Fairness-aware Update Schedules for Improving Consistency in Multi-server Distributed Virtual Environments

Li Yusen*† s080007@e.ntu.edu.sg
Yunhua Deng† yhdeng@ntu.edu.sg
Wentong Cai† aswtcai@ntu.edu.sg
Xueyan Tang† asxytang@ntu.edu.sg

*Department of Computer Science and Information Security
Nankai University
Tianjin, China

†School of Computer Science and Engineering
Nanyang Technological University
Singapore

ABSTRACT

In Distributed Virtual Environments (DVEs), how to guarantee consistency and fairness is a primary concern for improving user experience. As the scale of DVE systems grows fast, multi-server architecture has been widely used in large scale DVEs such as Massively Multiplayer Online Games (MMOGs). In multi-server DVEs (MSDVEs), servers may get saturated with resources such as network bandwidth due to huge resource demand and workload variability. With resource limitations, state updates cannot be disseminated timely and inconsistency and unfairness will be greatly increased if the resources are not allocated properly. In this paper, we propose a fairness-aware state update scheduling algorithm which can minimize total inconsistency while guarantee fairness in MSDVEs under servers' network bandwidth constraints. Experimental results show that the proposed algorithm significantly improves the consistency and fairness compared to other existing update algorithms.

CCS Concepts

•**Computing methodologies** → **Distributed simulation;** *Distributed algorithms;* •**Computer systems organization** → Client-server architectures;

1. INTRODUCTION

Distributed Virtual Environment (DVE), a virtual world deployed on a group of computers connected via network, allows multiple geographically distributed participants, also known as clients, to communicate and interact with each other through the shared virtual world. In the virtual world, each client is often represented by an entity, called an avatar.

SIMUTOOLS 2016, August 22-23, Prague, Czech Republic

tual world and communicate and interact with other clients. DVEs have been widely used in many different applications such as military training, e-learning and online games [1, 4, 17, 21].

The client-server architecture is the most popular one to support DVEs. In client-server DVEs, the virtual world is maintained by a server and each client maintains a copy of the (relevant) virtual world state on his computer. When one client performs an action that affects the virtual world (e.g., the client controlled avatar moves to a new position), the state of the virtual world maintained by other clients must be updated in a timely manner. However, due to transmission delay and clock asynchrony, different clients may receive the same state update at different times, which will result in inconsistency and unfairness. Inconsistency refers to the differences between the virtual worlds seen by clients and the virtual world maintained by the server. Unfairness refers to the differences among the virtual worlds seen by different clients. It has been shown that both inconsistency and unfairness are crucial for user experience in DVEs [2]. Inconsistency may deteriorate the game responsiveness and playability while unfairness may provide unfair advantages or disadvantages to different players. Therefore, how to reduce inconsistency and unfairness is very important for DVEs.

As the development of the DVE applications, the scale of DVE systems becomes larger and larger. Like the most popular online game WoW [21], the peak number of simultaneous active users has exceeded 5 million [22]. Multi-server architecture, especially the zone based multi-server architecture, has become a popular platform to support large scale DVEs. In zone based multi-server DVEs (MSDVE), the virtual world is usually spatially partitioned into several disjoint zones, with each zone managed by only one server. A client interacts only with other clients in the same zone, and may move to other zones. As a server only needs to handle one or more zones instead of the entire virtual world, the system is more scalable. The zone based architecture has been used in many large-scale virtual environments such as RING [7], CittaTron [8] and many existing Massively Multiplayer Online Games (MMOGs) [6, 14].

At present, a large scale MSDVE like MMOGs can include millions of concurrent users spread across the world,

which demands huge amount of resources such as computing power and network bandwidth on various servers [22]. Adding clients to a MSDVE will increase the resource requirement of servers. As the number of clients increases, the total resource demand may become huge. Moreover, a DVE system is highly dynamic. A client can join or leave a DVE at any given time. Therefore, the number of participating clients is dynamically changing over time, which results in dynamic workload requirement at servers. In addition, the workload like network bandwidth in a DVE is also highly dependent on avatars' interactions [18]. When avatars are crowded in the virtual world, more network bandwidth for each client is required. For instance, the peak value of the network bandwidth requirement from server to a client in WoW can exceed 64kbps, which is much larger than the median value 6.9kbps [18]. Facing huge resource demand and workload variability, servers may get saturated with resources like network bandwidth. If resources are not allocated properly, inconsistency and unfairness will be greatly increased.

In our previous work [9, 11], we have studied how to schedule state updates for minimizing inconsistency in MSDVEs with network bandwidth constraints of servers. However, unfairness is not considered in the proposed update algorithm. In this paper, we take unfairness into account and reconsider the state update scheduling problem in MSDVEs. The aim is to minimize inconsistency while guarantee fairness in MSDVEs when the network bandwidth is constrained for servers. The contributions of this paper are three-fold. First, we propose a new metric using time-space inconsistency to measure unfairness in a DVE. The new metric takes both spatial and temporal differences into account in the measurement, which is more effective in reflecting the impact of unfairness on users' experiences. Second, we propose a fairness-aware update scheme which ensures the same update can be issued at the same time by different clients so that the unfairness among clients is eliminated. Third, we propose a heuristic update algorithm that can minimize inconsistency with fairness guarantees in a MSDVE with network bandwidth constraints of servers. The proposed update algorithm has been shown to outperform other algorithms by experimental results.

