{p}')} \rho_i((x, y))$.

Let $ADJ(c_i, t)$ be the set of cells adjacent to c_i at time t. The expected number of local VI events observed by users in c_i at t due to one user in c_i moving to an adjacent cell $c_j \in ADJ(c_i, t)$ (Type II) is:

$$Q^2_{LOC}(c_i^m, c_j) = R_i(t)2L_i^m \sum_{\mathrm{p}\in c_i \wedge \mathrm{p}'\in c_j} \rho_i(\mathrm{p})\rho_{\mathrm{p}\to\mathrm{p}'} I'_i(\mathrm{p}, \mathrm{p}')\beta_i$$

The expected number of local VI events observed by users in c_i due to one user joining/leaving the DVE at c_i or teleporting in/out of c_i (Type III) is:

$$Q^3_{LOC}(c_i^m) = R_i(t)2L_i^m \sum_{\mathrm{p}\in c_i} \rho_i(\mathrm{p}) J'_i(\mathrm{p})\beta_i$$

where $J'_i(\mathrm{p}) = \sum_{(x,y)\in AOI1(\mathrm{p})} \rho_i((x, y))$.

At time t, the expected number of users moving inside c_i is $N_{i\to i}(t) = R_i(t) = N_i(t^-)m_{i\to i}$. The expected number of users joining/leaving the DVE at c_i and teleporting in/out of c_i is $N_{\leftrightarrow i}(t)$=$N_i(t^-)(m_{i\to\overline{VE}} + m_{\overline{VE}\to i} + \sum_{k=1,k\neq i\wedge k\notin ADJ(c_i,t)}^{N_C} m_{i\to k})$. Hence, the expected number of local VI events observed by users in c_i is obtained by combining the above three types of local VI events:

$$lvi(c_i^m) = 2L_i^m \eta_{LOC}(c_i) \tag{9}$$

where

$$\eta_{LOC}(c_i) = \lim_{\tau\to\infty} \tfrac{1}{\tau} \textstyle\int_0^\tau R_i(t)[N_{i\to i}(t) \sum_{\mathrm{p}\in c_i\wedge \mathrm{p}'\in c_i} \rho_i(\mathrm{p}) \rho_{\mathrm{p}\to\mathrm{p}'} I'_i(\mathrm{p}, \mathrm{p}') + \sum_{c_j\in ADJ(c_i,t)} N_{i\to j}(t) \sum_{\mathrm{p}\in c_i\wedge \mathrm{p}'\in c_j} \rho_i(\mathrm{p}) \rho_{\mathrm{p}\to\mathrm{p}'} I'_i(\mathrm{p}, \mathrm{p}') + N_{\leftrightarrow i}(t) \sum_{\mathrm{p}\in c_i} \rho_i(\mathrm{p}) J'_i(\mathrm{p})]\beta_i \delta t$$

4.5 Total View Inconsistency in the DVE

Based on the above analysis, we may now quantify the total amount of VI in the DVE.

As each cell is managed by only one server, we use a flag x_{im} to indicate if a cell c_i is managed by server s_m. Hence, if c_i is assigned to s_m, x_{im} = 1; otherwise, x_{im} = 0. The total amount of VI in the DVE can be computed by adding all the remote and local VI events together:

$$\sum_{\substack{i,j=1\\ i\neq j}}^{N_C} \sum_{\substack{m,n=1\\ m\neq n}}^{N_S} \sum_{k=1}^{4} x_{im}x_{jn}vi_k(c_i^m, c_j^n) + \sum_{i=1}^{N_C} x_{im} lvi(c_i^m) \tag{10}$$

where N_S is the number of servers in the DVE. The first term of Eq. (10) adds up all three types of remote VI, i.e., Eq. (5), (6), (7) and (8), while the second term represents the local VI, i.e., Eq. (9).

Since $\psi(c_i^m, c_j^n) = L_i^m + T_{m,n} + L_j^n$, Eq. (10) can be rewritten as:

$$\begin{aligned}
&\sum_{\substack{i,j=1\\ i\neq j}}^{N_C} \sum_{\substack{m,n=1\\ m\neq n}}^{N_S} \sum_{k=1}^{4} x_{im}x_{jn}T_{m,n}\eta_k(c_i, c_j)\\
&+ \sum_{\substack{i,j=1\\ i\neq j}}^{N_C} \sum_{\substack{m,n=1\\ m\neq n}}^{N_S} \sum_{k=1}^{4} x_{im}x_{jn}(L_i^m + L_j^n)\eta_k(c_i, c_j)\\
&+ \sum_{m=1}^{N_S} \sum_{i=1}^{N_C} x_{im} 2L_i^m \eta_{LOC}(c_i)
\end{aligned} \tag{11}$$

From Eq. (11), we can see that the first term depends on $T_{m,n}$, the server-server delays. Hence, if two cells, c_i and c_j, are both managed by server s_m, the VI between them due to $T_{m,n}$, i.e., $\sum_k T_{m,m}\eta_k(c_i^m, c_j^m)$, becomes zero. Thus, to reduce the VI caused by $T_{m,n}$, we need an algorithm to partition the virtual cells in such a way that those cells with large values of $\sum_k \eta_k(c_i^m, c_j^n)$ among themselves form a partition to be managed by the same server, while at the same time this algorithm should not violate the constraint of load balancing. For the second and third terms, which depend on the value of L_i^m, we need an algorithm to assign partitions to the appropriate servers so as to minimize the latency between users and their local servers.

5 VE Partitioning and Assignment

Given the view consistency model presented in Section 4, it may now be possible to partition the VE so as to minimize both the view inconsistency problem and load imbalance among the servers.

