Consistency Aware Update Schedule in Multi-Server Distributed Virtual Environments

Yusen Li
Parallel and Distributed Computing Center
Nanyang Technological University
Singapore 639798
S080007@ntu.edu.sg

Wentong Cai
Parallel and Distributed Computing Center
Nanyang Technological University
Singapore 639798
ASWTCAI@ntu.edu.sg

ABSTRACT

Multi-server architecture is very popular in Distributed Virtual Environments. Although multi-server architecture has good scalability, the load balance and consistency are still challenging issues. Because of server resource limitations (computation, communication etc), the consistency of the virtual world cannot be guaranteed sometimes. In this paper, we address this problem by developing scheduling polices and try to get better overall consistency with limited resources. We formulate the problem by using integer programming model and propose a centralized approximate algorithm. A distributed update strategy is also developed accordingly. Our algorithm is validated by theoretical analysis and wide range of experiments.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems

General Terms

Algorithms, Performance

Keywords

DVE, Multi-server, Consistency, Update Scheduling

1. INTRODUCTION

Distributed Virtual Environments (DVEs) technology permits real-time human interaction with complex environments based on 3D graphical description that models the real-world with physical constrains [10]. So far, DVEs have been used mostly for military purposes, e-learning and online networked games [12].

In multi-server client/server architecture, each client is connected to one of the servers and all the servers are connected to each other. In order to share the workload, the virtual world is often partitioned into many unoverlapped regions, each of which is mapped to a server. If a server maintains a specified region, it also takes charge of avatars in this region, including rendering the interactions with the virtual world and disseminating update packets to other clients. Due to the dynamic characteristic of the virtual world, workload distribution may be imbalanced among servers. When servers get overloaded (computation, communication etc), the state update packets may not be disseminated timely and thus, the consistency of the virtual environment cannot be guaranteed. Even with the load-balancing technique adopted, some servers may still become saturated sometimes because most of the load-balancing methods cannot be executed frequently due to the high overhead and long execution time. Therefore, it is necessary to consider update schedule for overloaded servers to improve the consistency in DVEs.

The problem of providing QoS in a DVE system has been already described in many ways. Firstly, some research work has been done on the influences of network quality on consistency and player departure behavior in DVEs [7, 14]. It shows the consistency plays an important role in the players' behaviors. In order to compensate the network latency and reduce the traffic workload, research work has also been conducted on dead reckoning [8, 5, 16] and relevance filtering [3, 9]. Most of these work focuses on latency hiding and bandwidth reduction without considering consistency. For multi-server architecture in DVEs, many partitioning algorithms have been proposed for load balance [6, 11]. Unfortunately, again, most of the existing work does not consider the consistency maintenance.

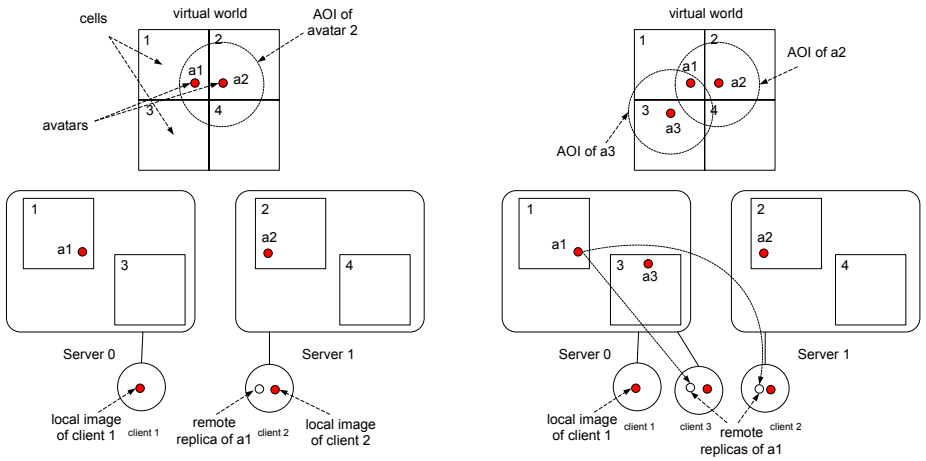
In this paper, we try to reduce the inconsistency of virtual environment by adapting update schedule for each server. We formulate the problem by using integer programming and propose a fully distributed update strategy. By theoretical analysis and simulations, the results show that our algorithms can substantially improve the consistency of the DVEs when some servers are saturated.

The rest of the paper is structured as follows. The background information is introduced in the second section. The problem definition is described in the third section. In the fourth section, a two phase centralized approximate algorithm is proposed to get an approximate solution. A distributed update strategy is developed in the fifth section. The simulations and results are shown in the sixth section. The conclusion and future work will be discussed in the last section.

2. BACKGROUND

DISIO 2010 March 15, Torremolinos, Malaga, Spain.

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2010.8669
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8669



(a) (b)

Figure 1: DVE on Multi-Server Architecture

2.1 Architecture Description

We consider multi-server architecture in DVEs. In this architecture, multiple servers are connected to each other in a peer-to-peer manner via high speed links. A client is connected to one of the servers based on their virtual locations in the virtual world. The virtual world is partitioned into several fixed cells. Each server maintains a portion of these cells. Client connects to a server if its avatar is currently residing in the cell maintained by this server. Each avatar has an Area of Interest (AOI).