The rest of the paper is structured as follows. The system model and problem definition are introduced in Section 2. In Section 3, the proposed update schedules are presented. The experimental evaluations are presented in Section 4. The related work is summarized in Section 5. In the last section, a conclusion is given and the future work is discussed.

2. SYSTEM MODEL

In this paper, we focus on zone based MSDVEs. However, the proposed approach can be easily applied to other types of MSDVEs. In zone based MSDVEs[1], as shown in Figure 1, the virtual world is assumed to be partitioned into zones and each zone is maintained by one server. For instance, in Figure 1, the virtual world is partitioned into three zones z_1, z_2 and z_3 which are maintained by servers s_1, s_2 and s_3 respectively. Avatars located in a zone are maintained by the server that is in charge of this zone. Generally, an avatar is only allowed to interact with the avatars in the same zone

[1]In the rest of the paper, MSDVE refers to the zone-based MSDVE

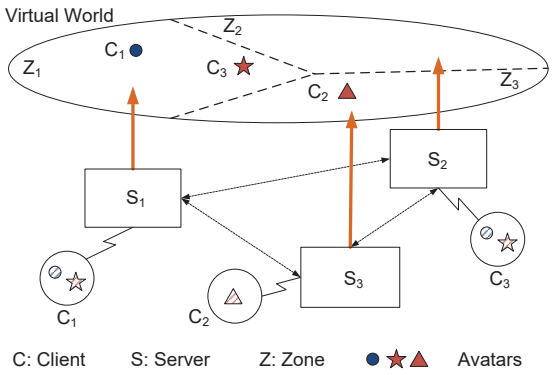

Figure 1: System Model of a MSDVE

where the avatar is residing.

Traditionally, clients can be assigned to servers in two different ways: based on virtual position or based on physical position. In the virtual position manner, each client is connected to the server that maintains the zone where his avatar is. The server-to-server communication is avoided in this case. However, the network delay between a client and the server may be large. In the physical position manner, each client is connected to the closest server that incurs minimum network delay. In this case, the network delay between a client and its connected server is small. However, if the connected server is not the server that maintains his avatar, the state updates need to be forwarded by the connected servers, thus the communications between servers are required. Therefore, client assignment will affect the transmission delays from servers to clients, and cause inconsistency. In our previous work [10, 12], the client assignment problem has been studied and a new client assignment approach was proposed. In that approach, a client is allowed to connect to any server, which is not necessarily the server that maintains his avatar or the closest server. The clients are assigned to servers with the purpose of minimizing inconsistency. In our system model, the client assignment will be determined by the approach proposed in [12].

In this paper, we focus on the position update of avatars, which are the most common type of avatar states requiring constant updates. Based on the above system model, the position update in a MSDVE is accomplished as follows. The server maintaining the avatar updates the position of the avatar according to user's input commands and periodically (generally by frame) disseminates the position updates to the clients in the same zone. To improve accessibility and responsiveness, each client in the same zone will maintain a replicated copy of the avatar. When a new position update is received by a client, the replica of the avatar will be updated and reflected to the user.

Note that when a server disseminates position update to a client, the client may not be connected to the server. In this case, the position update should be forwarded by the server that is connected to the client. Consider an avatar and a replica of the avatar, we define *target server* of the replica as the server maintaining the avatar and *contact server* of the replica as the server that is connected by the client where the replica is. To send position update to the replica, if the contact server and the target server of the replica are the same (e.g., the replicas maintained by client c_1 in Figure 1),

the target server will directly disseminate position update to the replica. Otherwise, if the target server and the contact server are different (e.g., the replicas maintained by client c_3 in Figure 1), the target server will first send position update to the contact server in the manner of forwarding request, and then the contact server will forward position update to the replica.

In our system model, network bandwidth consumption on position update at each server can be divided into three parts. The first part is used to disseminate position updates to the replicas whose target server and contact server both are this server. The second part is used to send position updates in the manner of forwarding request for the replicas whose target server is this server and contact server is different. The third part is used to forward position update for the replicas whose contact server is this server and target server is another one. Suppose the network capacity for the position updates (the sum of three parts) is constrained for some servers, that is, each saturated server is allowed to send or forward position updates to only a given number of replicas. The aim of this paper is to investigate position update scheduling strategies for those saturated servers to reduce inconsistency while guarantee fairness in the whole DVE.

Based on the system model, we formally define inconsistency and unfairness using the metric of time-space inconsistency in a MSDVE. Time-space inconsistency is defined based on the following observation: participants make their judgements of a specific situation in a DVE based not only on the positions of entities, but also on the duration of this situation [25]. It has been shown that time-space inconsistency can effectively reflect the impact of inconsistency on a participant's perception and decision making. Therefore, it would be more reasonable to use time-space inconsistency to evaluate inconsistency and unfairness.