To convert the VI problem shown in Eq. (11) into a partitioning problem, we let P_m be the partition of virtual cells managed by server s_m, and N_S be the number of servers in the DVE. We need to find a partitioning strategy PA to partition all the cells of the VE into N_S disjoint subsets, P_1, P_2, ..., P_{N_S}, so that for any $1 \leq m, n \leq N_S, m \neq n$, it holds that $\mathrm{P}_m \cap \mathrm{P}_n = \phi$ and $\cup_{m=1}^{N_S}\mathrm{P}_m = \{c_i\}$. Next, each P_m is assigned to a different server s_m; this is a one-to-one assignment function that assigns all the cells in P_m to server s_m. An indicator X_{im} is used to denote if partition P_i is assigned to server s_m such that if P_i is assigned to server s_m, then X_{im} = 1; otherwise, X_{im} = 0. Eq. (11) can now be rewritten as:

$$\begin{aligned} min \quad & \sum_{\substack{i,j=1\\ i\neq j}}^{N_S} \sum_{\substack{m,n=1\\ m\neq n}}^{N_S} \sum_{\substack{c_k\in \mathrm{P}_i\\ c_l\in \mathrm{P}_j}} X_{im}X_{jn}T_{m,n}\sigma(c_k,c_l) \\ & + \sum_{\substack{i,j=1\\ i\neq j}}^{N_S} \sum_{\substack{m,n=1\\ m\neq n}}^{N_S} \sum_{\substack{c_k\in \mathrm{P}_i\\ c_l\in \mathrm{P}_j}} X_{im}X_{jn}(L_i^m + L_j^n)\sigma(c_k,c_l) \\ & + \sum_{m=1}^{N_S}\sum_{i=1}^{N_S}\sum_{c_k\in \mathrm{P}_i} X_{im}2L_k^m\eta_{LOC}(c_k) \qquad (12) \\ s.t. \quad & \textstyle\sum_{j=1}^{N_S}X_{ij} = 1, i\in\{1,\ldots,N_S\} \\ & \textstyle\sum_{i=1}^{N_S}X_{ij} = 1, j\in\{1,\ldots,N_S\} \\ & X_{ij}\in\{0,1\} \end{aligned}$$

where $\sigma(c_k,c_l) = \sum_{i=1}^{4}\eta_i(c_k,c_l)$.

Since directly finding an optimal solution to this stochastic problem is very difficult, in Eq. (12), we use the average rate at which VI events occur between each pair of adjacent cells (or inside one cell) to define the amount of remote/local VI. Thus, the original stochastic problem is abstracted into a deterministic model. As shown in Theorem 1 of the Supplementary, the partitioning problem of dividing the virtual cells $\{c_i\}$ among servers based on this deterministic model is NP-hard and there does not exist a polynomial time ϵ-approximation algorithm for this problem. Hence, we propose a two-step process to address this problem. First, we propose a greedy Knapsack-based partitioning (KP) algorithm to minimize the first term of Eq. (11) when dividing the cells into N_S partitions, while keeping the load deviation among the partitions within a specified threshold. This is discussed in Section 5.1. Second, we propose two assignment algorithms to minimize the second and the third terms of Eq. (11) when assigning the N_S partitions to the N_S servers. This is discussed in Section 5.2.

5.1 Dividing Virtual Cells into Partitions

To present our KP algorithm, we first formulate our partitioning problem into a joint optimization of remote VI and load balancing as follows:

$$\begin{aligned} min \quad & \sum_{m,n=1,m\neq n}^{N_S} \mu(\mathrm{P}_m,\mathrm{P}_n) \qquad (13) \\ s.t. \quad & \frac{Max_{1\le m\le N_S}\{w(\mathrm{P}_m)\} - \overline{w}}{\overline{w}} \le \theta \end{aligned}$$

where $\mu(\mathrm{P}_i,\mathrm{P}_j) = \sum_{c_k\in \mathrm{P}_i, c_l\in \mathrm{P}_j}\sigma(c_k,c_l)$ is used to quantitatively measure the amount of VI occurring in P_n caused by users in P_m. $w(\mathrm{P}_m)$ is the workload of P_m, and $\overline{w} = \frac{1}{N_S}\sum_{n=1}^{N_S} w(\mathrm{P}_m)$ is the average workload of all partitions. θ is a workload deviation threshold. Let W be the average server computation cost in handling a single user. The workload of c_i, denoted by $w(c_i)$, can be measured by the average number of users in c_i multiplied by W as follows:

$$\begin{aligned} w(c_i) = W\cdot\lim_{\tau\to\infty}\frac{1}{\tau}\int_0^\tau (N_i(t^-)(m_{i\to i} + m_{\overline{VE}\to i} - \\ m_{i\to\overline{VE}} - \sum_{j\neq i} m_{i\to j}) + \sum_{j\neq i} N_j(t^-)m_{j\to i})\delta t \quad (14) \end{aligned}$$

where c_j is a neighboring cell of c_i. The total workload of P_m, denoted by $w(\mathrm{P}_m)$, is then:

$$w(\mathrm{P}_m) = \sum_{c_i\in \mathrm{P}_m} w(c_i) \qquad (15)$$

We then construct a weighted undirected graph VIG=(V,E) of the VE, where V represents the set of $\{c_i\}$ in the VE, and E represents the set of edges such that for any $c_i, c_j\in V$, there is an edge $e_{ij}\in E$ between them. Thus, VIG is a complete graph. The weight of each node v_i, denoted by $w(v_i)$, is defined by the workload of c_i, that is, $w(v_i) = w(c_i)$. The weight of edge e_{ij}, denoted by $w(e_{ij})$, is defined by the amount of VI between c_i and c_j, i.e., $w(e_{ij}) = \sigma(c_i,c_j) + \sigma(c_j,c_i)$. With this VIG, Theorem 2 of the Supplementary shows that the virtual cell partitioning problem given in Eq. (13) is NP-hard. As such, we present a greedy partitioning algorithm, named KP, based on the idea of the *0-1 knapsack problem* [7].