Figure 1(a) shows the concepts we just described. There are two servers and two clients in the physical world. The virtual world is partitioned into four cells where cells 1 and 3 are maintained by Server 0 and cells 2 and 4 are maintained by Server 1. The dashed circle represents the AOI of avatar a2. Since avatar a1 is in avatar a2's AOI, there is remote replica of a1 maintained by client 2's host station.

2.2 Server Side Bandwidth Limitation

The cost of bandwidth is a primary budget for the server provider in server based applications. In DVEs, the bandwidth consumption from server to clients (i.e., uploading bandwidth for server) should be considered with high priority because server should disseminate the update packets to clients and replicas timely. Here we assume that the uploading bandwidth is limited for each server.

For each avatar maintained by server S_i, its remote replicas are divided into two types. The first type of replicas are maintained by the clients who are also connected to S_i. In order to update these replicas, S_i needs to send one update packet to each of them. The second type of replicas are maintained by the clients who are connected to a different server S_j where $j \neq i$. For these replicas, S_i should first send a relaying request to S_j, then S_j should relay the update packets to the replicas. For each avatar, the relaying request just needs to be sent once from S_i to S_j. Figure 1(b) shows an example. Client 1 has its avatar a1 maintained by Server 0. Client 3 is connected to the same server as Client 1 and Client 2 is connected to a different server. In order to update the replicas of a1, one update packet is needed from Server 0 to Client 3 and one relaying request is needed from Server 0 to Server 1. In addition, Server 1 needs relay the update packet to Client 2.

According to the above updating mechanism, each server's uploading bandwidth is divided into three parts. The first part is used to disseminate the update packets to clients connected to the server directly. The second part is used to send relaying request to other servers. The third part is used to disseminate the update packets received from other servers.

2.3 Time-Space Inconsistency

The concept of time-space inconsistency in DVEs is proposed in [13]. It reflects the difference between an avatar and its remote replica in DVEs. In multi-server architecture, we consider inconsistency between each avatar and its remote replicas. In order to save network bandwidth, we assume dead-reckoning model is used to predict trajectories. If we ignore the clock drift among all the clients and servers, we can get the time-space inconsistency between an avatar and its replica [13] by the following equation:

$$\Omega = \frac{\delta}{2}(T_{DR} + T_d) + |a|_{max} T_{DR} * \frac{1}{2} T_d^2 \quad (1)$$

T_{DR} is the time interval between two update packets. T_d is the total latency of the update packet from the avatar to its replica. For dead-reckoning to be effective, T_{DR} should be larger than T_d [13]. δ is the spacial threshold for the dead-reckoning algorithm and $|a|_{max}$ represents the max acceleration of the avatar.

To determine T_d, there are two situations that need to be considered. Figure 2(a) shows the situation between an avatar and its replicas maintained by the clients connected to the same server (e.g., avatar a1 and the remote replica of a1 maintained by Client 3 in Figure 1(b)). Figure 2(b) shows the other situation between an avatar and its replicas maintained by the clients connected to a different server (e.g., avatar a1 and the remote replica maintained by Client 2 in Figure 1(b)). In Figure 2(a), T_d is the network latency from

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2010.8669
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8669

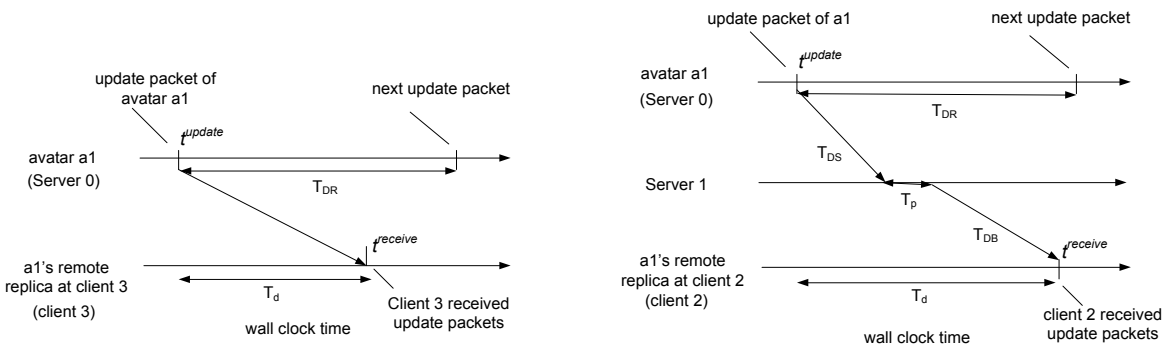


Figure 2: T_d and T_{DR} used in Time-space Inconsistency

Server 0 to Client 3. In Figure 2(b), $T_d = T_{DS} + T_p + T_{DB}$. T_{DS} is the network latency from Server 0 to Server 1. T_{DB} represents the network delay from Server 1 to Client 2. T_p represents the delaying time of the update packet at Server 1.

We assume that server will maintain the dead reckoning model for each replica and a replica is called replica without QoS if its Ω is greater than a predefined threshold value.