Consider a replica r of an avatar a. Let $\Delta(r, a, t)$ denote the spatial difference between the position of replica r at client side and the position of avatar a at server side at time t, the time-space inconsistency between r and a in a time session $[t_s, t_e]$ (denoted by s) is defined by

$$\Omega(r, a, s) = \int_{t_s}^{t_e} \Delta(r, a, t)dt \tag{1}$$

The total inconsistency associated with an avatar a in session s is defined as the total time-space inconsistency of all replicas of a in session s, which is given by

$$\Theta(a, s) = \sum_{r \in \mathcal{R}(a)} \Omega(r, a, s) \tag{2}$$

where $\mathcal{R}(a)$ denotes the set of all replicas of a. The total inconsistency of the DVE in session s is defined as the total inconsistency of all avatars in session s in the DVE, which is given by

$$\Theta(s) = \sum_{a} \Theta(a, s) \tag{3}$$

Consider an avatar a. Let $p(a, t)$ denote the position of a at server side at time t. For a replica r of a, let $p(r, t)$ represent the position of r at client side at time t. The divergence associated with a at time t is defined by

$$D(a, t) = \sqrt{\frac{1}{NR(a)} \cdot \sum_{r \in \mathcal{R}(a)} (p(r, t) - \overline{p}(a, t))^2} \tag{4}$$

where

$$\overline{p}(a, t) = \frac{1}{NR(a)} \cdot \sum_{r \in \mathcal{R}(a)} p(r, t) \tag{5}$$

and $NR(a)$ denotes the size of $\mathcal{R}(a)$. The unfairness associated with avatar a in session s is defined by

$$\Gamma(a, s) = \int_{t_s}^{t_e} D(a, t)dt \tag{6}$$

The unfairness of the MSDVE in session s is then defined by

$$\Gamma(s) = \sum_{a} \Gamma(a, s) \tag{7}$$

Based on the above definitions, this paper aims at two objectives:

1. The first objective is to minimize unfairness of the system in all sessions, i.e.,

$$minimize \sum_{s} \Gamma(s) \tag{8}$$

2. The second objective is to minimize total inconsistency of the system in all sessions, i.e.,

$$minimize \sum_{s} \Theta(s) \tag{9}$$

3. UPDATE SCHEDULES

In this section, we investigate position update schedules to minimize inconsistency while guarantee fairness in MSDVEs with network bandwidth constraints of servers. We first propose a fairness-aware update scheme which can guarantee fairness in a MSDVE. Based on the fairness-aware update scheme, we then propose an update scheduling algorithm which can minimize total inconsistency in a MSDVE. The basic idea of the scheduling algorithm is to update avatars according to their update priorities which are defined according to their potential impacts on inconsistency. When the network bandwidth is limited, the avatars with higher update priorities are updated first.

3.1 Fairness-aware Update Scheme

According to the definition, fairness can be achieved if each update of avatars is applied on the relevant replicas at the same time. Local lag is the most commonly used technique to achieve such goal [13]. For our problem, the basic idea is to add additional "lag" to the updates with smaller transmission delays so that the updates with larger transmission delays can be applied simultaneously. The "lag" can be performed either at server side (i.e., delaying disseminate update after it is generated) or at client side (i.e., delaying application of update after it is received) or both at server and client sides.

Consider a replica r. Let d_r denote the transmission delay of a position update from r's target server to r. Consider an avatar a. Let d_a^{max} denote the maximum transmission delay of a position update from the server maintaining avatar a

to all the replicas of a, i.e., d_a^{max} is the maximum d_r for all $r \in \mathcal{R}(a)$. Suppose a position update of avatar a is generated at time t. In the fairness-aware update scheme, the position update will be applied on all replicas of avatar a at time $t + d_a^{max}$. The time $t + d_a^{max}$ is defined as the *due time* of the position update of avatar a generated at time t. The additional lag to be added for replica r ($r \in \mathcal{R}(a)$) is $d_a^{max} - d_r$. In practical system, where the additional lag is performed will be flexibly determined by the update schedules according to the system context.

3.2 Derivations of Update Priority

The overview of the derivations of update priorities is as follows: firstly, we formulate the problem of finding optimal update schedule to minimize total inconsistency as an inequality constrained problem (ICP) for a static MSDVE. Secondly, Lagrange Multipliers are used to help derive the update priorities that can be estimated in practical systems.