In this KP algorithm, we start with the VIG graph. Given any $v_i\in V$ in VIG, we define the *worth* and the *weight* of another node v_j $(j\neq i)$ as:

1) wor_{ji}: The worth of v_j to v_i is defined as the weight of the edge between v_i and v_j, i.e., wor_{ji}=$w(e_{ij})$. Hence, for v_i, the worth of v_j is the total amount of VI between c_i and c_j. It holds that $wor_{ji} = wor_{ij}$.
2) wei_j: The weight of v_j is defined by its workload in integer value, i.e., wei_j=$\lceil w(v_j)\rceil$.

Next, for each node v_i in the current VIG, we use the 0-1 knapsack algorithm to find $SN(v_i)$, which contains a set of nodes that are most valuable to v_i under the constraint that the total workload of the nodes in $SN(v_i)$ cannot be higher than $\lceil(1+\theta)\overline{w} - w(c_i)\rceil$. For simplicity, we use $SN^+(v_i)$ to denote the set of nodes in $SN(v_i)$ plus v_i, i.e., $SN^+(v_i) = \{v_j | v_j = v_i \text{ or } v_j\in SN(v_i)\}$. Then, for the set of $SN^+(v_i)$ found, i.e., $\{SN^+(v_i)\}$, we choose the one currently holding the maximum worth, $SN^+(v_k)$. As $w(e_{lj}) = \sigma(v_l,v_j) + \sigma(v_j,v_l)$, v_k can be rewritten as:

$$v_k = argmax_{v_i\in V}(\sum_{v_l,v_j\in SN^+(v_i), l\neq j} w(e_{lj})) \qquad (16)$$

Then, $SN^+(v_k)$ forms a partition in our algorithm. We then remove from VIG all the nodes in $SN^+(v_k)$ and the edges incident to them. This algorithm repeats until the final partition is formed.

Our virtual cell partitioning algorithm is summarized in Fig. 2. It has two main steps. In the first step, at each iteration, the algorithm finds a partition that has the maximum amount of VI under the constraint of load

Fig. 2. Our Knapsack-based partitioning algorithm.

```
Input:   VIG=$(V, E)$ constructed from the $VE$;
Output: $\{\mathrm{P}_m\}$;

for $m$ = 1 to $N_S$
    $\mathrm{P}_m = \phi$;
for $m$ = 1 to $N_S$
    for every node $v_i \in V$
        for every node $v_j \in V(j \neq i)$
            $v_j$'s worth to $v_i$ is $wor_{ji} = w(e_{ij})$;
            $v_j$'s weight is $wei_j = \lceil w(v_j) \rceil$;
        endfor
        under the constraint $W = \lceil (1+\theta)\overline{w} - w(c_i) \rceil$,
        use the 0-1 Knapsack algorithm to find the
        set of nodes $SN(v_i)$ from the set $\{v_j\}$,
        by considering each $v_j$'s $wor_{ji}$ and $wei_j$;
    endfor
    Find $v_k$ from $\{SN^+(v_i)\}$ such that
        $v_k = argmax_{v_i \in V}(\sum_{v_l, v_j \in SN^+(v_i), l \neq j} w(e_{lj}))$;
    let $\mathrm{P}_m = SN^+(v_k)$ ;
    remove all the nodes in $\mathrm{P}_m$ and all incident
    edges to them from VIG;
endfor
while VIG $\neq \phi$
    randomly choose a node $v_i$ from $V$;
    find $\mathrm{P}_n$ of the maximum amount of VI with $v_i$, i.e.,
    $n = argmax_{1 \leq m \leq N_S}(\sum_{v_j \in \mathrm{P}_m}(w(e_{ij})))$;
    add $v_i$ to $\mathrm{P}_n$;
    remove $v_i$ and all incident edges from VIG;
endwhile
```

balancing, and then remove the nodes in this partition from the graph. In the second step, each node remaining in the graph is proportionally assigned among the partitions based on the amount of VI between this node and those nodes inside the partitions.

5.2 Assigning Partitions to Servers

After partitioning the virtual cells, we may now perform a one-to-one assignment of partitions to servers, with a main objective to minimize the client-server delays.

We first propose a matching algorithm for reducing the local VI in the partitions, i.e., the second and third terms in Eq. (12) caused by L_i^m. We construct a weighted complete bipartite graph $G_B = ((X_B, Y_B), E_B)$, where X_B represents the set of partitions, $\{\mathrm{P}_i\}$, Y_B represents the set of servers, $\{s_m\}$, and E_B represents the set of edges, such that for any $x_i \in X_B$, $y_m \in Y_B$, there is an edge $e_{im} \in E_B$ between them. The weight of each edge e_{im}, $w(e_{im})$, is defined by the amount of VI caused by the latency between users in P_i and s_m when P_i is assigned to s_m, due to the second and third items in Eq. (12):

$$\begin{aligned} w(e_{im}) \quad = \quad & \sum_{c_k \in \mathrm{P}_i} L_k^m (2\eta_{LOC}(c_k) \\ & + \sum_{j=1, j\neq i}^{N_S} \sum_{c_l \in \mathrm{P}_j} (\sigma(c_k, c_l) + \sigma(c_l, c_k))) \end{aligned} \tag{17}$$