3. PROBLEM DEFINITION

We focus on the position update packets, that is because position update packets are sent most frequently and consume most of the network bandwidth. For clients, they send update packets to the server according to the players' input commands and receive other clients' update packets from the server. For servers, they periodically disseminate the new state of avatars to update the replicas without QoS. The period between two successive refreshes is known as a frame. For each frame, if available bandwidth cannot afford to disseminate or relay state updates to all the replicas without QoS, scheduling algorithms and relaying strategies for each server need to be developed. We take the number of replicas without QoS who cannot receive the update packets due to the network capacity as the metric to evaluate the inconsistency in each frame. Therefore, the objective is to maximize the number of replicas without QoS who can receive the update packet in every frame.

According to the concepts introduced in the previous section, we first define the notations as follows:

- N: the number of servers in the DVE
- S_i: ith server in the DVE, $1 \leq i \leq N$
- n_i: the number of avatars maintained by S_i, $1 \leq i \leq N$
- $c_{i,j}$: the jth client connected to S_i, $1 \leq i \leq N$, $1 \leq j \leq n_i$
- $a_{i,j}$: the one-to-one corresponding avatar of client $c_{i,j}$, $1 \leq i \leq N$, $1 \leq j \leq n_i$
- $r^k_{i,j}$: the number of $a_{i,j}$'s remote replicas without QoS. These replicas are maintained by clients connected to S_k.
- $\Phi(r^k_{i,j})$: function to map $r^k_{i,j}$ to 0 or 1, which indicates whether there needs a relaying request from S_i to S_k $(k \neq i)$ for updating $a_{i,j}$'s remote replicas,

$$\Phi(r^k_{i,j}) = \begin{cases} 1, & r^k_{i,j} > 0, k \neq i \\ 0, & r^k_{i,j} = 0, k \neq i \end{cases}$$

- B_i: capacity of uploading bandwidth for S_i, $1 \leq i \leq N$
- α: bandwidth consumption of one update packet
- β: bandwidth consumption of one relaying request

We also have some variables which will be used in the problem definition:

- x_i is used to represent how many replicas in $\sum_{j=1}^{n_i} r^i_{i,j}$ will be updated by S_i, $1 \leq i \leq N$.
- $y^k_{i,j}$, a 0-1 variable, is used to represent whether S_i will send relaying request to S_k, $i \neq k$, to update $a_{i,j}$'s remote replicas maintained by the clients connected to S_k. 1 means will; whereas 0 means will not.
- $z^k_{i,j}$, a 0-1 variable, is valid if and only if $y^k_{i,j} = 1$. It is used to represent whether S_k will relay the update packets for S_i, $i \neq k$, to update $a_{i,j}$'s remote replicas maintained by the clients connected to S_k. 1 means will; whereas 0 means will not.

Using the above notations, we define the problem as follows. The objective function is:

$$max \sum_{i=1}^{N} (x_i + \sum_{k=1,k\neq i}^{N} \sum_{j=1}^{n_k} y^i_{k,j} * z^i_{k,j} * r^i_{k,j})$$

with constrains:

$$\alpha x_i + \beta \sum_{k=1,k\neq i}^{N} \sum_{j=1}^{n_i} y^k_{i,j} + \alpha \sum_{k=1,k\neq i}^{N} \sum_{j=1}^{n_k} y^i_{k,j} * z^i_{k,j} * r^i_{k,j} \leq B_i \quad (2)$$

for all $1 \leq i \leq N$.

The objective function is trying to maximize the number of replicas without QoS to be updated in the DVEs. x_i represents the number of replicas updated by S_i directly. $\sum_{k=1,k\neq i}^{N} \sum_{j=1}^{n_k} y^i_{k,j} * z^i_{k,j} * r^i_{k,j}$ represents the number of

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2010.8669
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8669

replicas updated by S_i by relaying other servers' relaying requests. The sum of these two parts is the total number of replicas without QoS updated by S_i. The first part in constrain (2) represents the bandwidth consumption on sending update packets directly to the clients connected to S_i itself. The second part in constrain (2) represents the bandwidth consumption of S_i on sending relaying requests to other servers. The third part represents the bandwidth consumption of S_i on relaying the update packets from other servers. This is an integer programming problem and is NP hard. It is difficult to get an optimal solution in polynomial time and especially hard in distributed manner.

4. CENTRALIZED APPROXIMATE ALGORITHM

We first propose a centralized approximate algorithm (CAA) to get an approximate solution for the integer programming problem.