Consider an avatar a. Due to transmission delay (including the lag that may be added by the fairness-aware update scheme), position update will not take effect on each replica until d_a^{max} later after it is generated. The growth of time-space inconsistency between each replica of avatar a and avatar a is given by

$$\Omega = \int_{t_{last}+d_a^{max}}^{t} \Delta(r, a, s)ds \tag{10}$$

where r is a replica of a and t_{last} is the time of the update last generated by the server maintaining a before $t - d_a^{max}$ [12]. To simplify the problem definition, we assume after each update of a is applied on the replicas, the difference between each replica's position and avatar a's position grows in the same manner following an increasing function $\delta(\cdot)$. Therefore, for a replica r of avatar a, we have $\Delta(r, a, t) = \delta(t - (t_{last} + d_a^{max})))$ and $\frac{d\delta(t)}{dt} > 0$. It is plausible to assume $\delta(\cdot)$ to be increasing, otherwise, the difference between the replica and the avatar is unchanged or getting smaller between two successive updates, position update is no longer needed in that case. With this assumption, Equation (10) can be rewritten as

$$\Omega = \int_{0}^{t-(t_{last}+d_a^{max})} \delta(s)ds \tag{11}$$

Note that $\delta(t)$ is the same for all the replicas of avatar a since update is applied with all the replicas at the same time. Similarly to Theorem 1 in [11], it can be proved that under fairness-aware update scheme, given a fixed number of updates allowed in a period for an avatar in a MSDVE, the updates should be generated periodically at server side over this period for minimizing total inconsistency over all replicas of this avatar. Therefore, to minimize total inconsistency, it just needs to determine the optimal update period for each avatar.

Suppose there are NA avatars in the DVE, which are denoted by $a_1, a_2, ..., a_{NA}$. Consider an avatar a_i ($1 \leq i \leq NA$). Suppose the update period of a_i is p_i, then the inconsistency between a_i and the replicas between two successive updates applied on the replicas is given by

$$NR(a_i) \cdot \int_0^{p_i} \delta_i(t)dt \tag{12}$$

The total inconsistency of the virtual world over period T is given by

$$\sum_{i=1}^{NA} \left(\frac{T}{p_i} \cdot NR(a_i) \cdot \int_0^{p_i} \delta_i(t)dt \right) \tag{13}$$

Then, we analyze the constraint of network bandwidth for each server. Suppose there are NS servers in the DVE, which are denoted by $s_1, s_2, ..., s_{NS}$. Consider a server s_j ($1 \leq j \leq NS$). s_j needs to send position updates directly to the replicas whose target server and contact server are both s_j. Moreover, s_j needs to send forwarding requests for the replicas whose target server is s_j but contact server is different. Let α denote the bandwidth consumption for disseminating one position update directly or in the manner of forwarding request. The total network bandwidth requirements for these two parts over T are

$$\alpha \cdot \left(\sum_{a_i \in \mathcal{R}(s_j)} NR(a_i) \cdot \frac{T}{p_i} \right) \tag{14}$$

where $\mathcal{R}(s_j)$ denotes the set of avatars maintained by s_j. In addition, s_j needs to forward position updates to the replicas whose contact server is s_j but target server is different. The bandwidth consumption on this part over T is given by

$$\alpha \cdot \left(\sum_{a_k \notin \mathcal{R}(s_j)} NR(a_k, s_j) \cdot \frac{T}{p_k} \right) \tag{15}$$

where $NR(a_k, s_j)$ denotes the number of replicas of avatar a_k whose contact server is s_j. Suppose the network capacity of s_j for position update at each update frame is constrained by c_j. Let f denote the update frame length of each server. The total network capacity over T will be constrained by $c_j \cdot \frac{T}{f}$. Therefore, we have

$$\alpha \cdot \left(\sum NR(a_i) \cdot \frac{T}{p_i} + \sum NR(a_k, s_j) \cdot \frac{T}{p_k} \right) \leq c_j \cdot \frac{T}{f} \tag{16}$$

Thus, the problem can be defined as the following inequality constrained problem (ICP), the objective is to minimize

$$f(p) = \sum_{i=1}^{NA} \left(\frac{T}{p_i} \cdot NR(a_i) \cdot \int_0^{p_i} \delta_i(t)dt \right) \tag{17}$$

subject to

$$g_j(p) \leq 0, \quad 1 \leq j \leq NS \tag{18}$$

, where $p = [p_1, ..., p_{NA}]$ and $g_j(p)$ is defined as

$$g_j(p) = \alpha \cdot \left(\sum NR(a_i) \cdot \frac{T}{p_i} + \sum NR(a_k, s_j) \cdot \frac{T}{p_k} \right) - c_j \cdot \frac{T}{f}$$

Based on the ICP formulated above, using Lagrange Multipliers, the update priority of avatar a_i at time t can be defined by

$$\frac{1}{\varphi_i(t)} \cdot \big((t - t_{last}) \cdot \Delta(r, a_i, t + d_{a_i}^{max}) - \int_{t_{last}+d_{a_i}^{max}}^{t+d_{a_i}^{max}} \Delta(r, a_i, s)ds \big) \tag{19}$$

φ_i is defined as

$$\frac{\alpha \cdot NR(a_i) \cdot \mu_k + \alpha \cdot \sum_{j=1, j \neq k}^{NS} (NR(a_i, s_j) \cdot \mu_j)}{NR(a_i)} \quad (20)$$

where μ_i $(1 \leq i \leq NS)$ are Lagrange Multipliers. The derivations are similar to that in [11]. The only difference is the update priority was derived in terms of each replica in [11], while the update priority is derived in terms of each avatar whereas in this paper.