Our assignment problem is now transformed into finding a *minimum weighted perfect matching* in this bipartite graph G_B, so as to minimize the total amount of VI. Hence, Eq. (17) can now be rewritten as:

$$\begin{aligned} min \quad & \sum_{i=1}^{N_S} \sum_{m=1}^{N_S} X_{im} w(e_{im}) \\ s.t. \quad & \textstyle\sum_{j=1}^{N_S} X_{ij} = 1, i \in \{1, \ldots, N_S\} \\ & \textstyle\sum_{i=1}^{N_S} X_{ij} = 1, j \in \{1, \ldots, N_S\} \\ & X_{ij} \in \{0, 1\} \end{aligned} \tag{18}$$

Here, we may apply the Kuhn-Munkres algorithm [16] [26] to find the minimum weighted perfect matching. We first transform it into a *maximum weighted perfect matching problem*. Let MB be the matrix to denote the weights of E_B in G_B, i.e., the value of each element: $mb_{ij} = w(e_{ij})$. Let mb^* be the element having the largest value in MB, i.e., $mb^* = max_{1 \leq i,j \leq N_S} mb_{ij}$. Let MB' be another matrix, with the value of each element: $mb'_{ij} = mb^* - mb_{ij}$.

Next, we construct another bipartite graph, $G'_B = ((X'_B, Y'_B), E'_B)$, which holds $X'_B = X_B$, $Y'_B = Y_B$, $E'_B = E_B$, and the weight of each edge $w(e'_{ij})$ in E'_B is set to be $w(e'_{ij}) = mb'_{ij}$. Thus, the maximum weighted perfect matching MA^*, which can be found in G'_B, is exactly the minimum weighted perfect matching MA in G_B [16] [26]. In the perfect matching MA returned by the Kuhn-Munkres algorithm, for each $e_{xy} \in MA$ ($x \in X'_B$, $y \in Y'_B$), the corresponding partition P_x is then assigned to the server s_y.

Based on the results generated by the matching algorithm, we further present a greedy algorithm for reducing the remote and local VI's of the DVE. Let $VI_T|[\mathrm{P}_i^m, \mathrm{P}_j^n]$ be the total amount of VI in the DVE under current assignment scenario, where P_i and P_j are assigned to s_m and s_n, respectively. Now, we attempt to change this assignment by assigning P_i to s_n and P_j to s_m. Let $VI_T|[\mathrm{P}_i^n, \mathrm{P}_j^m]$ be the total amount of VI in the DVE under this new assignment scenario. If $VI_T|[\mathrm{P}_i^n, \mathrm{P}_j^m] < VI_T|[\mathrm{P}_i^m, \mathrm{P}_j^n]$, we should accept this new assignment. Hence, our greedy algorithm is shown in Fig. 3. During each iteration, we choose the new assignment that maximizes the reduction of VI.

6 Results and Discussions

To study the performance of the proposed method, we have implemented four other methods for comparison: UA [34] and IPA [21], which employs optimization techniques for assigning users directly to servers, and PA [23] and GRASP [25], which employs optimization techniques for partitioning users into regions to be served by different servers. All five methods are implemented in C++. The testing platform is a PC with an i5 2.8GHz CPU and 4GB RAM. We have set up a game scene of $5 \times 5\text{km}^2$ in size and equally divided it into 32×32 virtual cells to be managed by 64 servers. We have created two groups of users, each of 800 users. One group simply

Fig. 3. A greedy algorithm to reduce total amount of VI.

```
Input:  The assignment from the matching algorithm;
Output: The refined assignment;

Compute the total amount of VI, $cur\_VI$, by Eq. (12);
$diff\_VI$=$\infty$;
$d^*$ is the threshold for accepting the new assignment;
while ($diff\_VI > d^*$)
    $reduced\_VI$ = 0;
    for i = 1 to $N_S - 1$
        for j = i+1 to $N_S$
            calculate $VI_T|[\mathrm{P}_i^n, \mathrm{P}_j^m]$, due to the assignment
                P$_i$ to $s_n$ and P$_j$ to $s_m$;
            if ($cur\_VI - VI_T|[\mathrm{P}_i^n, \mathrm{P}_j^m] > reduced\_VI$)
                $reduced\_VI = cur\_VI - VI_T|[\mathrm{P}_i^n, \mathrm{P}_j^m]$;
                P1=P$_i$; P2=P$_j$; s1=$s_m$; s2=$s_n$;
            endif
        endfor
    endfor
    assign P1 to s2 and P2 to s1;
    $diff\_VI = reduced\_VI$;
    $cur\_VI = cur\_VI - diff\_VI$;
endwhile
```

moves around the VE randomly to model the statistical behavior of typical users, and the other moves circularly around of the perimeter of the VE to model large crowd movement in order to test how the five methods respond in such a demanding situation. All users move at speeds randomly changing between 0 and 5m per frame, with an average speed of 2.5m per frame. We set the frame rate to 10 frames per second, and execute the five methods once a second, i.e., once every 10 frames. Based on this movement information, we can determine the values of most of the statistical parameters. In addition, we set 1-2% of the users dynamically joining or leaving the DVE each second, but the total number of users is kept around 1,600. For simplicity, we assume that all users generate the same amount of load and each server can handle a maximum of 30 users.

In the following subsections, we discuss three sets of experiments to study the performance of our method.

6.1 Experiments on View Inconsistency

To evaluate the effectiveness of our method in reducing the view inconsistency problem, we vary three parameters, L_i^m (client-server latency), $T_{m,n}$ (server-server latency), and β (rate of users generating queries), and measure the total number of VI events observed by the users for every 10-frames, i.e., between two consecutive load balancing processes.

TABLE 1
Effect of $T_{m,n}$ on average number of VI events observed.