Since we know the global knowledge of all the servers in the centralized manner, we firstly try to simplify the problem definition by introducing the concept of equivalent solution. We define the equivalent solutions as the solutions which can update the same number of replicas without QoS. We also define a failed relaying request as the one which is received by a server but not relayed by the server (i.e., $y^k_{i,j} = 1$, $z^k_{i,j} = 0$). We can prove that for any solution with failed relaying request, we can always find an equivalent solution without any failed relaying request. For instance, we assume there is an solution A with $y^k_{i,j} = 1$ and $z^k_{i,j} = 0$. We can propose another solution B and the only difference between A and B is that $y^k_{i,j}$ is set to 0 in B. Therefore, the failed relaying request does not exist in B. Due to $y^k_{i,j} * z^k_{i,j}$ is 0 in both A and B, the numbers of replicas without QoS updated by A and B are the same, i.e., they are equivalent. Based on this conclusion, we can just consider the solution that does not contain any failed relaying request, i.e., $z^k_{i,j} = y^k_{i,j}$ is always satisfied for $1 \leq i, k \leq N$ and $1 \leq j \leq n_i$. Therefore, if we can determine the values of all $y^k_{i,j}$, we can then get x_i for server S_i by the following equation according to (2): $x_i = \frac{1}{\alpha}(B_i - \beta \sum_{k=1,k\neq i}^{N} \sum_{j=1}^{n_i} y^k_{i,j} - \alpha \sum_{k=1,k\neq i}^{N} \sum_{j=1}^{n_k} y^i_{k,j} * z^i_{k,j} * r^i_{k,j})$, where $z^i_{k,j} = y^i_{k,j}$.

Algorithm 1 shows the description of CAA. There are two phases in CAA. We assume $B_i > \beta \sum_{k=1,k\neq i}^{N} \sum_{j=1}^{n_i} \Phi(r^k_{i,j}) + \alpha \sum_{k=1,k\neq i}^{N} \sum_{j=1}^{n_k} \Phi(r^i_{k,j}) * r^i_{k,j}$ is always satisfied, that is because the relaying events only occur at the cell boundaries in the virtual world. The total bandwidth consumption on the relaying requests defined by $\beta \sum_{k=1,k\neq i}^{N} \sum_{j=1}^{n_i} \Phi(r^k_{i,j}) + \alpha \sum_{k=1,k\neq i}^{N} \sum_{j=1}^{n_k} \Phi(r^i_{k,j}) * r^i_{k,j}$ of server S_i should be small compared to the total available network bandwidth B_i. In the first phase, we make an initial solution by setting all $z^k_{i,j} = y^k_{i,j} = \Phi(r^k_{i,j})$ and $x_i = \frac{1}{\alpha}(B_i - \beta \sum_{k=1,k\neq i}^{N} \sum_{j=1}^{n_i} \Phi(r^k_{i,j}) - \alpha \sum_{k=1,k\neq i}^{N} \sum_{j=1}^{n_k} \Phi(r^i_{k,j}) * r^i_{k,j})$ (lines 2 to 7).

The objective of phase 2 is to save on the bandwidth used in relaying requests and to increase number of local updates. Lines 9 to 19 in Algorithm 1 show the refining process. For each $y^k_{i,j} = 1$, we check whether $r^k_{i,j} < \sum_{j=1}^{n_k} r^k_{k,j} - x_k$ is satisfied. If it is satisfied, $y^k_{i,j}$ can be reset to 0. This means that S_i does not need to send the relaying request $r^k_{i,j}$ to S_k. This is because S_k has more than $r^k_{i,j}$ replicas without QoS connected directly to itself, which have not been updated. S_k can update the same number of replicas without QoS instead of disseminating $r^k_{i,j}$ update packets for server S_i. Therefore, the relaying request from S_i can be saved and the total updated replicas without QoS is not reduced. As a result, S_i will be also able to update more of its local replicas that are without QoS. Accordingly, x_k and x_i should be updated (lines 13 to 15).

Algorithm 1 Description of CAA

1: ***Phase One***
2: **for** Each $y^k_{i,j}$ where $1 \leq i \leq N$, $1 \leq j \leq n_i$ and $1 \leq k \leq N$, $k \neq i$ **do**
3: Set $z^k_{i,j} = y^k_{i,j} = \Phi(r^k_{i,j})$
4: **end for**
5: **for** $1 \leq i \leq N$ **do**
6: Set $x_i = \frac{1}{\alpha}(B_i - \beta \sum_{k=1,k\neq i}^{N} \sum_{j=1}^{n_i} y^k_{i,j} - \alpha \sum_{k=1,k\neq i}^{N} \sum_{j=1}^{n_k} y^i_{k,j} * z^i_{k,j} * r^i_{k,j})$
7: **end for**
8: ***Phase Two***
9: **for** $1 \leq i \leq N$ **do**
10: **for** $1 \leq j \leq n_i$ **do**
11: **for** $1 \leq k \leq N$, $k \neq i$ **do**
12: **if** $y^k_{i,j} = 1$ and $r^k_{i,j} < \sum_{j=1}^{n_k} r^k_{k,j} - x_k$ **then**
13: Set $z^k_{i,j} = y^k_{i,j} = 0$
14: Set $x_k = x_k + r^k_{i,j}$
15: **end if**
16: **end for**
17: **end for**
18: Set $x_i = Min(x_i + \frac{\beta}{\alpha}(\sum_{k=1,k\neq i}^{N} \sum_{j=1}^{n_k} \Phi(r^k_{i,j}) - \sum_{k=1,k\neq i}^{N} \sum_{j=1}^{n_k} y^k_{i,j}), \sum_{j=1}^{n_i} r^i_{i,j})$
19: **end for**

5. DISTRIBUTED UPDATING STRATEGIES

Due to the real-time characteristic of DVEs, it is impractical to implement a centralized algorithm among all servers. Therefore, in this section, we discuss the fully distributed update strategies without any intercommunication among servers. We first give a local preference update strategy, then, we propose a more efficient remote preference update strategy.