To calculate the update priority, the values of $\Delta(r, a_i, t + d_{a_i}^{max})$, $\int_{t_{last}+d_{a_i}^{max}}^{t+d_{a_i}^{max}} \Delta(r, a_i, s)ds$ and μ_i $(1 \leq i \leq NS)$ should be estimated by the server at time t. The estimations of these values in practical systems have been discussed in [11].

3.3 Update Scheduling Algorithm

Based on the above analysis, we investigate a fairness-ware update scheduling algorithm for minimizing total inconsistency. The update scheduling algorithm is designed towards achieving the following two targets:

- To maintain fairness, each position update of an avatar should be applied on all replicas of the avatar at same time.
- To minimize inconsistency, the avatar with highest update priority among all avatars in the virtual world should be updated.

The update scheduling algorithm is denoted as Fair-LMH. There are two data structures maintained by each server for running Fair-LMH.

- **AvatarList**: the avatar list maintained by each server. At each update frame, the avatars maintained by each server are sorted according to update priorities and put into the AvatarList.
- **RequestList[]**: an array of lists for storing updates (position update or forwarding request). If an update is stored in the list **RequestList[i]**, it means that the update can be delayed at most i frames at this server.

As has been mentioned earlier, each server will generate position updates (for the replicas whose target server is this server) and receive forwarding requests (for the replicas whose contact server is this server but target server is different). All the generated position updates and received forwarding requests will be first placed in **RequestList** before sent out.

Consider a position update generated by server s_i. Suppose the update is for replica r (s_i is the target server of r). The update will be put into the list **RequestList[j]** at server s_i, where j is equal to

$$\lfloor \frac{d_{a(r)}^{max} - d_r}{f} \rfloor \quad (21)$$

Consider a forwarding request received by server s_i. Suppose the request is received at time t and the update is for replica r (r's contact server is s_i but target server is not s_i). Let t' denote the time when the update is generated at r's target server. The forwarding request will be put into the list **RequestList[j]** at server s_i, where j is equal to

$$\lfloor \frac{d_{a(r)}^{max} - (t' - t)}{f} \rfloor \quad (22)$$

Algorithm 1 Fair-LMH (Server Side)

1: Put all avatars into **AvatarList** and sort them according to update priority
2: Move the items in list **RequestList[i]** to list **RequestList[i-1]** for all $i > 0$
3: Put each received forwarding request since last update frame into **RequestList**
4: Disseminate the updates in **RequestList[0]** as many as possible
5: **while** (**AvatarList** is not empty and bandwidth is not used up) **do**
6: Select the avatar with highest priority in **AvatarList**
7: Generate position updates for the selected avatar and put each of them into **RequestList**
8: Disseminate the updates in **RequestList[0]** as many as possible
9: Remove the selected avatar from **AvatarList**
10: **end while**
11: **while** (There are some unempty lists in **RequestList** and bandwidth is not used up) **do**
12: Disseminate the updates in the unempty list with smallest index
13: **end while**

Algorithm 2 Fair-LMH (Client Side)

1: Assume the client receives an update at time t. Suppose the due time to apply the update (according to the fairness-aware update scheme) is t_d. If $t_d > t$, the update will be applied $t_d - t$ time later, otherwise, the update is applied immediately.

The Fair-LMH algorithm has two parts, which run at server side and client side respectively. The part for server side, shown in Algorithm 1, runs at each update frame. At each update frame, all the avatars maintained by the server are first put into **AvatarList** and sorted according to update priority which can be calculated by (19) (line 1). Then, items in list **RequestList[i]** are moved to list **RequestList[i-1]** for all $i > 0$ (line 2) since one update frame has elapsed. After that, each received forwarding request since last update frame is placed in **RequestList**, where the index of the list is calculated according to (22) (line 3). Then, the most urgent updates which are in list **RequestList[0]** are sent out first (line 4). Next, avatars are handled in decreasing order of their update priorities as long as there is network bandwidth available (lines 5 to 10). To handle an avatar, the position updates are generated first and placed in **RequestList**. Then, the most urgent updates which were placed in list **RequestList[0]** are sent out. If the network bandwidth is still available after all the avatars in **AvatarList** are handled, we continue processing each unempty list in **RequestList** (lines 11 to 13). Each time, the list with smallest index is processed and all updates in the list will be disseminated if there is sufficient bandwidth.

The part for client side is shown in Algorithm 2. When a client receives an update, if the receiving time is earlier than the due time of the update, the update will be issued at the due time. Otherwise, the update is applied on the replica immediately.

4. EXPERIMENTAL EVALUATION

4.1 Simulation Setup

We implemented an event-driven simulator to simulate a MSDVE. The virtual world is of scale 10000 x 10000, which was divided into 25 equal sized squares, each of them represents a zone. A total of 10 servers and 20000 clients were simulated. Each avatar is associated with a client, which was randomly distributed in the virtual world initially. The client assignment and zone mapping were determined by the approaches proposed in [12]. The length of a frame was set at 0.025 seconds, i.e., there are 40 frames per second.

The avatars move around the virtual world with random way point (RWP) mobility model. Specifically, as the simulation starts, each avatar selects one location in the virtual world as the destination. It then travels towards this destination with constant velocity chosen uniformly and randomly from [0.1, 1] distance units per frame. Upon reaching the destination, the avatar again chooses another random destination in the virtual world and move towards it.