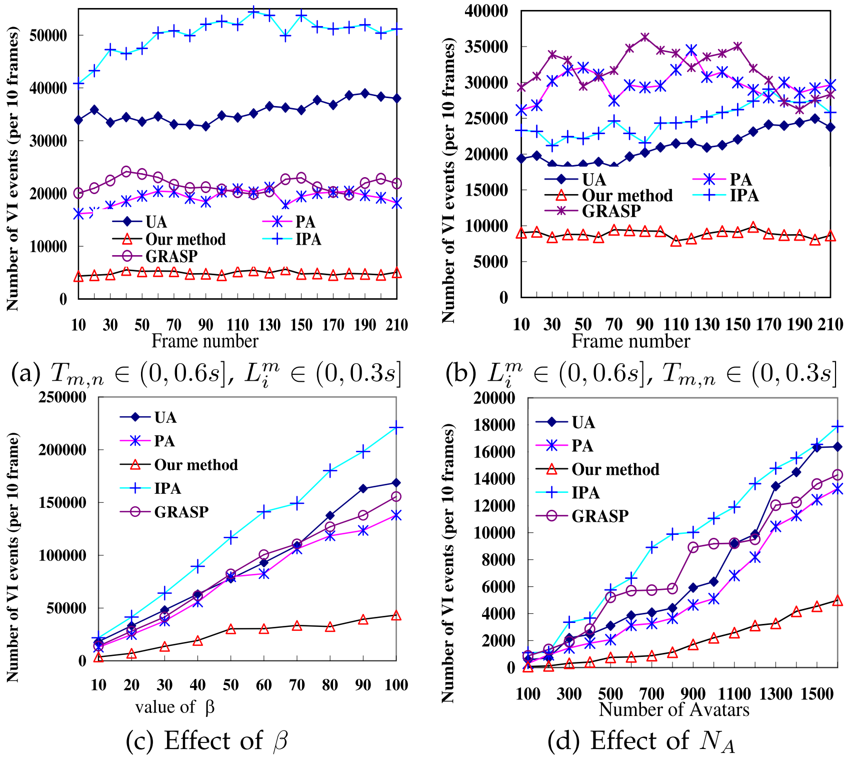
$T_{m,n}$	**(0,0.2s]**	**(0,0.4s]**	**(0,0.6s]**
Our Method	4445	4850	4926
PA	15145	16992	19226
UA	12985	22574	35532
GRASP	16656	20181	21569
IPA	16293	28958	50131

TABLE 2
Effect of L_i^m on average number of VI events observed.

L_i^m	**(0,0.2s]**	**(0,0.4s]**	**(0,0.6s]**
Our Method	3334	5798	8870
PA	11121	20680	29830
UA	19778	21111	23750
GRASP	12819	25973	31666
IPA	22429	23630	24709

The first experiment studies the impact of $T_{m,n}$ on the number of VI events observed by the users. Each L_i^m is randomly selected from the range of (0, 0.3s]. β is set to 10, i.e., once per frame. $T_{m,n}$ is also randomly selected from a range but we vary the range from low to high: (0, 0.2s], (0, 0.4s] and (0, 0.6s]. (According to our earlier study [5], round-trip network delays vary from $5ms$ for LAN to $325ms$ between HK and UK. These delays also vary from day to night. The three ranges are to cover a wider network latency conditions.) Table 1 shows the average numbers of VI events observed by the users. Fig. 4(a) shows the situation when $T_{m,n}$ is randomly selected from the range of (0,0.6s]. The second experiment studies the impact of L_i^m on the number of VI events observed by the users. Each $T_{m,n}$ is randomly selected from the range of (0, 0.3s]. β is set to 10. L_i^m is also randomly selected from a range but we vary the range from low to high: (0, 0.2s], (0, 0.4s] and (0, 0.6s]. Table 2 shows the average numbers of VI events observed by the users. Fig. 4(b) shows the situation when L_i^m is randomly selected from the range of (0,0.6s]. The third experiment studies the impact of β on the number of VI events observed by the users. Both $T_{m,n}$ and L_i^m are randomly selected from the range of (0, 0.3s], but we vary β from 10 to 100, as shown in Fig. 4(c). The fourth experiment studies the impact of N_A on the number of VI events observed by the users. Both $T_{m,n}$ and L_i^m are randomly selected from the range of (0, 0.3s], β is set to 10, and we increase N_A from 100 to 1600 as shown in Fig. 4(d).

From Tables 1 and 2, we can see that for all five methods, the average number of VI events observed by the users increases as the range of $T_{m,n}$ or L_i^m increases. We can also see that our method has much lower numbers than all the other methods. This means that our method can significantly reduce the number of VI events. The reason is that although PA and GRASP try to minimize the inter-server communication costs while UA tries to minimize the client to client latency, which help reduce VI in some way, their analyses are based on users' static positions or the latency at every time moment. As we have shown in Section 4, when a user changes its state, the amount of VI events caused depends on the initial and the final positions of the movement in each frame. While PA, GRASP and UA do not consider this, our method is designed to address this VI problem through the proposed joint optimization



(a) $T_{m,n} \in (0, 0.6s]$, $L_i^m \in (0, 0.3s]$ (b) $L_i^m \in (0, 0.6s]$, $T_{m,n} \in (0, 0.3s]$

(c) Effect of β (d) Effect of N_A

Fig. 4. Experiments on the total number of VI events observed by users.

framework. As a result, our method has a much lower number of VI events observed. Although IPA is designed to address the inconsistency problem, it needs to know in advance the user position over time in order to reduce time-space inconsistency among all the clients. However, as users' motion behaviors are often unpredictable, IPA can only help reduce the communication delay among the users and users close to each other in the VE may be assigned to different servers. This explains why IPA does not perform too well in reducing VI. For a similar reason, we can see from Fig. 4(c) and 4(d) that although the numbers of VI events for all methods increase as we increase the rate of user generating queries and the total number of users, our method increases much more gently than the other four methods.