5.1 Local Preference Update Strategy

As the name implies, in the local preference update strategy (LPUS), the request of local avatar is processed first and the relaying request from other servers will be processed at the end of each frame. Server iterates local avatars and processes the related requests for each avatar maintained by the server. The requests include updating the replicas without QoS maintained by the clients connected to the same server and sending relaying request to other servers. The received relaying requests from other servers will be processed at the end of each frame.

Algorithm 2 shows the description of LPUS for server S_i. For avatar $a_{i,j}$, $1 \leq j \leq n_i$, S_i tries to send the relaying request of $a_{i,j}$ first and then update the replicas without QoS of $a_{i,j}$ maintained by the clients connected to S_i (lines 1-18). After that, if the network bandwidth is available, S_i processes the relaying request from other servers (lines 19-27).

5.2 Remote Preference Update Strategy

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2010.8669
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8669

Algorithm 2 Description of LPUS for S_i

1: **for** Each $a_{i,j}$, $1 \leq j \leq n_i$ **do**
2: **for** Each $r^k_{i,j} > 0$, $1 \leq k \leq N$ **do**
3: **if** $k \neq i$ **then**
4: **if** $B_i - \beta > 0$ **then**
5: Send relaying request to S_k
6: $B_i = B_i - \beta$
7: **end if**
8: **else**
9: **if** $B_i - \alpha * r^i_{i,j} > 0$ **then**
10: Update $r^i_{i,j}$ local replicas without QoS
11: $B_i = B_i - \alpha * r^i_{i,j}$
12: **else**
13: Update $\frac{1}{\alpha} * B_i$ local replicas
14: **return**
15: **end if**
16: **end if**
17: **end for**
18: **end for**
19: **for** Each received relaying request $r^i_{k,j}$, $1 \leq k \leq N, k \neq i$ **do**
20: **if** $B_i - \alpha r^i_{k,j} > 0$ **then**
21: relay update packets to $r^i_{k,j}$ replicas
22: $B_i = B_i - \alpha r^i_{k,j}$
23: **else**
24: relay update packets to $\frac{1}{\alpha} B_i$ replicas
25: **return**
26: **end if**
27: **end for**

5.2.1 Update Strategy Description

We notice that, LPUS can be further improved to update more replicas without QoS when the workload is imbalanced among servers. For this reason, we designed a remote preference update strategy, denoted by RPUS, for each server. Different from LPUS, the relaying requests of the local avatars and the relaying requests from other servers are processed at the beginning of each frame. After that, if the network bandwidth is available, S_i will update the replicas without QoS of the local avatars maintained by the clients connected to S_i. The Algorithm 3 shows the pseudocode of RPUS for S_i.

Actually, RPUS is just the distributed version of the first phase of CAA. In RPUS, first, S_i sends out all the relaying requests of the local avatars to other servers and receives the relaying requests from other servers (lines 1-3). Then, S_i tries to relay the update packets to the replicas for all the relaying requests received from other servers (lines 4-12). After that, S_i updates the replicas without QoS of the local avatars maintained by the clients connected to itself according to the available resources (lines 13-17).

5.2.2 Performance Analysis

First of all, we have some definitions as follows.

- **Optimal Update Strategy**: the update strategies which can update the maximum number of replicas without QoS for the whole virtual environment.
- **Optimal Solution**: the total number of replicas without QoS updated by any optimal update strategy, denoted by N_{OPT}.

Algorithm 3 Description of RPUS for S_i

1: Send relaying request to all S_k for all $r^k_{i,j} > 0$, $1 \leq j \leq n_i$, $1 \leq k \leq N$, and $k \neq i$;
2: Receive relaying request $r^i_{k,j}$ from all S_k, $1 \leq k \leq N$, $k \neq i$ and $1 \leq j \leq n_k$;
3: $B_i = B_i - \beta \sum_{k=1,k\neq i}^{N} \sum_{j=1}^{n_i} \Phi(r^k_{i,j})$;
4: **for** each non-zero $r^i_{k,j}$ where $1 \leq k \leq N$, $k \neq i$ and $1 \leq j \leq n_k$ **do**
5: **if** $B_i - \alpha r^i_{k,j} > 0$ **then**
6: relay update packets to $r^i_{k,j}$ replicas
7: $B_i = B_i - \alpha r^i_{k,j}$
8: **else**
9: relay update packets to $\frac{1}{\alpha} B_i$ replicas;
10: **return**
11: **end if**
12: **end for**
13: **if** $B_i - \alpha \sum_{j=1}^{n_i} r^i_{i,j} > 0$ **then**
14: update $\sum_{j=1}^{n_i} r^i_{i,j}$ local replicas without QoS;
15: **else**
16: select $\frac{1}{\alpha} B_i$ local replicas to update;
17: **end if**

We first discuss the performance of S_i in the optimal solution. We use $N^{S_i}_{OPT}$ to represent the number of replicas without QoS updated by S_i in the optimal solution. We have

$$\begin{aligned} N^{S_i}_{OPT} &= x_i + \sum_{k=1,k\neq i}^{N} \sum_{j=1}^{n_k} y^i_{k,j} * z^i_{k,j} * r^i_{k,j} \\ &\leq \frac{1}{\alpha}(B_i - \beta \sum_{k=1,k\neq i}^{N} \sum_{j=1}^{n_i} y^k_{i,j}) \\ &\leq \frac{1}{\alpha} B_i \end{aligned} \tag{3}$$

according to inequality (2).