In our simulation, the transmission delay from a server to a client was generated according to a shifted exponential distribution with probability density function $f(x) = \lambda e^{-\lambda(x-\tau)}(x \geq \tau)$ [5]. To generate a mean transmission delay d with a variance of v $(0 < v < 1)$, τ and λ should be set to $v * d$ and $\frac{1}{(1-v)*d}$ respectively.

Without loss of generality, the bandwidth consumption of position update and forwarding request, i.e., α was set at 1. The bandwidth capacity at each server was set to allow the server to send a given number of c_i updates at each frame, where c_i was varied in the experiments.

4.2 Update Algorithms to Compare

We also implemented the LMH algorithm for comparison, which is the optimal update algorithm proposed in our previous work for minimizing total inconsistency in MSDVEs [11]. However, the fairness is not considered in LMH. Moreover, we implemented some intuitive fairness-aware update algorithms including Fair-FF and Fair-SPACE. The Fair-FF and Fair-SPACE also follow the updating algorithm described in Algorithm 1 and Algorithm 2. The only difference between Fair-FF, Fair-SAPCE and Fair-LMH is on how the update priority of each avatar is calculated.

- Fair-RR: the update priority of each avatar is defined as the time since last update was generated.
- Fair-Space: the update priority of each avatar is defined as the spatial difference between the avatar and the replicas of the avatar.

4.3 Experimental Results

In the simulations, each experiment runs for a period of 30 minutes where the first 10 minutes was considered the warm-up period. Statistics were collected for the remaining 20 minutes of simulated time. In the results, the inconsistency and unfairness are presented as a ratio between 0 and 1 after normalization. For each group of result, let v_{min} and v_{max} denote the maximum value and the minimum value of the data respectively. The normalized ratio of a value v is calculated by $(v - v_{min})/(v_{max} - v_{min})$.

4.4 Impact of Bandwidth Capacity

We first evaluated the impact of bandwidth capacity on the performance of the algorithms. For each experiment, the bandwidth capacity was set in a range from 100 to 6000 updates per frame at each server. The average network delay was set at 0.1s with a variance of 0.95.

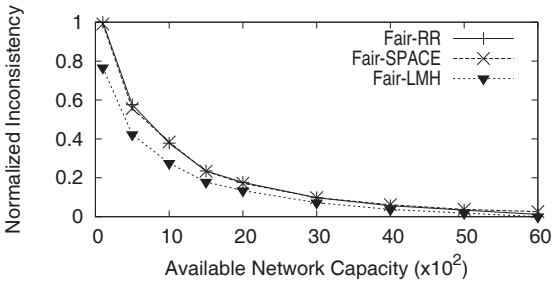

Figure 2: Impact of Bandwidth Capacity on Inconsistency

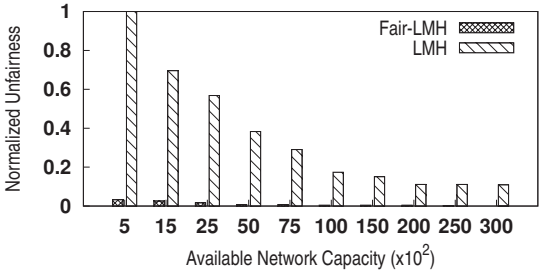

Figure 3: Impact of Bandwidth Capacity on Unfairness

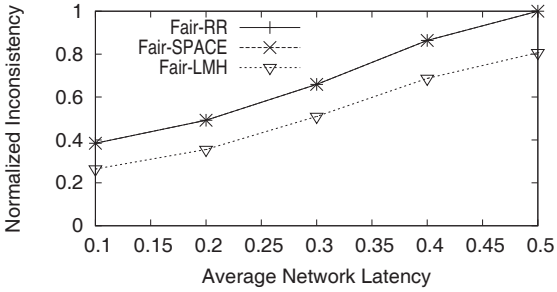

Figure 4: Impact of Network Delay on Inconsistency

Figure 2 shows the normalized inconsistency achieved by different algorithms. As can be seen, lower bandwidth makes larger inconsistency for all algorithms. For small bandwidth capacity, which implies more serious bandwidth constraints, Fair-LMH performs much better than other algorithms. For large bandwidth capacity, all algorithms get similar performance. This is because all the replicas can be updated frequently so the inconsistency is small.

Figure 3 shows the normalized unfairness achieved by LMH and Fair-LMH. As can be seen, Fair-LMH can greatly decrease the unfairness compared to LMH for all bandwidth capacities.

4.5 Impact of Network Delay

Next, we evaluated the impact of network delay on the performance of the algorithms. For each experiment, the

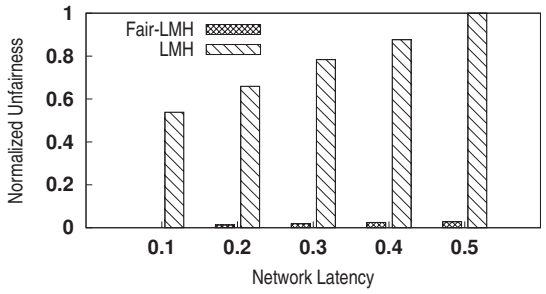

Figure 5: Impact of Network Delay on Unfairness

mean network delay was set in a range of 0.1s to 0.5s. The variance of network delay was set at 0.95. The bandwidth capacity was set at 1000 updates per frame at each server.