It is interesting to note that with our method, the increase in the number of VI events observed due to the increase in the range of $T_{m,n}$ (Table 1) is much smaller than that due to the increase in the range of L_i^m (Table 2). This indicates that our method is able to cluster interacting users to the same partitions to minimize the need to send update messages among the servers. As a result, increasing the range of $T_{m,n}$ only slightly increases the number of VI events. However, increasing the range of L_i^m affects every user and thus has a more significant effect on the number of VI events. On the contrary, with UA and IPA, the increase in the number of VI events observed due to the increase in the range of $T_{m,n}$ is much higher than that due to the increase in the range of L_i^m. The reason is that since UA assigns a new user to a server that will give the lowest total latency between this new user and each of the other users already assigned, it puts a higher weight on a server with a lower latency to the new user than a server with a lower latency to the other servers managing the relevant users. In a similar way, IPA assigns a new user to a server in order to minimize the total latency between the new user and all other users already-assigned as well as not-yet-assigned. As a result, UA and IPA can address L_i^m better than $T_{m,n}$.

6.2 Experiment on Computation Time

To study the efficiency of our method, we investigate the impact of the number of users, N_A, on the computation time in this experiment. We vary N_A in the set of {1200, 1600, 2000}, with radius of the AOI, $r = 25m$. Again, half of the users move randomly and the other half move circularly around the VE. Table 3 shows the computation times of the five methods. We can see that UA and our method are much more efficient than the other methods, with UA being the most efficient one and PA being the most inefficient one. As we increase the number of users, the computation times of IPA, GRASP, UA and our method also increase, while PA only increases slightly. We also note that as the number of users increases from 1,200 to 2,000, the changes in computation time of UA (+160%), GRASP (+385%), and IPA (+316%) are much higher than that of our method (+78%).

To explain the results shown in Table 3, we need to analyze the complexity of the five methods:

- Our method consists of three main algorithms: knapsack-based partitioning (KP), Kuhn-Munkres-

based (KM) and greedy assignment (GA). The dominant process is the KP algorithm, with a complexity of $O(N_A \times N_C^2)$, where N_C is the number of cells.
- PA also consists of three main algorithms: recursive bisection partitioning (RBP), layering partitioning (LP) and communication refinement partitioning (CRP). The complexity of RBP is $O(|N_C|^3 \times (N_S - 1))$, where N_S is the total number of servers. The complexity of both LP and CRP is $O(N_S^6)$.
- UA consists of two main algorithms, a greedy algorithm (GA) and a distributed greedy algorithm (DGA). The dominant process is the GA algorithm, with a complexity of $O(N_A \times N_S \times (N_S + N_A))$.
- GRASP has three main steps. Its complexity is dominated by the third step, which involves assigning the non-assigned avatars and reassigning avatars inside their AOIs to servers according to the total intra-server workload and inter-server communication cost. It has a complexity of $O(N_{NA} \times N_A^2 \times N_S \times r^2)$, where N_{NA} is the number of non-assigned avatars after the initial partitioning step.
- IPA has two main steps. Its complexity is dominated by the second step, which assigns each user to servers so as to minimize the total amount of time-space inconsistency between the user and all other users of the DVE. It has a complexity of $O(N_A^2 \times N_S^2)$.

From this complexity analysis, we can see that PA, GRASP and IPA have much higher complexities than UA and our method. This explains their higher computation times as shown in Table 3. In addition, as the complexity of PA is independent of N_A, its computation time only changes very little as we increase the number of users. On the other hand, the complexity of UA, GRASP and IPA quadratically increases with N_A, while our method linearly increases with N_A. This explains why the changes in computation time of UA, GRASP and IPA are much higher than that of our method. From this experiment, we may conclude that UA and our method are much more efficient than PA, GRASP and IPA, and are more suitable for DVE applications in practice.

6.3 Experiment on Load Balancing

To study the load balancing performance, we set $\beta = 10$ and $r = 25m$, and measure the number of overloaded servers during each 10-frame period, i.e., between two consecutive load balancing processes. Since a higher number of overloaded servers means that more users suffer from higher response time, we would prefer a method with a lower number of overloaded servers.

TABLE 3
Computation times.

No. of Users	1,200	1,600	2,000
Our Method	0.703s	0.936s	1.251s
PA	168.06s	175.06s	181.40s
UA	0.317s	0.562s	0.826s
GRASP	5.522s	11.29s	21.27s
IPA	35.79s	68.26s	113.24s

Table 4 compares the average numbers of overloaded servers for the five methods over a simulation run. Recall from Table 3 that PA, GRASP and IPA take much longer than $1s$ to run each load balancing process. In order to focus our comparison on the load balancing performance, we have neglected this time constraint and allow extra time for these methods to complete their load balancing processes during the experiment. We can see from Table 4 that the numbers of overloaded servers for all five methods are very small, meaning that all methods can effectively prevent servers from getting overloaded, with our method having the best performance.

TABLE 4
Average number of overloaded servers.

Our Method	PA	UA	GRASP	IPA
0.24	0.33	0.28	0.71	0.61

6.4 Overall Evaluation

In this paper, we first point out that view consistency and load balancing are both important, but also conflicting, objectives of DVEs. From our experimental results shown in Section 6.1, we may summarize that our method produces a much lower number of view inconsistent events than all the other four methods that we compare with. This is mainly because PA, GRASP, and UA do not consider this issue in their designs, while IPA needs to know in advance the users' movement behaviors in order to be effective. Minimizing the number of view inconsistent events is important as it helps improve the overall interactivity of the DVE and reduces the chance of user arguments. Our results in Section 6.3 also show that our method has a lower average number of overloaded servers than the other methods. This means that our method is able to improve both the view inconsistency problem as well as the load balancing performance. Although our method has a slightly higher computation time than UA as shown in Section 6.2, it is efficiently enough for DVE applications.