We use $N^{S_i}_{RPUS}$ to represent the number of replicas without QoS updated by S_i using RPUS. For each S_i in RPUS, after sending relaying request to other servers, the available bandwidth can be defined by

$$B_i - \beta \sum_{k=1,k\neq i}^{N} \sum_{j=1}^{n_i} \Phi(r^k_{i,j}) \tag{4}$$

under the assumption of $B_i > \beta \sum_{k=1,k\neq i}^{N} \sum_{j=1}^{n_i} \Phi(r^k_{i,j})$. The total network bandwidth requirement to update the replicas without QoS and to relay update packets from other servers to the replicas without QoS maintained by S_i is defined by

$$\alpha(\sum_{j=1}^{n_i} r^i_{i,j} + \sum_{k=1,k\neq i}^{N} \sum_{j=1}^{n_k} r^i_{k,j}) \tag{5}$$

Thus, if we use $N^{S_i}_{RPUS}$ to define the number of replicas without QoS updated by S_i in RPUS, we have

$$\begin{aligned} N^{S_i}_{RPUS} = min(&\sum_{j=1}^{n_i} r^i_{i,j} + \sum_{k=1,k\neq i}^{N} \sum_{j=1}^{n_k} r^i_{k,j}, \\ &\frac{1}{\alpha}(B_i - \beta \sum_{k=1,k\neq i}^{N} \sum_{j=1}^{n_i} \Phi(r^k_{i,j}))) \end{aligned} \tag{6}$$

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2010.8669
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8669

according to (4) and (5). If $N_{RPUS}^{S_i}$ is equal to $\frac{1}{\alpha}(B_i - \beta \sum_{k=1,k\neq i}^{N} \sum_{j=1}^{n_i} \Phi(r_{i,j}^k))$, we have:

$$N_{OPT}^{S_i} - N_{RPUS}^{S_i} \leq \frac{\alpha}{\beta} \sum_{k=1,k\neq i}^{N} \sum_{j=1}^{n_i} \Phi(r_{i,j}^k) \quad (7)$$

according to (3) and (6). Otherwise, if $N_{RPUS}^{S_i}$ is equal to $\sum_{j=1}^{n_i} r_{i,j}^i + \sum_{k=1,k\neq i}^{N} \sum_{j=1}^{n_k} r_{k,j}^i$, we have:

$$N_{OPT}^{S_i} - N_{RPUS}^{S_i} = 0 \quad (8)$$

since $y_{k,j}^i * z_{k,j}^i = 1$ if $r_{k,j}^i \neq 0$ under the optimal solution.

Based on (7) and (8), we can conclude that for the whole DVEs, the upper bound of $N_{OPT} - N_{PRUS}$ is:

$$\sum_{i=1}^{N} \frac{\alpha}{\beta} \sum_{k=1,k\neq i}^{N} \sum_{j=1}^{n_i} \Phi(r_{i,j}^k) \quad (9)$$

where N_{RPUS} is the total number of replicas without QoS updated using RPUS strategy.

6. SIMULATION AND EXPERIMENTS

We have implemented a simulator and conducted a wide range of experiments to evaluate the proposed update strategies. In this section, we will present our experimental results.

6.1 Experimental Settings

In order to reflect the real environment, we used BRITE [1] network topology generator to generate the physical topology. We used Dijkstra algorithm to calculate the network delay between each pair of nodes. For the parameter settings of BRITE, we used Router Waxman model [2] and Random Distribution for node placement to generate the topology.

Based on the physical topology generated by BRITE, we used greedy algorithm [4] to select servers. The objective of this algorithm is to achieve the minimal overall network delay between each client and server. Based on the server selection results, we calculated the network delay between each client and each server.

After setting the physical connection, we partitioned the virtual world into several fixed cells and randomly mapped the cells to different servers. After mapping, each server maintains a set of cells. For the initialization, we just randomly distributed the clients into the virtual world.

For the avatar mobility model, we adopted the Pursue Mobility Model [15] and divided the avatars into several groups. All the avatars in the same group move to the same destination. We did not use any mechanism to do load balance and implemented the first order dead-reckoning model for each remote replica.

We define the replicas without QoS which cannot be updated due to the resource limitation each frame as the remaining replicas without QoS. We use R to denote the number of remaining replicas without QoS and use it as the performance metric to evaluate the proposed algorithms. Let N_{Total} represent the total number of replicas without QoS in the whole DVE. If we use R_{OPT} to represent the number of remaining replicas without QoS in the optimal update strategy, according to (9), we can get $R_{OPT} \geq N_{Total} - N_{RPUS} - \sum_{i=1}^{N} \frac{\alpha}{\beta} \sum_{k=1,k\neq i}^{N} \sum_{j=1}^{n_i} \Phi(r_{i,j}^k)$. Therefore, we define $R^* = Max(0, N_{Total} - N_{RPUS} - \sum_{i=1}^{N} \frac{\alpha}{\beta} \sum_{k=1,k\neq i}^{N} \sum_{j=1}^{n_i} \Phi(r_{i,j}^k))$ as the lower bound of the number of remaining replicas without QoS.