Figure 4 shows the normalized inconsistency achieved by different algorithms. As can be seen, the total inconsistency gets larger as the average network delay increases. This is because time-space inconsistency is affected by transmission delays and generally increases with the delays. The Fair-LMH algorithm alway outperforms other algorithms for various network delays.

Figure 5 shows the normalized unfairness incurred by Fair-LMH and LMH with different network delays. As can be seen, as the network delay increases, the unfairness incurred by the two algorithms both increases. For various network delays, the unfairness incurred by Fair-LMH is much smaller than the unfairness incurred by LMH.

4.6 Impact of Network Delay Variance

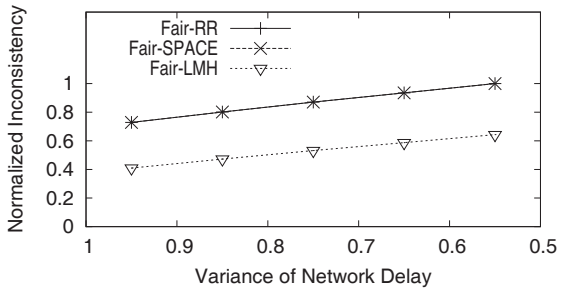

Figure 6: Impact of Network Delay Variance on Inconsistency

At last, we evaluated the impact of the variance of network delays on the performance of the algorithms. For each experiment, the mean network delay was set at 0.1s. The variance of network delay was set in a range of 0.55 to 0.95. The bandwidth capacity was set at 1000 updates per frame at each server.

Figure 6 shows the normalized inconsistency incurred by different algorithms. As can be seen, the total inconsistency gets larger as the variance of network delay increases. However, the growth is not quite obvious. The Fair-LMH algorithm always performs better than other algorithms for all variance settings.

Figure 7 shows the normalized unfairness incurred by Fair-LMH and LMH. As can be seen, similar to the inconsistency, the impact of network variance on the unfairness is also not

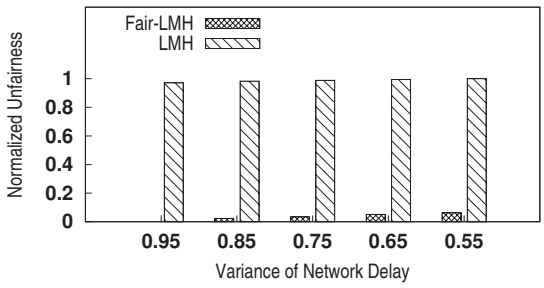

Figure 7: Impact of Network Delay Variance on Unfairness

obvious. The Fair-LMH algorithm always performs better than LMH for all variance settings.

5. RELATED WORK

The update scheduling problem for improving consistency in DVEs has been studied a lot in the literature. A state update scheduling algorithm to minimize total inconsistency in single server DVEs with constraint of network bandwidth was proposed in [19]. The proposed algorithm can be easily extended for multi-server DVEs where a client is allowed to directly connect to multiple servers. In our previous work [9, 11], we considered MSDVEs and proposed a heuristic update algorithm to prioritize state updates from a global perspective for improving consistency in MSDVEs where each server has a limit network bandwidth. However, the unfairness is not considered in any of the work mentioned above.

Research has also been carried out to address unfairness in DVEs. The causal relationship between propagation time, inconsistencies, playability and fairness in online multiplayer games was explored in [3]. Two strategies were proposed for managing playability and fairness in online games. Zander et.al [23] studied the impact of delay differences among players on the fairness in multiplayer network gaming. An approach that can be used with the existing network games to equalize the delay differences was designed and implemented. However, in all the work mentioned above, only network delay is considered while the spatial difference is not considered in the measurement.

Many research activities have been carried out to address the bandwidth reduction issues in DVEs. For example, dead reckoning [3, 16, 24] and relevance filtering [15, 20] are widely used in DVEs to compensate the network latency and reduce the network traffic. By maintaining a DR prediction model, remote node can predict the state of the replicated entity between two consecutive update packets thus update frequency can be reduced. Relevance filtering can eliminate the irrelevant information by using the concept of Area of Interest (AOI). For a single avatar, if some entities are not in the avatar's AOI (that implies the avatar is not interested in these entities), state updates of these entities can be saved. Although the total network traffic can be greatly reduced by using these techniques, the total bandwidth requirement may still be very high as the population of the DVE grows. If servers are not able to disseminate all state updates due to the constraints of network capacity, our algorithm is to determine the updating priority for each state update according to their potential impact on inconsistency

and unfairness. Therefore, our algorithm can be easily used on top of those techniques.