7 Conclusion

Due to network latency and user movements, users of DVEs often suffer from the view inconsistency problem. How to minimize the view inconsistent problem is important to the popularity of DVEs, such as online games. In this paper, we have analyzed different types of user movement and estimated the amount of view inconsistency events that they may cause. Based on this analysis, we have proposed an efficient joint optimization framework to reduce the view inconsistency problem under the constraint of load balancing. We first construct a view-inconsistency graph (VIG) for the VE, and present a partitioning algorithm to divide the cells into partitions so as to minimize the amount of view inconsistency while at the same time, keeping the workload of each partition below a given threshold. We then propose another algorithm to assign the partitions to different servers by considering the resulting amount of view inconsistency in each partition due to network delays. Our experimental results show that the proposed method performs better than existing methods, in terms of view consistency and load balancing. At the same time, the proposed method is shown to be efficient enough for DVEs.

As a future work, we are currently working an incremental version of the proposed joint optimization framework. If it is successful, we expect the computational cost of the method to be further reduced. Another interesting work is to extend the proposed method to heterogeneous servers. This will require modifying both the partitioning and the assignment algorithms.

8 Acknowledgements

We would like to thank the three reviewers for their insightful and constructive comments/suggestions on this paper. The work described in this paper was partially supported by the National 973 Programs (Grant No. 2013CB329102), four NSFC grants (Grant No. 60903053, 61100194, 61300117, 61272188), the Natural Science Foundation of Zhejiang Province (Grant No. LY12F02005), the Open Foundation of State Key Laboratory of Networking and Switching Technology in Beijing University of Posts and Telecommunications (Grant No. SKLNST-2013-1-14), the Open Foundation of State Key Laboratory for Novel Software Technology of Nanjing University (Grant No. KFKT2014B15), and two GRF grants from the RGC of Hong Kong (RGC Reference Numbers: CityU 116010 and CityU 115112).

References

[1] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, ”A View of Cloud Computing,” *Communications of the ACM*, **53**(4):50-58, April 2010.

[2] B. van Den Bossche, B. de Vleeschauwer, T. Verdickt, F. de Turck, B. Dhoedt, and P. Demeester, ”Autonomic microcell assignment in massively distributed online virtual environments,” *Journal of Network and Computer Applications*, Elsevier, **32**(6):1242-1256, 2009.

[3] R. Bryant, ”Simulation of Packet Communication Architecture Computer Systems,” *Technical Report*, Computer Science Laboratory, MIT, 1977.

[4] A. Chan, R. Lau, and B. Ng, ”Motion Prediction for Caching and Prefetching in Mouse-Driven DVE Navigation,” *ACM Trans. on Internet Technology*, **5**(1):70-91, Feb. 2005.

[5] A. Chan, R. Lau, and L. Li, ”Hand Motion Prediction in Distributed Virtual Environments,” *IEEE Trans. on Visualization and Computer Graphics*, **14**(1):146-159, Jan. 2008.

[6] J. Chen, B. Wu, M. Delap, B. Knutsson, H. Lu, and C. Amza, ”Locality Aware Dynamic Load Management for Massively Multiplayer Games,” In *Proc. ACM SIGPLAN Symp. on PPoPP*, pp. 289-300, 2005.

[7] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd Edition, McGraw-Hill, 2003.

[8] Y. Deng and R. Lau, ”Heat Diffusion Based Dynamic Load Balancing for Distributed Virtual Environments,” In *Proc. ACM VRST*, pp. 203-210, 2010.

[9] C. Diot and L. Guatier, ”A Distributed Architecture for Multiplayer Interactive Applications on the Internet,” *IEEE Network*, **13**(4):6-15, 1999.

[10] R. Fujimoto, *Parallel and Distribution Simulation Systems*, John Wiley and Sons, 1999.

[11] E. Frecon, ”DIVE: A Scalable Network Architecture for Distributed Virtual Environment,” *Distributed Systems Engineering Journal*, **5**(3):91-100, 1998.

[12] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, 1979.

[13] D. Jefferson, ”Virtual time,” *ACM Trans. on Programming Languages and Systems*, **7**(3):404-425, 1985.

[14] X. Ji and E. John. ”Branch-and-Price-and-Cut on the clique Partitioning Problem with Minimum Clique Size Requirement,” *Proc. IMA Special Workshop: Mixed-Integer Programming*, 2005.

[15] T. Koopmans and M. Beckmann, ”Assignment Problems and the Location of Economic Activities,” *Econometrica*, **25**(1):53-76, 1957.

[16] H. Kuhn, ”The Hungarian Method for the Assignment Problem,” *Naval Research Logistics Quarterly*”, **2**:83-97, 1955.

[17] K. Lee and D. Lee, ”A Scalable Dynamic Load Distribution Scheme for Multi-server Distributed Virtual Environment systems with highly-skewed user distribution,” In *Proc. ACM VRST*, pp. 160-168, 2003.

[18] F. Li, L. Li, and R. Lau, ”Supporting Continuous Consistency in Multiplayer Online Games,”In *Proc. ACM Multimedia*, pp. 388-391, 2004.

[19] F. Li, F. Li, and R. Lau, ”A Trajectory-Preserving Synchronization Method for Collaborative Visualization,” *IEEE Trans. on Visualization and Computer Graphics*, **12**(5):989-996, Sept. 2006.

[20] Y. Li and W. Cai, ”Consistency Aware Update Schedule in Multi-Server Distributed Virtual Environments,” In *Proc. Int'l Workshop on SIMUTOOLS*, 2010.