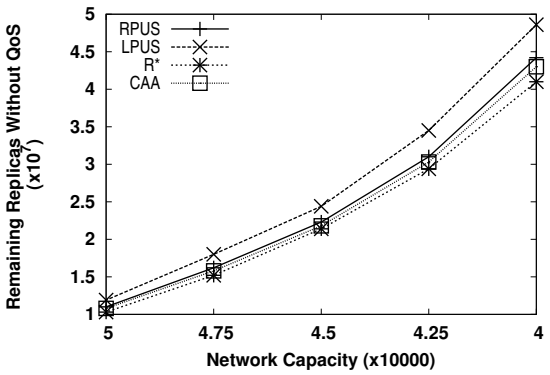


Figure 3: Impact of Network Capacity

In the following experiments, we generated 5000 nodes and selected 10 nodes as servers and rest as clients. The frame length is 0.1s and for each group of experiment, the simulation runs 10000 frames. Table 1 shows the default values of parameters which are used in the experiments.

Table 1: Parameters Default Values

Parameter	Default Value
DVE Dimension	1000x1000 (distance units)^2
Moving Speed	5 distance units/s
Max Acceleration	1 distance unit/s^2
Spatial DR Threshold	0.1 distance units
Number of Cells	100
AOI Size	20 distance units
Bandwidth Capacity	4x10^4 update packets/frame
Group Size	50 avatars/group

6.2 Experimental Results

First, we examined the impact of the network capacity. We used the same mobility trajectories for each group of experiments and decreased bandwidth from 50000 to 40000 with a step of 2500 update packets per frame. For each group of experiments, we collected R for each update strategy. As seen from Figure 3, the number of remaining replicas without QoS increases as the bandwidth capacity decreases. RPUS can reduce the number of remaining replicas without QoS substantially compared to LPUS. RPUS can also achieve approximately the same performance with CAA.

Then, we examined the impact of the group size of the mobility model. Group size represents the number of avatars for each group. The group size is increased from 1 to 100 with the step of 25. Figure 4 shows the impact of the group size of the mobility model. When the group size is 1, the mobility model becomes Random Move Mobility Model. There are seldom opportunity to have workload imbalance among servers. Therefore, bandwidth capacity is sufficient to update all the replicas for each server and R is 0 for all the update strategies. As the group size becomes larger, the avatar density will be uneven and workload distribution will be more imbalanced. Therefore, R will be increased.

Next, we examined the impact of the number of cells of the virtual world. We increased the number of cells from 25 to 400 with a step of 75. Figure 5 shows the experimental results. As the number of cells increases, the workload

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2010.8669
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8669

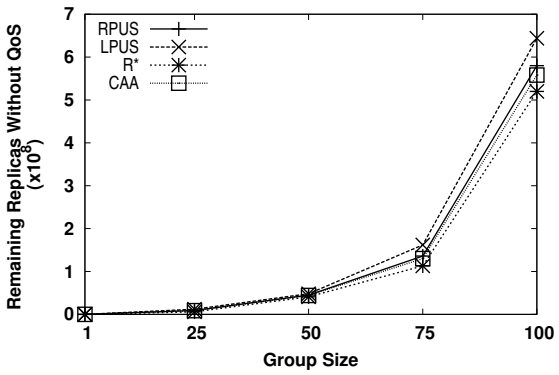


Figure 4: Impact of Group Size

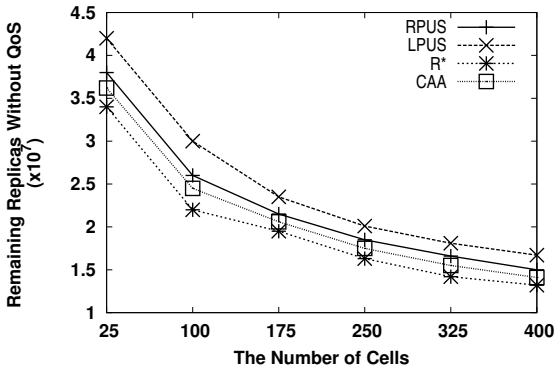


Figure 5: Impact of Cell Number

distribution among servers will be more and more balanced. Therefore, R will be reduced as the cell number increases.

At last, we examined the impact of the size of AOI. We increased the size of AOI from 10 to 30 with a step of 5. Figure 6 shows the simulation results. The total number of remaining replicas without QoS increases with the size of AOI. That is because the total number of replicas increases as the size of AOI increases. Accordingly, the number of replicas without QoS and the number of remaining replicas will increase.

RPUS generally has better performance than LPUS. The main difference between RPUS and LPUS is the processing sequence of the relaying request. In RPUS, we first send out all the relaying requests. The aim for this action is to transfer the workload to other servers. This is because load balance in the multi-server architecture cannot always be guaranteed. When one server is overloaded, other servers may be under-utilized. In addition, the bandwidth consumption for sending relaying request is small compared to those used for replica updates.