6. CONCLUSIONS

In this paper, we investigated update schedules for improving fairness and consistency in MSDVEs with network bandwidth constraints for servers. A fairness-aware update scheme was proposed to guarantee fairness among clients in a MSDVE. Based on the fairness-aware update scheme, a distributed state update scheduling algorithm was proposed to minimize total inconsistency. A variety of experiments were performed to evaluate the proposed algorithm. From the results we can see that the proposed algorithm outperforms other algorithms. However, the proposed algorithm is only evaluated by simulations. In the future, we would like to evaluate the proposed algorithm in some real large scale DVE systems.

7. ACKNOWLEDGMENTS

This research is supported by the National Research Foundation, Prime Minister's Office, Singapore under its IDM Futures Funding Initiative, and Singapore Ministry of Education Academic Research Fund Tier 2 under Grant MOE2013-T2-2-067.

8. REFERENCES

[1] C. Bouras, G. Hornig, V. Triantafillou, and T. Tsiatsos. Architectures supporting e-learning through collaborative virtual environments: the case of INVITE. In *IEEE International Conference on Advanced Learning Technologies*, pages 13–16, 2001.

[2] J. Brun, F. Safaei, and P. Boustead. Fairness and playability in online multiplayer games. *Faculty of Informatics-Papers*, page 232, 2006.

[3] J. Brun, F. Safaei, and P. Boustead. Managing latency and fairness in networked games. *Communications of the ACM*, 49(11):46–51, 2006.

[4] J. Chim, R. W. H. Lau, H. V. Leong, and A. Si. Cyberwalk: A web-based distributed virtual walkthrough environment. *IEEE Transactions on Multimedia*, 5:503–515, 2003.

[5] A. Corlett, D. Pullin, and S. Sargood. Statistics of one-way internet packet delays. *53 rd IETF*, 2002.

[6] Everquest. http://everquest.station.sony.com.

[7] T. Funkhouser. Ring: a client-server system for multi-user virtual environments. In *Proceedings of Symposium on Interactive 3D graphics*. ACM, 1995.

[8] M. Hori, T. Iseri, K. Fujikawa, S. Shimojo, and H. Miyahara. Scalability issues of dynamic space management for multiple-server networked virtual environments. In *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, volume 1, pages 200–203. IEEE, 2001.

[9] Y. Li and W. Cai. Determining optimal update period for minimizing inconsistency in multi-server distributed virtual environments. In *2011 IEEE/ACM 15th International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, pages 126–133. IEEE, 2011.

[10] Y. Li and W. Cai. Consistency-aware partitioning algorithm in multi-server distributed virtual environments. In *2012 IEEE 26th International Parallel & Distributed Processing Symposium (IPDPS)*, pages 798–807. IEEE, 2012.

[11] Y. Li and W. Cai. Update schedules for improving consistency in multi-server distributed virtual environments. *Journal of Network and Computer Applications*, 41:263–273, 2014.

[12] Y. Li and W. Cai. Consistency-aware zone mapping and client assignment in multi-server distributed virtual environments. *IEEE Transactions on Parallel and Distributed Systems*, 26(6):1570–1579, 2015.

[13] Z. Li, X. Tang, W. Cai, and S. J. Turner. Fair and efficient dead reckoning-based update dissemination for distributed virtual environments. In *Principles of Advanced and Distributed Simulation (PADS), 2012 ACM/IEEE/SCS 26th Workshop on*, pages 13–22. IEEE, 2012.

[14] B. Ng, A. Si, R. Lau, and F. Li. A multi-server architecture for distributed virtual walkthrough. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, 2002.

[15] K. Pan, W. Cai, X. Tang, S. Zhou, and S. Turner. A hybrid interest management mechanism for peer-to-peer networked virtual environments. In *IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*, 2010.

[16] L. Pantel and L. Wolf. On the suitability of dead reckoning schemes for games. In *Proceedings of the 1st Workshop on Network and System Support for Games (NetGames)*, 2002.

[17] SecondLife. http://secondlife.com.

[18] P. Svoboda, W. Karner, and M. Rupp. Traffic analysis and modeling for world of warcraft. In *IEEE International Conference on Communications (ICC)*, pages 1612–1617, 2007.

[19] X. Tang and S. Zhou. Update scheduling for improving consistency in distributed virtual environments. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 21:765–777, 2010.

[20] D. Van Hook, S. Rak, and J. Calvin. Approaches to relevance filtering. In *Proc. of 11th DIS Workshop*, 1994.

[21] WoW. http://www.worldofwarcraft.com/index.xml.

[22] WoWCencus. http://www.warcraftrealms.com/censusplus.php.

[23] S. Zander, I. Leeder, and G. Armitage. Achieving fairness in multiplayer network games through automated latency balancing. In *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology*, pages 117–124. ACM, 2005.

[24] Y. Zhang, L. Chen, and G. Chen. Globally synchronized dead-reckoning with local lag for continuous distributed multiplayer games. In *Proceedings of 5th ACM SIGCOMM Workshop on Network and System Support for Games (NetGames)*, 2006.

[25] S. Zhou, W. Cai, B.-S. Lee, and S. J. Turner. Time-space consistency in large-scale distributed virtual environments. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 14(1):31–47, 2004.