[21] Y. Li and W. Cai, ”Consistency-aware Partitioning Algorithm in Multi-server Distributed Virtual Environments,” In *Proc. IEEE IPDPS*, pp. 798-807, 2012.

[22] J. Lui, ”Constructing Communication Subgraphs and Deriving an Optimal Synchronization Interval for Distributed Virtual Environment Systems,” *IEEE Trans. on Knowledge and Data Engineering*, **13**(5):778-792, 2001.

[23] J. Lui and M. Chan.” An Efficient Partitioning Algorithm for Distributed Virtual Environment Systems,” *IEEE Trans. on Parallel and Distributed Systems*, **13**(3):193-211, 2002.

[24] M. Mauve, ”Consistency in Replicated Continuous Interactive Media,” *Proc. ACM CSCW*, pp. 181-190. 2000.

[25] P. Morillo, S. Rueda, J. Orduna, and J. Duato, ”A Latency-Aware Partitioning Method for Distributed Virtual Environment Systems,” *IEEE Trans. on Parallel and Distributed Systems*, **18**(9):1215-1226, 2007.

[26] J. Munkres, ”Algorithms for the Assignment and Transportation Problems,” *Journal of the SIAM*, **5**(1):32-38, 1957.

[27] B. Ng, A. Si, R. Lau, and F. Li, ”A Multi-Server Architecture for Distributed Virtual Walkthrough,” In *Proc. ACM VRST*, pp. 163-170, 2002.

[28] B. Ng, R. Lau, A. Si, and F. Li, ”Multi-Server Support for Large Scale Distributed Virtual Environments,” *IEEE Trans. on Multimedia*, **76**:1054-1065, 2005.

[29] S. Sahni and T. Gonzalez. ”P-Complete Approximation Problems,” *Journal of the ACM*, **23**(3):555-565, 1976.

[30] S. Singhal and M. Zyda, *Networked Virtual Environments*, ACM Press, 1999.

[31] X. Tang and S. Zhou, ”Update Scheduling for Improving Consistency in Distributed Virtual Environments,” *IEEE Trans. on Parallel and Distributed Systems*, **21**(6):765-777, 2010.

[32] B. de Vleeschauwer, B. van Den Bossche, T. Verdickt, F. de Turck, B. Dhoedt, and P. Demeester, ”Dynamic Microcell Assignment for Massively Multiplayer Online Gaming,” In *Proc. ACM SIGCOMM Workshop on Network and System Support for Games*, pp. 1-7, 2005.

[33] D. West, ”Optimizing Time Warp: Lazy Rollback and Lazy Reevaluation,” *Technical Report*, Computer Science Department, University of Calgary, 1988.

[34] L. Zhang and X. Tang, ”Client Assignment for Improving Interactivity in Distributed Interactive Applications,” *Proc. IEEE INFOCOM*, pp.3227-3235, 2011.

[35] S. Zhou, W. Cai, B. Lee, and S. Turner, ”Time-Space Consistency in Large-Scale Distributed Virtual Environments,” *ACM Trans. Modeling and Computer Simulation*, **14**(1):31-47, 2004.

Haiyang Hu received the Ph.D., M.S. and B.S. degrees in Computer Science from Nanjing University, China in 2006, 2003 and 2000, respectively. Currently, he is a professor of the Hangzhou Dianzi University, China. His research interests include mobile computing and distributed computing. His research results have been published in more than 20 papers in international journals and conference proceedings.

Rynson W.H. Lau received his Ph.D. degree from University of Cambridge. He was on the faculty of Durham University and Hong Kong Polytechnic University. He is now with City University of Hong Kong.

Rynson serves on the Editorial Board of Computer Animation and Virtual Worlds, Int'l Journal of Virtual Reality, and IEEE Trans. on Learning Technologies. He has served as the Guest Editor of a number of journal special issues, including *ACM Trans. on Internet Technology*, *IEEE Trans. on Multimedia*, *IEEE Trans. on Vis. and Computer Graphics*, and *IEEE Computer Graphics & Applications*. In addition, he has also served in the committee of a number of conferences, including Program Co-chair of *ACM VRST 2004*, *ICEC 2007*, *ACM MTDL 2009*, *IEEE U-Media 2010*, and Conference Co-chair of *CASA 2005*, *ACM VRST 2005*, *ICWL 2007*, *ACM MDI 2009*, *ACM VRST 2010*. His research interests are distributed virtual environments and multimedia systems.

Hua Hu is a full professor of Hangzhou Dianzi University, China. He received the Ph.D., M.S. and B.S. degree in Computer Science from Zhejiang University, China in 1998, 1992 and 1989, respectively. His research interests mainly include parallel computing and distributed system and pervasive computing. His research results have been published in more than 50 papers in international journals and conference proceedings.

Benjamin Wah is currently the Provost of The Chinese University of Hong Kong and the Wei Lun Professor of Computer Science and Engineering. He is also Professor Emeritus of The University of Illinois, Urbana-Champaign. He received his Ph.D. degree from UC Berkeley. His current research interests are in the areas of nonlinear optimization, multimedia signal processing, and computer networks.

He cofounded the IEEE Trans. on Knowledge and Data Engineering in 1988 and served as its Editor-in-Chief between 1993-1996. He received the IEEE-CS Technical Achievement Award in 1998, the IEEE Millennium Medal in 2000, the Raymond T. Yeh Lifetime Achievement Award from the Society for Design and Process Science in 2003, the IEEE Computer Society W. Wallace-McDowell Award in 2006, and the IEEE-CS Richard E. Merwin Award and IEEE-CS Technical Committee on Distributed Processing Outstanding Achievement Award both in 2007. He has served the IEEE Computer Society in various capacities, including Vice President for Publications (1998 and 1999) and President (2001). He is a Fellow of the IEEE, the ACM and the AAAS.