7. CONCLUSIONS AND FUTURE WORK

In this paper, we considered the consistency aware update scheduling in the multi-server architecture for DVEs. We formed the problem as an integer programming problem and discussed different update strategies. A centralized approximate algorithm (CCA) and two distributed update strategies were proposed. From the simulation results we can see that RPUS generally has better performance than

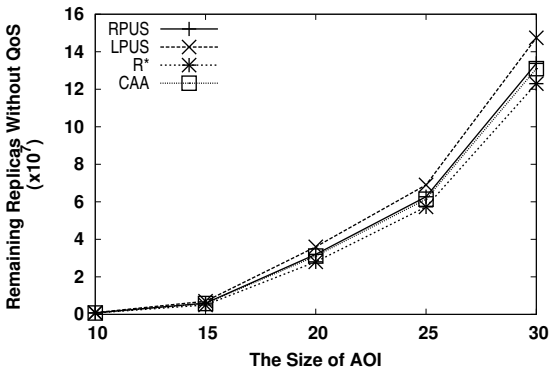


Figure 6: Impact of AOI Size

LPUR when the workload is imbalanced.

There are also many other important issues that can be considered in the future. Currently, each client is assumed to connect the server who takes charge of the game cell the client locates in. However, the network delay between the server and the client may be very large. Therefore, in the future work, the new connection strategy of the client can be considered. In order to update some important replicas without QoS timely, the importance factor of each avatar can be considered.

8. REFERENCES

[1] Alberto Medina, Ibrahim Matta, John Byers,"BRITE: A Flexible Generator of Internet Topologies", *Technical Report: 2000-005*, 2000.

[2] B.M. Waxman,"Routing of Multipoint Connections", *IEEE Journal on Selected Areas in Communications*, Vol. 6, No. 9, pp. 1617-1622, December 1988.

[3] Daniel Van, Hook Steven, J. Rak James, O. Calvin, "Approaches to Relevance Filtering", *Eleventh Workshop on Standards for the Interoperability of Distributed Simulations*, 1994.

[4] Duong Ta, Suiping Zhou, Wentong Cai, Xueyan Tang, Rassul Ayani, "Network-Aware Server Placement for Highly Interactive Distributed Virtual Environments", *Proceedings of the 2008 12th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications*, pp. 95-102, 2008.

[5] Jeremy Brun, Farzad Safaei, Paul Boustead, "Managing Latency and Fairness in Networked Games", *Commun. ACM*, Vol. 49, No. 11, pp. 46-51, 2006.

[6] John C. S. Lui, M. F. Chan, "An Efficient Partitioning Algorithm for Distributed Virtual Environment Systems", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 13, No. 3, pp. 193-211, 2002.

[7] Kuan-Ta Chen, Polly Huangm, Chin-Laung Lei,"Effect of Network Quality on Player Departure Behavior in Online Games", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 2, No. 5, pp. 593-606, 2009.

[8] Lothar Pantel, Lars C. Wolf, "On the Suitability of Dead Reckoning Schemes for Games", *Proceedings of the 1st workshop on Network and system support for games*, pp. 79-84, 2002.

[9] Mostafa, A. Bassiouni, Ming-Hsing, Chiu Margaret, Loper Michael, Garnsey Jim, Williams, "Performance

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2010.8669
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8669

and reliability analysis of relevance filtering for scalable distributed interactive simulation", *ACM Transactions on Modeling and Computer Simulation*, Vol. 7, No. 3, pp. 293-331, 1997.

[10] M. R. Stytz, "Distributed Virtual Environments", Vol. 16, No. 3, pp. 19-31, *IEEE Computer Graphics and Applications*, 1996.

[11] P. Morillo, S. Rueda, J. M. Orduna, J. Duato, "A Latency-Aware Partitioning Method for Distributed Virtual Environment Systems", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 18, No. 9, pp. 1215-1226, 2007.

[12] Sandeep Singhal, Michael Zyda, "Networked Virtual Environments: Design and Implementation", *ACM Press/Addison-Wesley Publishing Co*, 1999.

[13] Suiping Zhou, Wentong Cai, Bu-Sung Lee, Stephen J. Turner, "Time-space Consistency in Large-scale Distributed Virtual Environments", *ACM Transactions on Modeling and Computer Simulation*, Vol. 14, No. 1, pp. 31-44, 2004.

[14] Takahiro Yasui, Yutaka Ishibashi, Tomohito Ikedo, "Influences of Network Latency and Packet Loss on Consistency in Networked Racing Games", *Proceedings of NetGames'05*, pp. 1-8, 2005.

[15] T Camp, J Boleng, V Davies, "A Survey of Mobility Models for Ad Hoc Network Research", *Wireless Communications and Mobile Computing*, Vol. 2, pp. 483-502, 2002.

[16] Yi Zhang, Ling Chen, Gencai Chen, "Globally Synchronized Dead-reckoning with Local Lag for Continuous Distributed Multiplayer Games", *Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games*, No. 7, 2006.

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2010.8669
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2010.8669